SYNOPSYS®

DesignWare DW_apb_uart Databook

DW_apb_uart – Product Code

4.00a
June 2015

Copyright Notice and Proprietary Information Notice

Destination Control Statement

All technical data contained in this publication is subject to the export control laws of the United States of America. Disclosure to nationals of other countries contrary to United States law is prohibited. It is the reader's responsibility to determine the applicable regulations and to comply with them.

Disclaimer

SYNOPSYS, INC., AND ITS LICENSORS MAKE NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.

Trademarks

Synopsys and certain Synopsys product names are trademarks of Synopsys, as set forth at http://www.synopsys.com/Company/Pages/Trademarks.aspx.

All other product or company names may be trademarks of their respective owners.

Synopsys, Inc.
690 E. Middlefield Road
Mountain View, CA 94043

www.synopsys.com

Contents

Revision History

This table shows the revision history for the databook from release to release. This is being tracked from version 3.06b onward.

Version	Date	Description
4.00a	June 2015	■ Added following section in "Functional Description": - 9-bit Data Transfer - RS485 Serial Protocol - Fractional Baud Rate Support ■ Updated Registers, Signal Descriptions, Parameters, and Programming the DW_apb_uart chapters ■ Added "Running SpyGlass® Lint and SpyGlass® CDC" on page 28 ■ Added "Running Spyglass on Generated Code with coreAssembler" on page 36 ■ Added Chapter 7, "Internal Parameter Descriptions" ■ "Signal Descriptions" on page 97 auto-extracted from the RTL ■ Added Appendix A, "Synchronizer Methods" ■ Updated area and performance number in sections "Area" on page 202 and "Power Consumption" on page 203
3.15a	June 2014	■ Version change for 2014.06a release ■ Added "Performance" section in the "Integration Considerations" chapter ■ Corrected Default Input/Output Delay in Signals chapter
3.14c	May 2013	■ Corrected the de-assert sequence in "Reset Signals" on page 67 ■ Corrected the label of the UART_ADD_ENCODED_PARAMS parameter ■ Updated the template
3.14b	Sep 2012	Added the product code on the cover and in Table 1-1.
3.14b	Jun 2012	Added new RTC_FCT coreConsultant parameter.
3.13a	Mar 2012	■ Enhanced timing information for serial clock modules ■ Corrected reset values for MSR[3:0] bits ■ Added note to write MCR before LCR for SIR mode ■ Updated CPR register description
3.12c	Nov 2011	Version change for 2011.11a release.

(Continued)

Version	Date	Description
3.12b	Oct 2011	■ Updated DLAB bit description of the LCR register ■ Added flow charts in programming chapter ■ Edited “Product Overview” material and “Functional Description” material for better flow in reading ■ Enhanced SIRE bit description of MCR register
3.12a	Jun 2011	■ Updated material for Stick Parity bit of Line Control Register ■ Updated system diagram in Figure 1-1 ■ Enhanced “Related Documents” section in Preface ■ Corrected address offset and R/W for LPDLL, LPDLH, and DMASA registers
3.11a	12 Apr 2011	Added note for break condition in BI bit of LSR register.
3.11a	Apr 2011	■ Corrected description of APB_DATA_WIDTH parameter ■ Added sections for “Potential Deadlock Conditions in DW_apb_uart/DW_ahb_dmac Systems” and “Reset Signals” ■ Edited descriptions for Parity Error and Framing Error bits in LSR register ■ Corrected dma* signals in Figure 3-25 ■ Added register in acknowledge page in “RTL Diagram of Data Synchronization Module” diagram
3.10a	25 Jan 2011	Corrected description of reset operation in Answer 7 of “Application Notes”
3.10a	Jan 2011	Corrected “DW_apb_uart Testbench” illustration
3.10a	Nov 2010	■ Corrected DW_ahb_dmac response in “Receive Watermark Level and Receive FIFO Overflow” section ■ Modified values to which APB_DATA_WIDTH parameter can be set
3.10a	9 Sep 2010	Corrected LCR link to LSR link in RBR register description.
3.10a	Sep 2010	■ Enhanced USR[0] busy description to explain non-busy conditions ■ Corrected names of include files and vcs command used for simulation ■ Added information regarding false start bit detection ■ Updated the ADDITIONAL_PARAMETERS description
3.08a	Jun 2010	■ Corrected synchronous description from “pclk” to “N/A” for cts_n, dsr_n, dcd_n and ri_n signals ■ Added: - Syntax for include files in database tables - +v2k option in vcs command syntax - Information for back-to-back character stream transmission
3.08a	Jan 2010	Revised FCR register description for condition in which FIFOs are not implemented
3.08a	Dec 2009	Updated databook to new template for consistency with other IIP/VIP/PHY databooks
3.08a	Jul 2009	Corrected equations for avoiding underflow when programming a source burst transaction

(Continued)

Version	Date	Description
3.08a	May 2009	Removed references to QuickStarts, as they are no longer supported
3.08a	Apr 2009	Enhanced “Clock Support” section
3.08a	Oct 2008	Version change for 2008.10a release
3.07b	Jun 2008	Version change for 2008.06a release
3.07a	Jan 2008	■ Updated for revised installation guide and consolidated release notes titles ■ Changed references of “Designware AMBA” to simply “DesignWare” ■ Performance information temporarily removed ■ Corrections made to Figures 7 and 8
3.07a	Dec 2007	■ Correction of Figure 8 ■ Removed area tables pending more current data
3.06b	Jun 2007	Version change for 2007.06a release.

Preface

This databook provides information that you need to interface the DW_apb_uart component to the Advanced Peripheral Bus (APB). This component conforms to the *AMBA Specification, Revision 2.0* from ARM.

The information in this databook includes a functional description, pin and parameter descriptions, and a memory map. Also provided are an overview of the component testbench, a description of the tests that are run to verify the component, and synthesis information.

Organization

The chapters of this databook are organized as follows:

- Chapter 1, "Product Overview" provides a system overview, a component block diagram, basic features, and an overview of the verification environment.
- Chapter 2, "Building and Verifying a Component or Subsystem" introduces you to using the DW_apb_uart within the coreAssembler and coreConsultant tools.
- Chapter 3, "Functional Description" describes the functional operation of the DW_apb_uart.
- Chapter 4, "Parameters" identifies the configurable parameters supported by the DW_apb_uart.
- Chapter 5, "Signals" provides a list and description of the DW_apb_uart signals.
- Chapter 6, "Registers" describes the programmable registers of the DW_apb_uart.
- Chapter 7, "Internal Parameter Descriptions", "Registers" describes the programmable registers of the DW_apb_uart.
- Chapter 8, "Programming the DW_apb_uart" provides information needed to program the configured DW_apb_uart.
- Chapter 9, "Verification" provides information on verifying the configured DW_apb_uart.
- Chapter 10, "Integration Considerations" includes information you need to integrate the configured DW_apb_uart into your design.
- Appendix A, "Synchronizer Methods" documents the synchronizer methods (blocks of synchronizer functionality) used in DW_abp_uart to cross clock boundaries.

- Appendix B, "Application Notes" includes information you need to integrate the configured DW_apb_uart into your design.
- Appendix C, "Glossary" provides a glossary of general terms.

Related Documentation

- *DW_apb_uart Driver Kit User Guide* – Contains information on the Driver Kit for the DW_apb_uart; requires source code license (DWC-APB-Periph-Source)
- *Using DesignWare Library IP in coreAssembler* – Contains information on getting started with using DesignWare SIP components for AMBA 2 and AMBA 3 AXI components within coreTools
- *coreAssembler User Guide* – Contains information on using coreAssembler
- *coreConsultant User Guide* – Contains information on using coreConsultant

To see a complete listing of documentation within the DesignWare Synthesizable Components for AMBA 2, refer to the *Guide to Documentation for DesignWare Synthesizable Components for AMBA 2 and AMBA 3 AXI*.

Note

Information on the DW_apb_uart component in this databook assumes that the reader is fully familiar with the National Semiconductor 16550 (UART) component specification. This specification can be obtained on the web at:

http://www.national.com/ds/PC/PC16550D.pdf

Information provided on IrDA SIR mode assumes that the reader is fully familiar with the IrDa Serial Infrared Physical Layer Specification. This specification can be obtained from the following website:

http://www.irda.org

Web Resources

- DesignWare IP product information: http://www.designware.com
- Your custom DesignWare IP page: http://www.mydesignware.com
- Documentation through SolvNet: http://solvnet.synopsys.com (Synopsys password required)
- Synopsys Common Licensing (SCL): http://www.synopsys.com/keys

Customer Support

To obtain support for your product:

- First, prepare the following debug information, if applicable:
  - For environment setup problems or failures with configuration, simulation, or synthesis that occur within coreConsultant or coreAssembler, use the following menu entry:

    File > Build Debug Tar-file

    Check all the boxes in the dialog box that apply to your issue. This menu entry gathers all the Synopsys product data needed to begin debugging an issue and writes it to the file <*core tool startup directory*>/debug.tar.gz.

- For simulation issues outside of coreConsultant or coreAssembler:
  - Create a waveforms file (such as VPD or VCD)
  - Identify the hierarchy path to the DesignWare instance
  - Identify the timestamp of any signals or locations in the waveforms that are not understood

- Then, contact Support Center, with a description of your question and supplying the above information, using one of the following methods:
  - *For fastest response*, use the SolvNet website. If you fill in your information as explained below, your issue is automatically routed to a support engineer who is experienced with your product. The **Sub Product** entry is critical for correct routing.

    Go to http://solvnet.synopsys.com/EnterACall and click on the link to enter a call.
    Provide the requested information, including:
    - **Product:** DesignWare Library IP
    - **Sub Product:** AMBA
    - **Tool Version:** <*product version number*>
    - **Problem Type:**
    - **Priority:**
    - **Title:** DW_apb_uart
    - **Description:** For simulation issues, include the timestamp of any signals or locations in waveforms that are not understood

    After creating the case, attach any debug files you created in the previous step.
  - Or, send an e-mail message to support_center@synopsys.com (your email will be queued and then, on a first-come, first-served basis, manually routed to the correct support engineer):
    - Include the Product name, Sub Product name, and Tool Version number in your e-mail (as identified above) so it can be routed correctly.
    - For simulation issues, include the timestamp of any signals or locations in waveforms that are not understood
    - Attach any debug files you created in the previous step.
  - Or, telephone your local support center:
    - North America:

      Call 1-800-245-8005 from 7 AM to 5:30 PM Pacific time, Monday through Friday.
    - All other countries:

      http://www.synopsys.com/Support/GlobalSupportCenters

Product Code

Table 1-1 lists all the components associated with the product code for DesignWare APB Advanced Peripherals.

Table 1-1 DesignWare APB Advanced Peripherals – Product Code: 3772-0

Component Name	Description
DW_apb_i2c	A highly configurable, programmable master or slave i2c device with an APB slave interface
DW_apb_i2s	A configurable master or slave device for the three-wire interface (I2S) for streaming stereo audio between devices
DW_apb_ssi	A configurable, programmable, full-duplex, master or slave synchronous serial interface
DW_apb_uart	A programmable and configurable Universal Asynchronous Receiver/Transmitter (UART) for the AMBA 2 APB bus

1 Product Overview

The DW_apb_uart is a programmable Universal Asynchronous Receiver/Transmitter (UART). This component is an AMBA 2.0-compliant Advanced Peripheral Bus (APB) slave device and is part of the family of DesignWare Synthesizable Components.

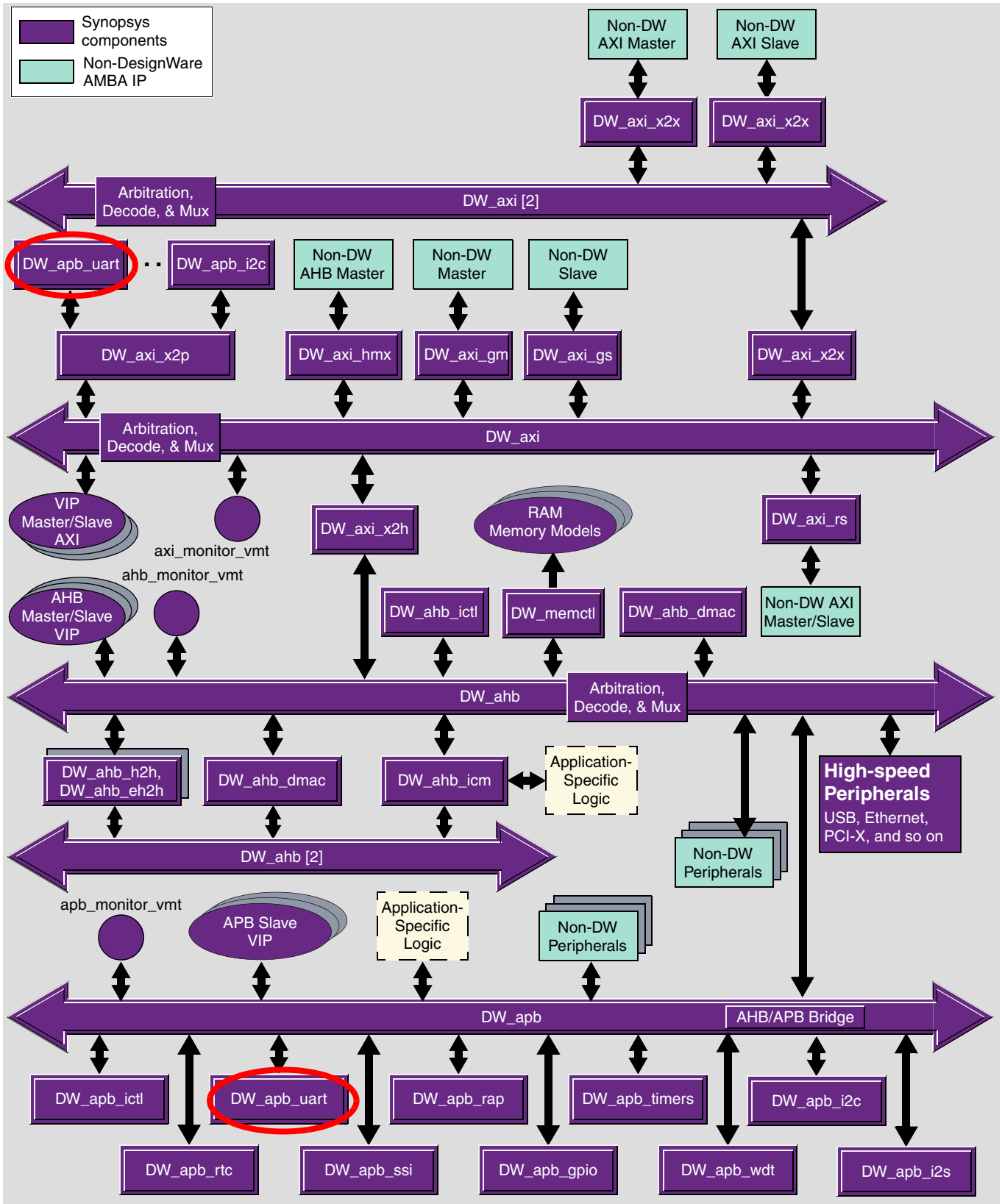
1.1 DesignWare System Overview

The Synopsys DesignWare Synthesizable Components environment is a parameterizable bus system containing AMBA version 2.0-compliant AHB (Advanced High-performance Bus) and APB (Advanced Peripheral Bus) components, and AMBA version 3.0-compliant AXI (Advanced eXtensible Interface) components.

Figure 1-1 illustrates one example of this environment, including the AXI bus, the AHB bus, and the APB bus. Included in this subsystem are synthesizable IP for AXI/AHB/APB peripherals, bus bridges, and an AXI interconnect and AHB bus fabric. Also included are verification IP for AXI/AHB/APB master/slave models and bus monitors. In order to display the databook for a DW_* component, click on the corresponding component object in the illustration.

Attention Links resolve only if you are viewing this databook from your $DESIGNWARE_HOME tree, and to only those components that are installed in the tree.

Figure 1-1 Example of DW_apb_uart in a Complete System



You can connect, configure, synthesize, and verify the DW_apb_uart within a DesignWare subsystem using coreAssembler, documentation for which is available on the web in the *coreAssembler User Guide*.

If you want to configure, synthesize, and verify a single component such as the DW_apb_uart component, you might prefer to use coreConsultant, documentation for which is available in the *coreConsultant User Guide*.

1.2 General Product Description

The DW_apb_uart is modeled after the industry-standard 16550. However, the register address space is relocated to 32-bit data boundaries for APB bus implementation. The DW_apb_uart can be configured, synthesized, and verified using the Synopsys coreConsultant GUI.

The DW_apb_uart is used for serial communication with:

- Peripherals
- Modems (data carrier equipment, DCE)
- Data sets

Data is written from a master (CPU) over the APB bus to the UART, and it is converted to serial form and transmitted to the destination device. Serial data is also received by the UART and stored for the master (CPU) to read back.

The DW_apb_uart contains registers that control:

- Character length
- Baud rate
- Parity generation/checking
- Interrupt generation

Although there is only one interrupt output signal (intr) from the DW_apb_uart, there are several prioritized interrupt types that can be responsible for its assertion. Each of the interrupt types can be separately enabled or disabled by the control registers.

The following describe various functionalities that you can configure into the DW_apb_uart:

- Transmit and receive data FIFOs – To reduce the time demand placed on the master by the DW_apb_uart, optional FIFOs are available to buffer transmit and receive data. The master does not have to access the DW_apb_uart each time a single byte of data is received. The optional FIFOs can be selected at configuration time.

  The FIFOs can be configured as external customer-supplied FIFO RAMs or as internal DesignWare D-flip-flop-based RAMs (DW_ram_r_w_s_dff).

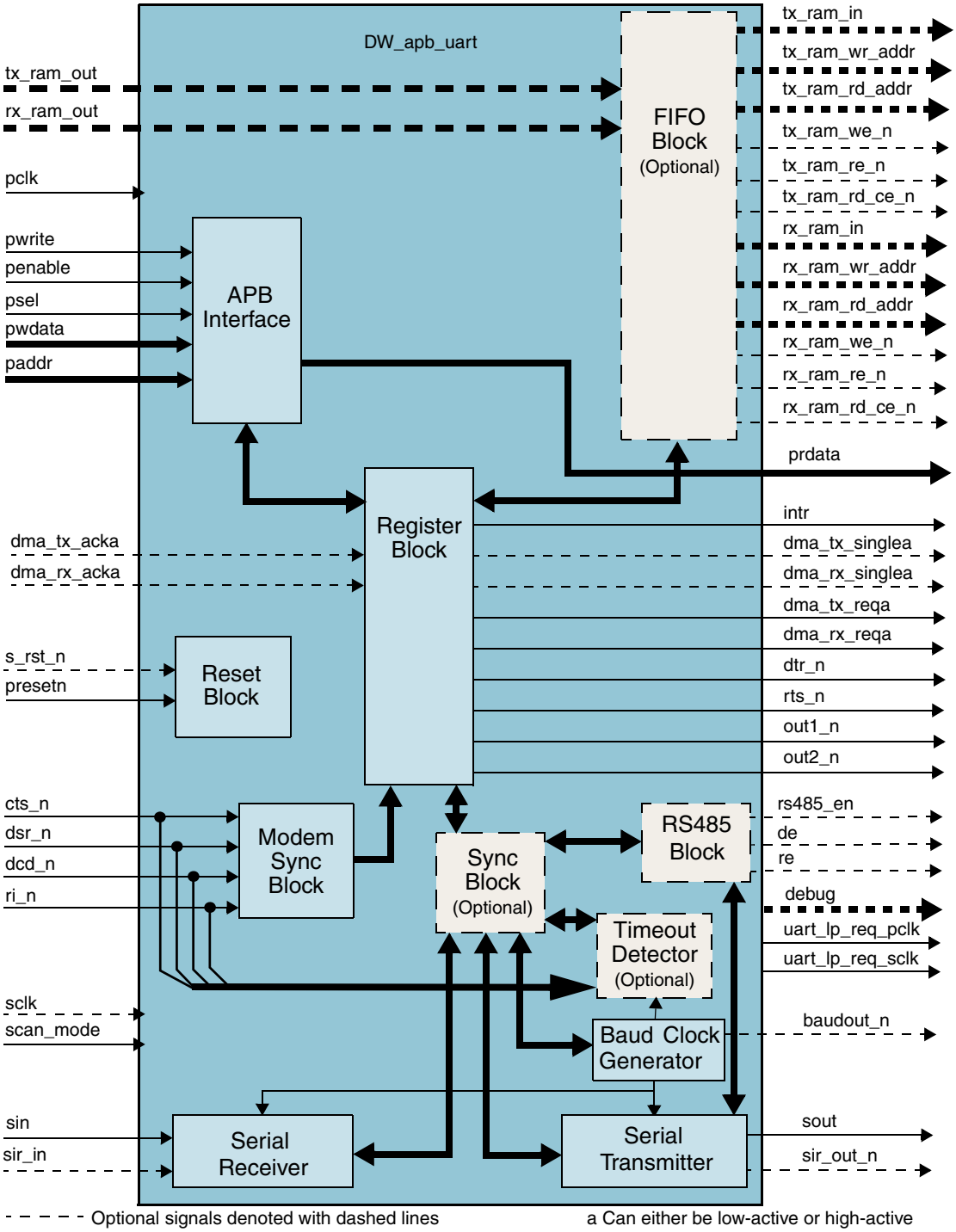
  - When external RAM support is chosen, both synchronous or asynchronous read-port memories are supported.
  - When FIFO support is selected, an optional test/debug mode is available to allow the receive FIFO to be written by the master and the transmit FIFO to be read by the master.

- DMA controller interface – The DW_apb_uart can interface with a DMA controller through external signals (dma_tx_req_n and dma_rx_req_n) in order to indicate when data is ready to be read or when the transmit FIFO is empty. Additional optional DMA signals are available for compatibility with a DesignWare DMA controller interface, such as the DW_ahb_dmac.
- Asynchronous clock support – To solve problems surrounding CPU data synchronization in relation to the required serial baud clock requirements, an optional separate serial data clock can be selected. Full handshaking and level-synchronization guarantees all data crossing between the two clock domains.
- Auto flow control – The DW_apb_uart uses a 16750-compatible Auto Flow Control Mode to increase system efficiency and decrease software load. When FIFOs and the Auto Flow Control are selected and enabled, the request-to-send (rts_n) output and clear-to-send (cts_n) input automatically control serial data flow.
- RS485 interface Support - For integration into systems for which an RS485 interface is required, the DW_apb_uart can be configured for a software-programmable RS485 mode. If this mode is not selected, only the UART (RS232 standard) serial data format is available.
- Programmable Transmit Holding Register Empty (THRE) interrupt – The DW_apb_uart uses a Programmable Transmitter Holding Register Empty (THRE) Interrupt Mode to increase system performance. When FIFOs and the THRE Mode are selected and enabled, THRE Interrupts are active at or below a programmed TX FIFO threshold level. Additionally, the Line Status THRE switches from indicating TX FIFO empty to TX FIFO full, which allows software to set a threshold that keeps the transmitter FIFO from running empty whenever there is data to transmit.
- Serial infrared support – For integration in systems where Infrared SIR serial data format is required, the DW_apb_uart can be configured for a software-programmable IrDA SIR Mode. If this mode is not selected, only the UART (RS232 standard) serial data format is available.
- Increase built-in diagnostic capabilities – To increase the built-in diagnostic capabilities of the DW_apb_uart, the Modem Control Loopback Mode has been extended. Modem Status bits actually reflect Modem Control Register deltas, as well as the bits themselves. Additionally, when FIFOs and Auto Flow Control Mode are selected and enabled, the Modem Control RTS is internally looped back to the CTS in order to control the transmitter, which allows local testing of the Auto CTS mode. Furthermore, the controllability of rts_n through the receiver FIFO threshold can be observed using the RTS Modem Status bit, which allows local verification of the Auto RTS mode.
- Level 1 and Level 2 debug support – To help with debug issues, optional debug signals are available on the DW_apb_uart. To comply with level 1 and level 2 debug support requirements, many internal points of interest to the debugger are available as outputs.

1.2.1 DW_apb_uart Block Diagram

Figure 1-2 illustrates the DW_apb_uart block diagram.

Figure 1-2 DW_apb_uart Functional Block Diagram



The following list describes each of the major blocks shown in Figure 1-2:

- **Reset block** – resets clock domains.
- **APB slave interface** – connects to APB bus.
- **Register block** – responsible for the main UART functionality including control, status and interrupt generation.
- **Modem Synchronization block** – synchronizes the modem input signal.
- **FIFO block** (optional) – responsible for FIFO control and storage—when using internal RAM—or optionally signaling to control external RAM.
- **Synchronization block** (optional) – implemented when the peripheral is configured to have a separate serial data clock (i.e. two clock implementation).
- **Timeout Detector block** (optional) – indicates the absence of character data movement in the receiver FIFO within a given time period; this is used to generate character timeout interrupts when enabled.

  This block can also have optional clock gate enable outputs—uart_lp_req_pclk for single clock implementations or uart_lp_req_pclk and uart_lp_req_sclk for two clock implementations—in order to indicate:

  - TX and RX pipeline is clear; that is, there is no data
  - No activity has occurred
  - Modem control input signals have not changed within a given time period
- **Baud Clock Generator** – produces the transmitter and receiver baud clock along with the output reference clock signal (baudout_n).
- **Serial Transmitter** – converts the parallel data—written to the UART—into serial form and adds all additional bits, as specified by the control register, for transmission. These serial data, referred to as a character, can exit the block in two formats:

  - Serial UART
  - IrDA 1.0 SIR
- **Serial Receiver** – converts the serial data character—specified by the control register—received in either the UART or IrDA 1.0 SIR format to parallel form. This block controls:

  - Parity error detection
  - Framing error detection
  - Line break detection
- **RS485 block** (optional) – implemented when the peripheral is configured to have an RS485 interface, responsible for the generation of driver enable (de) and receiver enable (re) signals required by the RS485 Transceiver.

1.3 Features

- AMBA APB interface allows integration into AMBA SoC implementations
- 9-bit serial data support
- False start bit detection
- Programmable fractional baud rate support
- Multi-drop RS485 interface support
- Configurable parameters for the following:
  - APB data bus widths of 8, 16 and 32
  - Additional DMA interface signals for compatibility with DesignWare DMA interface
  - DMA interface signal polarity
  - Transmit and receive FIFO depths of 0, 16, 32, 64, 128, 256, 512, 1024, 2048
  - Internal or external FIFO (RAM) selection
  - Use of two clocks—pclk and sclk—instead of just pclk
  - IrDA 1.0 SIR mode support with up to 115.2 Kbaud data rate and a pulse duration (width) as specified in the IrDA physical layer specification:

    $\text{width} = 3/16 \times \textit{bit period}$
  - IrDA 1.0 SIR low-power reception capabilities
  - Baud clock reference output signal
  - Clock gate enable output(s) used to indicate that the TX and RX pipeline is clear (no data) and no activity has occurred for more than one character time, so that clocks can be gated
  - FIFO access mode—for FIFO testing—enabling the master to write to the receive FIFO and read from the transmit FIFO
  - Additional FIFO status registers
  - Shadow registers to reduce software overhead and also include a software programmable reset
  - Auto Flow Control mode, as specified in the 16750 standard
  - Loopback mode that enables greater testing of Modem Control and Auto Flow Control features (Loopback support in IrDA SIR mode is available)
  - Transmitter Holding Register Empty (THRE) interrupt mode
  - Busy functionality
- Ability to set some configuration parameters during instantiation
- Configuration identification registers present

- Functionality based on the 16550 industry standard
  - Programmable character properties, such as:
    - Number of data bits per character (5-8)
    - Optional parity bit (with odd, even select or Stick Parity)
    - Number of stop bits (1, 1.5 or 2)
  - Line break generation and detection
  - DMA signaling with two programmable modes
  - Prioritized interrupt identification
- Programmable FIFO enable/disable
- Programmable serial data baud rate as calculated by the following:

  baud rate = (*serial clock frequency*)/(16×*divisor*)
- External read enable signal for RAM wake-up when using external RAMs
- Modem and status lines are independently controlled
- Separate system resets for each clock domain to prevent metastability
- Complete RTL version

1.4 Standards Compliance

The DW_apb_uart component conforms to the *AMBA Specification, Revision 2.0* from ARM. Readers are assumed to be familiar with this specification.

Information on the DW_apb_uart component in this databook assumes that the reader is fully familiar with the National Semiconductor 16550 (UART) component specification. This specification can be obtained on the web at:

http://www.national.com/pf/PC/PC16550D.html#Datasheet

Information provided on IrDA SIR mode assumes that the reader is fully familiar with the IrDa Serial Infrared Physical Layer Specification. This specification can be obtained from the following website:

http://www.irda.org

1.5 Speed and Clock Requirements

The DW_apb_uart has been synthesized and simulated with a pclk of 166 Mhz in 28nm technology. It met timing requirements at these speeds. The sclk signal was set to 25 MHz with a baud devisor of 1 to give a max baud rate of just over 1.5 M. This is the baud rate referred to in the National 16550 specification.

1.6 Verification Environment Overview

The DW_apb_uart is put through a verification process which utilizes constrained randomized testing (or CRT). This process is divided into several “groups” – for testing of the DW_apb_uart’s hardware associated with the transmit, receive, loopback and debug. Under normal verification runs, the test group selected is

randomly chosen for a given DW_apb_uart hardware configuration, although some amount of user-controlled selection is possible.

Under each group of tests, two more levels of randomization of the test stimulus are applied – one at the higher "system" level associated with nature of the test chosen, and one at the "parametric" level associated with the DW_apb_uart's registers. In doing so, control and/or intervention of/in the verification process and scope by the user is reduced to a minimum.

The "system" level of randomization ensures that the DW_apb_uart is, for example, injected with a varying number of characters of arbitrary contents, as well as the type and number of character corruptions applied.

The "parametric" level of randomization applied to the DW_apb_uart ensures that the DW_apb_uart's hardware is programmed as arbitrarily as possible; for example, the line settings for the characters exchanged during simulations, the varying patterns for the interrupt enables, as well as the various transmit/receive trigger thresholds.

Once the required set of randomized "system" and "parametric" variables are obtained, three separate groups of testcode are kicked off concurrently – one for the generating of the stimulus for the DW_apb_uart and supporting models; one for the overall environment support, such as scoreboarding, messaging, signal transition detections, etc.; and lastly, one for the checkers.

To support the serial exchanges of characters, in both the IrDA and normal transfer modes, VERA models in the SIO VIP are used. Two instances of both the SIOTxrx and the SIOMonitor models assist in verifying that the DW_apb_uart's hardware functionalities.

To support DMA-controlled transfers to and from the DW_apb_uart, an instance of a AHB DMA BFM is also included. This acts as an independent AHB master issuing AHB transfer commands separately from the AHB master model used to control the DW_apb_uart.

1.7 Licenses

Before you begin using the DW_apb_uart, you must have a valid license. For more information, refer to "Licenses" in the *DesignWare Synthesizable Components for AMBA 2/AMBA 3 AXI Installation Guide.*

1.8 Where To Go From Here

At this point, you may want to get started working with the DW_apb_uart component within a subsystem or by itself. Synopsys provides several tools within its coreTools suite of products for the purposes of configuration, synthesis, and verification of single or multiple synthesizable IP components—coreConsultant and coreAssembler. For information on the different coreTools, refer to *Guide to coreTools Documentation*.

For more information about configuring, synthesizing, and verifying just your DW_apb_uart component, refer to "Overview of the coreConsultant Configuration and Integration Process" on page 26.

For more information about implementing your DW_apb_uart component within a DesignWare subsystem using coreAssembler, refer to "Overview of the coreAssembler Configuration and Integration Process" on page 32.

2 Building and Verifying a Component or Subsystem

DesignWare Synthesizable IP (SIP) components for AMBA 2 and AMBA 3 AXI are packaged using Synopsys coreTools, which enable the user to configure, synthesize, and run simulations on a single SIP title, or to build a configured AMBA subsystem. You do this by generating a workspace view using one of the following coreTools applications:

- coreConsultant – Used for configuration, RTL generation, synthesis, and execution of packaged verification for a single SIP title. The *coreConsultant User Guide* provides complete information on using coreConsultant.
- coreAssembler – Used for building and configuration of a subsystem that connects multiple SIP titles, RTL generation, synthesis, and creation of a template subsystem testbench. The *coreAssembler User Guide* provides complete information on using coreAssembler.

A workspace is your working version of a DesignWare SIP component or subsystem. In fact, you can create several workspaces to experiment with different design alternatives.

Hint If you are unfamiliar with coreTools—which is comprised of the coreAssembler, coreConsultant, and coreBuilder tools—you can go to *Using DesignWare Library IP in coreAssembler* to "get started" learning how to work with DesignWare SIP components.

2.1 Setting up Your Environment

The DW_apb_uart is included in a release of DesignWare SIP components. It is assumed that you have already downloaded and installed the release. If you have not, you can download and install the latest versions of required tools using the *DesignWare Synthesizable Components for AMBA 2/AMBA 3 AXI Installation Guide*.

You also need to set up your environment correctly using specific environment variables, such as DESIGNWARE_HOME, VERA_HOME, PATH, and SYNOPSYS. If you are not familiar with these requirements and the necessary licenses, refer to "Setting up Your Environment" in the *DesignWare Synthesizable Components for AMBA 2/AMBA 3 AXI Installation Guide.*

2.2 Overview of the coreConsultant Configuration and Integration Process

Once you have correctly downloaded and installed a release of DesignWare SIP components and then set up your environment, you can begin work on the DW_apb_uart using coreConsultant.

2.2.1 coreConsultant Usage

Figure 2-1 illustrates some general directories and files in a coreConsultant workspace.

Figure 2-1 coreConsultant Usage Flow

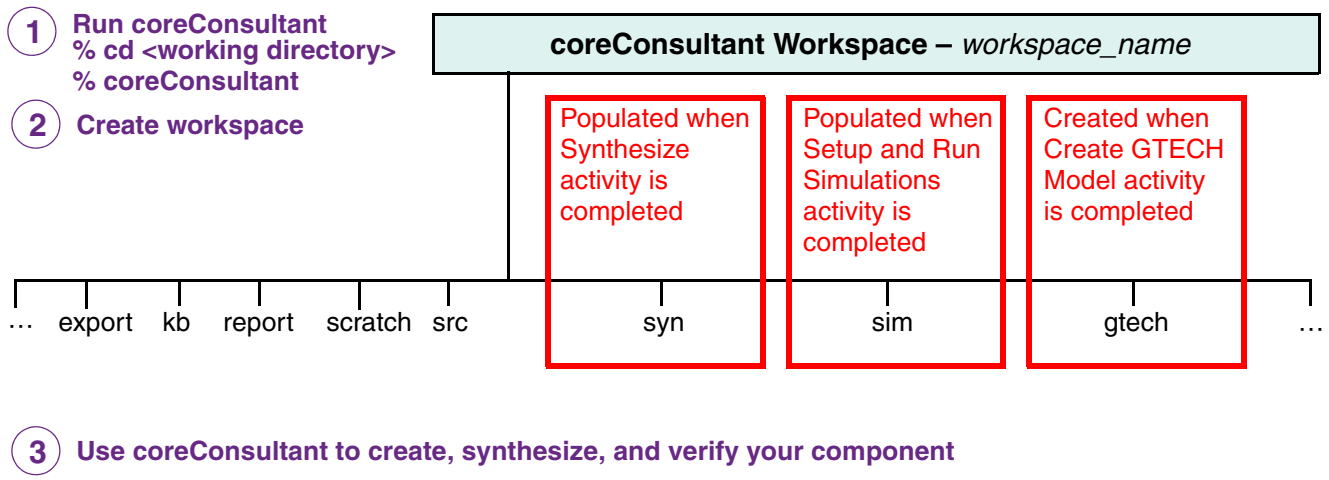


Table 2-1 provides a description of the implementation workspace directory and subdirectories.

Table 2-1 coreConsultant Implementation Workspace Directory Contents

Directory/Subdirectory	Description
auxiliary	Scripts and text files used by coreConsultant. Generated upon first creating workspace.
custom	Contains RTL preprocessor scripts. Generated during Specify Configuration activity.
doc	Contains local copies of component-specific databooks. Generated upon first creating workspace.
export	Contains files used to integrate results from the completed source configuration and synthesis activities into your design (outside coreConsultant). Generated upon first creating workspace; populated during Specify Configuration activity.
gtech	Contains synthesis scripts and output netlists from gtech generation; also used for RTL simulation of encrypted source code. Generated during Generate GTECH Model activity.

Table 2-1 coreConsultant Implementation Workspace Directory Contents (Continued)

Directory/Subdirectory	Description
kb	Contains knowledge base information used by coreConsultant. These are binary files containing the state of the design. Generated upon first creating workspace; populated and updated throughout activities.
leda	Contains Leda configuration files for the component. Generated upon first creating workspace; updated during Run Leda Coding Checker activity.
report	Contains all of the reports created by coreConsultant during build, configuration, test and synthesis phases. An index.html file in this directory links to many of these generated reports. Generated upon first creating workspace; populated and updated throughout activities.
scratch	Contains temp files used during the coreConsultant processes. Generated upon first creating workspace; populated and updated throughout activities.
sim	Contains test stimulus and output files. Generated upon first creating workspace; updated during Setup and Run Simulations activity.
spyglass	Contains SpyGlass Lint and CDC configuration files for the component. Generated upon first SpyGlass run; updated during Run Spyglass RTL Checker activity.
src	Includes the top-level RTL file, *design_name*.v. If you have a source license, this will contain plain-text RTL; if you only have a designware license, this will contain encrypted RTL. Generated upon first creating workspace; populated during Specify Configuration activity.
syn	Contains synthesis files for the component. Generated upon first creating workspace; updated during Synthesis activity and Formal Verification activity.
tcl	Contains synthesis intent scripts. Generated upon first creating workspace.

For details on some key files created during coreConsultant activities, refer to "Database Files" on page 36.

For information on using coreConsultant, refer to the *coreConsultant User Guide*.

2.2.2 Configuring the DW_apb_uart within coreConsultant

The "Parameters" chapter on page 89 describes the DW_apb_uart hardware configuration parameters that you configure using the coreConsultant GUI.

The "Creating the RTL View of a Core" chapter in the *coreConsultant User Guide* discusses how to specify a configuration for an individual component like the DW_apb_uart.

2.2.3 Creating Gate-Level Netlists within coreConsultant

The "Creating the Gate-Level Netlist for a Core" chapter in the *coreConsultant User Guide* discusses how to create a translation of the RTL view into a technology-specific netlist for an individual component like the DW_apb_uart.

2.2.4 Verifying the DW_apb_uart within coreConsultant

The "Verification" chapter on page 187 provides an overview of the testbench available for DW_apb_uart verification using the coreConsultant GUI.

The "Verifying Your Implementation" chapter in the *coreConsultant User Guide* discusses how to simulate an individual component like the DW_apb_uart.

2.2.5 Running Leda on Generated Code with coreConsultant

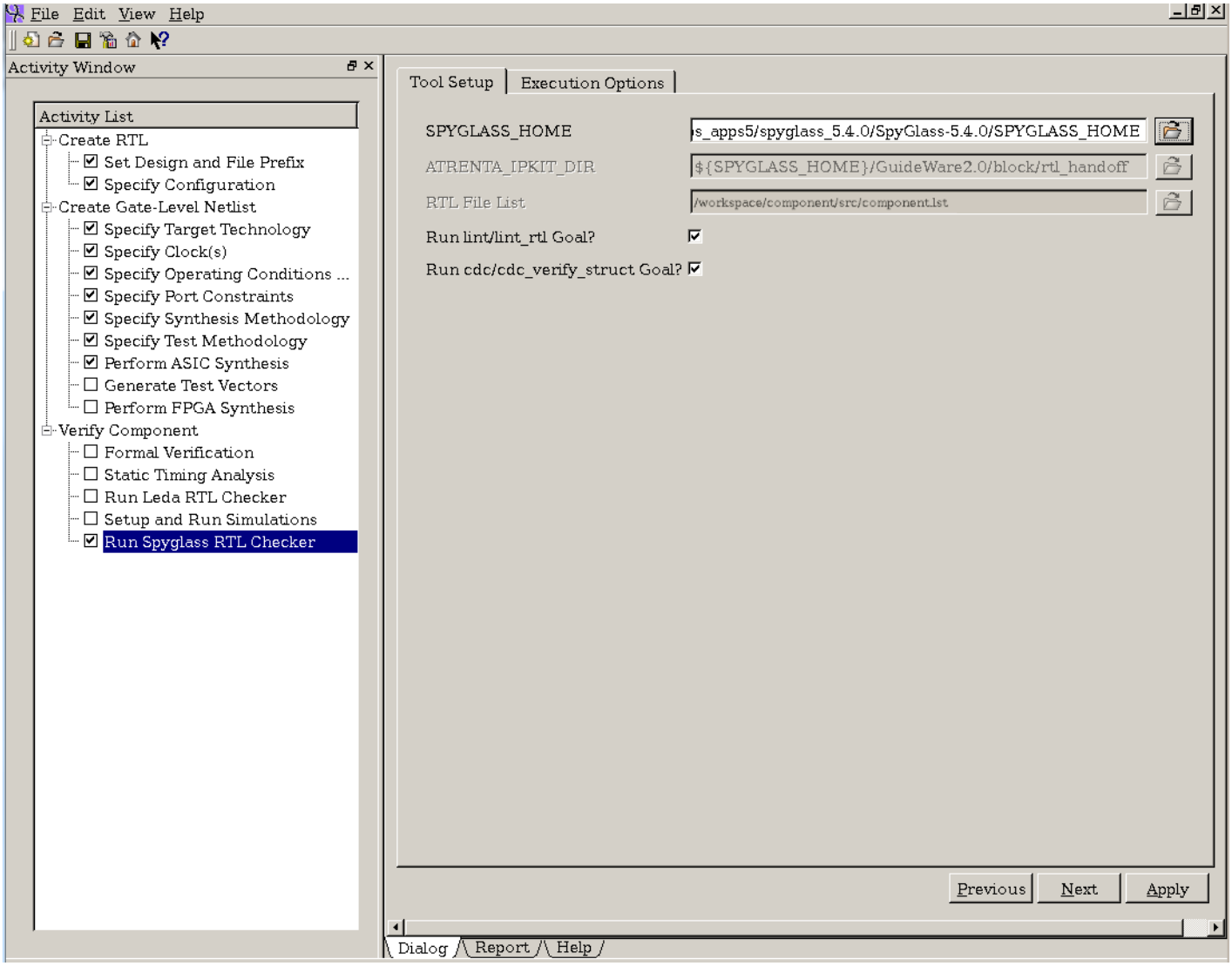
When you select **Verify Component > Run Leda Coding Checker** from the Activity List, the corresponding Activity View appears. In this Activity View you select rules configuration file and define Leda command line switches.

2.2.6 Running SpyGlass® Lint and SpyGlass® CDC

This section discusses the procedure to run SpyGlass Lint and SpyGlass CDC.

Figure 2-2 shows the coreConsultant GUI in which you run Lint and CDC goals.

Figure 2-2 SpyGlass Options in coreConsultant

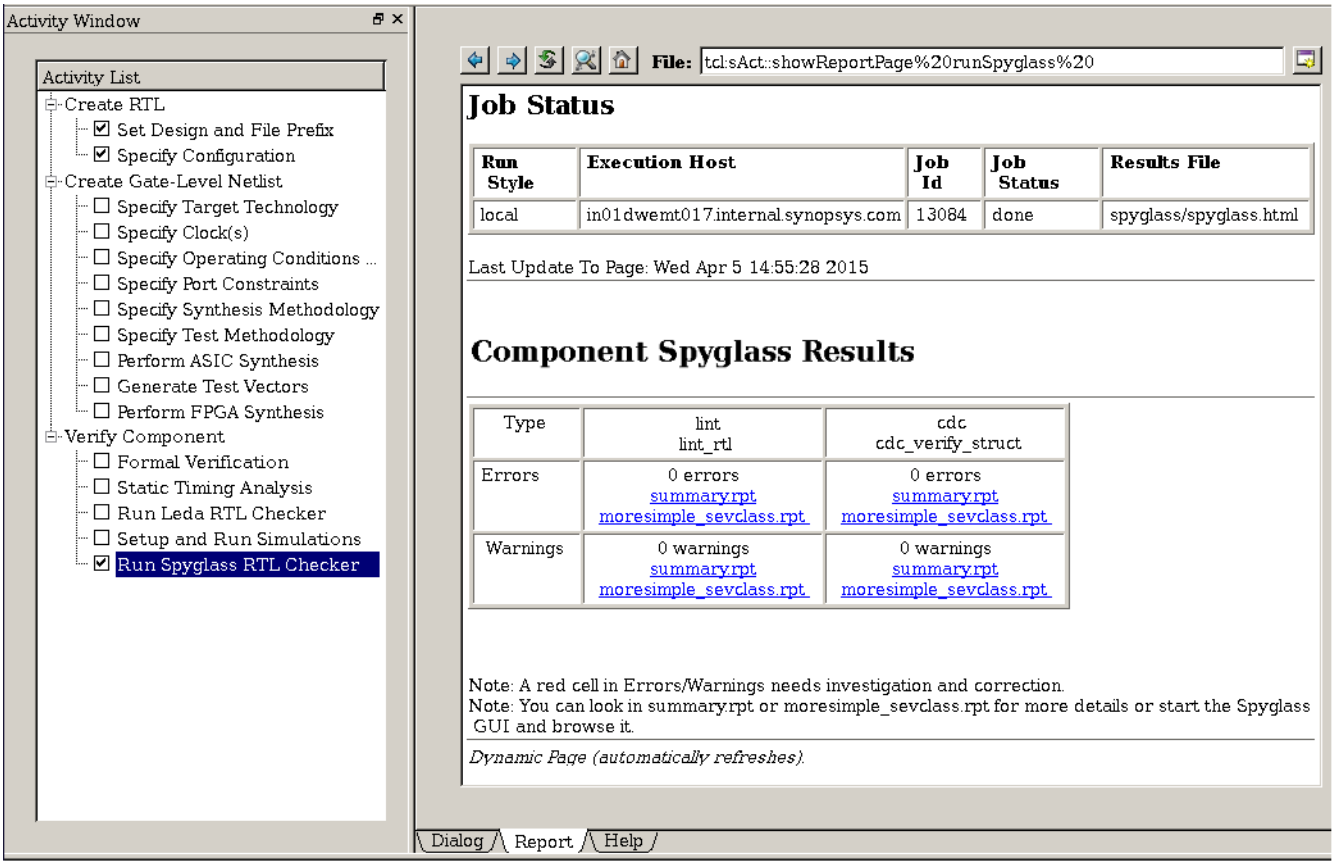


The SpyGlass flow in coreConsultant runs Guideware 2.0 rules for block/rtl_handoff. Within the block/rtl_handoff, only lint/lint_rtl and cdc/cdc_verify_struct goals are run.

In Figure 2-2, select the type of run goals. You can select either Lint run goal or CDC run goal, or both Lint and CDC run goals. By default, both Lint and CDC are selected.

When the Lint and/or CDC is run, the results are available in the Report tab. Errors (if any) are displayed with a red colored cell and warnings (if any) are displayed in yellow colored cell, as shown in Figure 2-3.

Figure 2-3 coreConsultant SpyGlass Report Summary



2.2.6.1 Fixed Settings

The settings are fixed (hardcoded) when you run SpyGlass in coreConsultant.

2.2.6.2 SpyGlass Lint

Table 2-2 lists the SpyGlass Link waiver files that are used by the coreConsultant tool.

Table 2-2 Waiver Files for Sypglass Lint

File Name	Description
<configured_workspace>/spyglass/spyglass_design_specific_waivers.swl	These are DW_apb_uart design-specific rule waivers. This file contains Lint waivers for DW_apb_uart (if applicable). The reason for each of the waivers (if any) are included as comments in the file.
<configured_workspace>/spyglass/spyglass_engineering_council_waivers.swl	This file contains rules that Synopsys waives for its IPs.

2.2.6.3 SpyGlass CDC

To define the SpyGlass CDC constraints, it is important to understand the reset and clock logic used in DW_apb_uart. For information on reset and clock logic, refer "Clock Support" on page 59 for the clocking scheme and "Reset Signals" on page 86 for the reset scheme.

2.2.6.3.1 CDC Files

Table 2-3 summarizes files for SpyGlass CDC used by coreConsultant.

Table 2-3 Waiver Files for Sypglass CDC

File Name	Description
<configured_workspace>/spyglass/manual.sgdc	These are the constraints pertaining to a given mode.
<configured_workspace>/spyglass/ports.sgdc	These are the list of I/O signals and their respective clocks.
<configured_workspace>/spyglass/spyglass_design_specific_waivers.swl	These are DW_apb_uart design-specific rule waivers. This file contains CDC waivers for DW_apb_uart (if applicable). The reason for each of the waivers (if any) are included as comments in the file.
<configured_workspace>/spyglass/spyglass_engineering_council_waivers.swl	These are rules that Synopsys waives for its IPs.

2.2.6.3.2 CDC Path Debug Using the SpyGlass GUI

For debugging the CDC path, it is necessary to run SpyGlass in interactive mode in the configured workspace. To invoke the SpyGlass GUI and to run CDC, complete the following steps:

1. Go to the <*configured_workspace*>/spyglass directory.
2. Issue `./sh.spyglass` to start the spyGlass GUI or issue `./sh.spyglass -batch` to start the SpyGlass in batch mode.
3. In the SpyGlass GUI, the Goal Setup window opens by default.
4. Uncheck the lint_rtl option and click the **Selected Goal (s)** button.
5. After the CDC run is complete, the Analyze Results window displays the results.

Navigate to and select the relevant errors to open a schematic for analysis.

2.3 Overview of the coreAssembler Configuration and Integration Process

Once you have correctly downloaded and installed a release of DesignWare SIP components and then set up your environment, you can begin work on your DesignWare subsystem with coreAssembler.

2.3.1 coreAssembler Usage

Figure 2-4 illustrates some general directories and files in a coreAssembler workspace.

Figure 2-4 coreAssembler Usage Flow

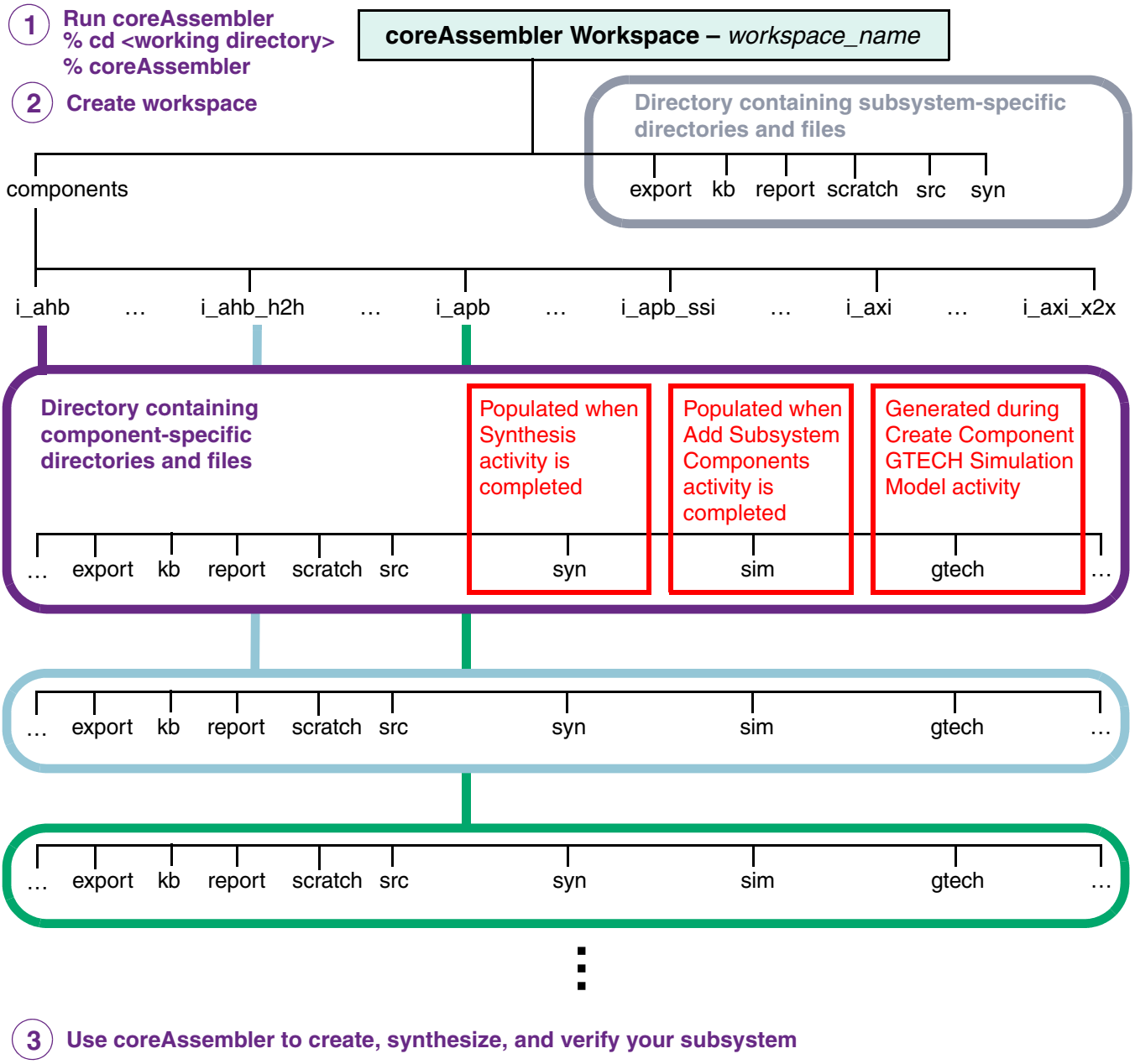


Table 2-4 provides a description of the implementation workspace directory and subdirectories.

Table 2-4 coreAssembler Implementation Workspace Directory Contents

Directory/Subdirectory	Description
components	Contains a directory for each IP component instance connected in the subsystem. Generated and populated with separate component directories upon first adding components; populated and updated throughout activities.
i_*component*/auxiliary	Scripts and text files used by coreAssembler. Generated during Add Subsystem Components activity.
i_*component*/custom	Contains RTL preprocessor scripts. Generated during Configure Components activity.
i_*component*/doc	Contains local copies of component-specific databooks. Generated during Add Subsystem Components activity.
i_*component*/export	Contains files used to integrate results from the completed source configuration and synthesis activities into your design (outside coreAssembler). Generated during Add Subsystem Components activity; populated during Configure Components activity.
i_*component*/gtech	Contains synthesis scripts and output netlists from gtech generation; also used for RTL simulation of encrypted source code. Generated during Create Component GTECH Simulation Model activity.
i_*component*/kb	Contains knowledge base information used by coreAssembler. These are binary files containing the state of the design. Generated during Add Subsystem Components activity; populated and updated throughout activities.
i_*component*/leda	Contains Leda configuration files for the component. Generated during Add Subsystem Components activity; populated during Run Leda Coding Checker (for /i_*component*) activity.
i_*component*/report	Contains all of the reports created by coreAssembler during build, configuration, test and synthesis phases. An index.html file in this directory links to many of these generated reports. Generated during Add Subsystem Components activity; populated and updated throughout activities.
i_*component*/scratch	Contains temp files used during the coreAssembler processes. Generated during Add Subsystem Components activity; populated and updated throughout activities.
i_*component*/sim	Contains test stimulus and output files. Generated during Add Subsystem Components activity; updated during Setup and Run Simulations (for /i_*component*) activity.

Table 2-4 coreAssembler Implementation Workspace Directory Contents (Continued)

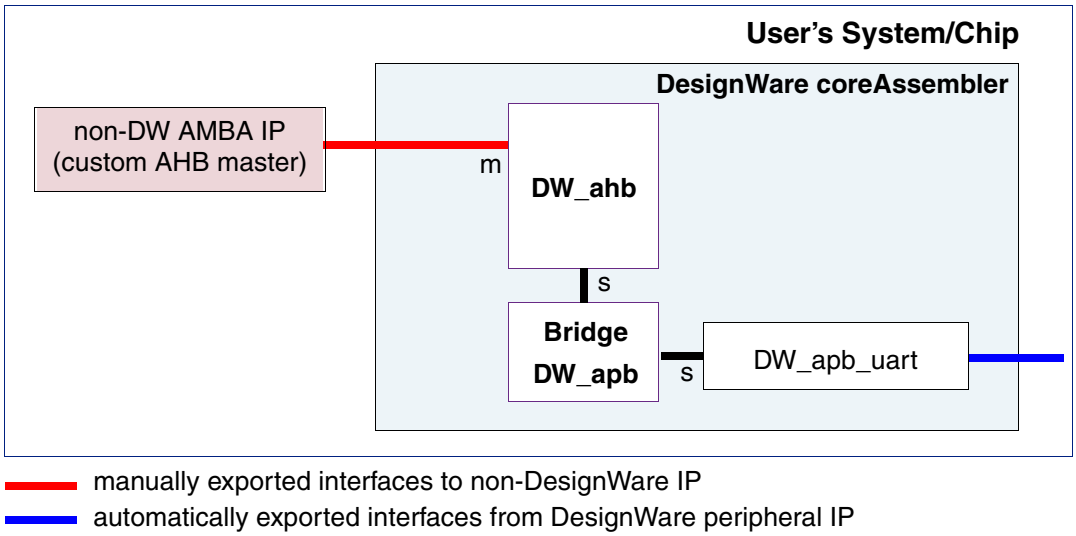
Directory/Subdirectory	Description
i_*component*/spyglass	Contains SpyGlass Lint and CDC configuration files for the component. Generated upon first SpyGlass run; updated during Run Spyglass RTL Checker activity.
i_*component*/src	Includes the top-level RTL file, *design_name*.v. If you have a source license, this will contain plain-text RTL; if you only have a designware license, this will contain encrypted RTL. Generated during Add Subsystem Components activity; populated during Specify Configuration activity.
i_*component*/syn	Contains synthesis files for the component. Generated during Add Subsystem Components activity; updated during Synthesis activity.
i_*component*/tcl	Contains synthesis intent scripts. Generated during Add Subsystem Components activity.
export	Contains subsystem files used to integrate the results from the completed source configuration and synthesis activities into your design (outside coreAssembler). Generated upon first creating workspace; populated starting with Memory Map Specification activity.
kb	Contains subsystem knowledge base information used by coreAssembler. These are binary files containing the state of the design. Generated upon first creating workspace; populated and updated throughout activities.
report	Contains subsystem reports created by coreAssembler during build, configuration, test and synthesis phases. An index.html file in this directory links to many of these generated reports. Generated upon first creating workspace; populated and updated throughout activities.
scratch	Contains subsystem temp files used during the coreAssembler processes. Generated upon first creating workspace; populated and updated throughout activities.
src	Includes the RTL related to the subsystem. If you have a source license, this will contain plain-text RTL; if you only have a designware license, this will contain encrypted RTL. Generated upon first creating workspace; populated starting with Generate Subsystem RTL activity.
syn	Contains synthesis files for the subsystem. Generated upon first creating workspace; updated during Synthesize activity and Formal Verification activity.

For details on some key files created during coreAssembler activities, refer to "Database Files" on page 36.

For information on using coreAssembler, refer to the *coreAssembler User Guide*. For information on getting started with using DesignWare SIP components for AMBA 2 and AMBA 3 AXI components within coreTools, refer to *Using DesignWare Library IP in coreAssembler*.

Figure 2-5 illustrates the DW_apb_uart in a simple subsystem.

Figure 2-5 DW_apb_uart in Simple Subsystem



The subsystem in Figure 2-5 contains the following components that you may want to use as you learn to use coreAssembler:

- DW_apb_uart
- DW_ahb
- DW_apb
- AHB Master

The AHB Master is meant to be exported out of the design and then replaced by a real AHB Master—such as a CPU—later in the design process; at least one exported AHB master is required in a subsystem if you intend to do a basic simulation that tests connections.

2.3.2 Configuring the DW_apb_uart within a Subsystem

The "Parameters" chapter on page 89 describes the DW_apb_uart hardware configuration parameters that you configure using the coreAssembler GUI. Corresponding databooks for the other components in a subsystem contain "Parameters" chapters that describe their respective configuration parameters.

The "Creating the RTL View of a Subsystem" chapter in the *coreAssembler User Guide* discusses how to configure subsystem components and automatically connect them using the coreAssembler GUI.

2.3.3 Creating Gate-Level Netlists within coreAssembler

The "Creating the Gate-Level Netlist for a Subsystem" chapter in the *coreAssembler User Guide* discusses how to create a translation of the RTL view into a technology-specific netlist for a subsystem.

2.3.4 Verifying the DW_apb_uart within coreAssembler

The "Verification" chapter on page 187 provides an overview of the testbench available for DW_apb_uart verification using the coreAssembler GUI.

The "Verifying Subsystems and Components" chapter in the *coreAssembler User Guide* discusses how to simulate a subsystem.

2.3.5 Running Leda on Generated Code with coreAssembler

When you select **Verify Component > Run Leda Coding Checker for /i_*component*)** from the Activity List, the corresponding Activity View appears. In this Activity View you select rules configuration file and define Leda command line switches.

2.3.6 Running Spyglass on Generated Code with coreAssembler

When you select **Verify Component > Run Spyglass RTL Checker for /*i_component*** from the Activity List, the corresponding Activity View appears. In this Activity View, you can select to run Spyglass Lint and Spyglass CDC.

2.4 Database Files

The following subsections describe some key files created in coreConsultant and coreAssembler activities.

2.4.1 Design/HDL Files

The following sections describe the design and HDL files that are produced by coreConsultant and coreAssembler when configuring and verifying a DesignWare Synthesizable Component. The following files are created in different directories by coreConsultant and coreAssembler:

- coreConsultant – *workspace*/ directory
- coreAssembler – *workspace*/components/i_*component*/ directory

2.4.1.1 RTL-Level Files

The following table describes the RTL files that are generated by the Create RTL activity. They are encrypted except where otherwise noted. Any Synopsys synthesis tool or simulator can read encrypted RTL files.

Table 2-5 RTL-Level Files

Files	Encrypted?	Purpose
./src/*component*_cc_constants.v	No	Includes definitions and values of all configuration parameters that you have specified for the component.

./src/*component*.v	No	Top-level HDL file. Include the DesignWare libraries by using the following options in your simulator invocation:
		`+libext+.v+.V`
		`-y ${SYNOPSYS}/packages/gtech/src_ver`
		`-y ${SYNOPSYS}/dw/sim_ver`

Table 2-5 RTL-Level Files (Continued)

Files	Encrypted?	Purpose
./src/*component_submodule*.v	Yes	Sub-modules of component
./src/*component*_constants.v	No	Includes the constants used internally in the design.
./src/*component*_undef.v		Includes an undef for each of the definitions found in the *component*_cc_constants.v file; compiled in after the last file listed in ./src/*components*.lst when compiling multiple instances of the same IP.
./src/*component*.lst	No	Lists the order in which the RTL files should be read into tools, such as simulators or dc_shell. For example, use the following option to read the design into VCS: `vcs +v2k -f component.lst`

2.4.1.2 Simulation Model Files

The following table includes files generated for the component during the Generate GTECH Simulation activity. These files are needed when you are using a non-Synopsys simulator (when you can not use the encrypted RTL).

Table 2-6 Simulation Model Files

Files	Encrypted?	Purpose

./gtech/final/db/*component*.v	No	Simulation model of the component for use with non-Synopsys simulators. A technology-independent, gate-level netlist; VHDL and Verilog versions are generated. Include the DesignWare libraries by using the following options in your simulator invocation:
		`+libext+.v+.V`
		`-y ${SYNOPSYS}/packages/gtech/src_ver`
		`-y ${SYNOPSYS}/dw/sim_ver`

2.4.2 Synthesis Files

The following table includes files generated after the Create Gate-Level Netlist activity is performed on a component.

Table 2-7 Synthesis Files

Files	Encrypted?	Purpose
./syn/auxScripts	No	Auxiliary files for synthesis.
./syn/final/db/*component*.db	Binary format	Synopsys .db files (gate level) that can be read into dc_shell for further synthesis, if desired.
./syn/final/db/*component*.v	No	Gate-level netlist that is mapped to technology libraries that you specify.
./syn/constrain/script/*.*	No	Constraint files for the components.
./syn/final/report/*.*	No	Synthesis result files.

2.4.3 Verification Reference Files

Files described in the following table include information pertaining to the component's operation so that you can verify installation and configuration of the component has been successful. These files are not for re-use during system-level verification.

Table 2-8 Verification Reference Files

Files	Encrypted?	Purpose
./sim/runtest	No	Perl script that runs the Setup and Run Simulations activity from the command line.
./sim/runtest.log	No	The overall result of simulation, including pass/fail results.
./sim/test_*testname*/test.result	No	Pass/fail of individual test.
./sim/test_*testname*/test.log	No	Log file for individual test.

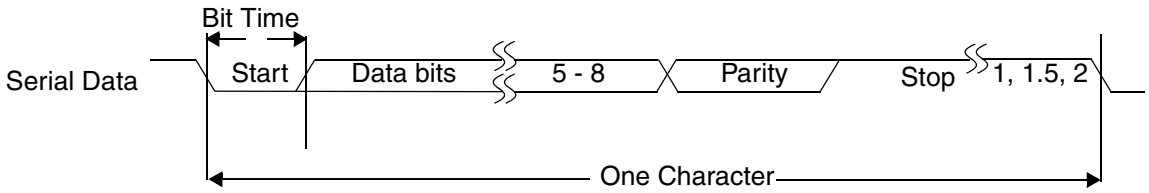
3 Functional Description

This chapter describes the functional operation of the DW_apb_uart.

3.1 UART (RS232) Serial Protocol

Because the serial communication between the DW_apb_uart and a selected device is asynchronous, additional bits (start and stop) are added to the serial data to indicate the beginning and end. Utilizing these bits allows two devices to be synchronized. This structure of serial data—accompanied by start and stop bits—is referred to as a character, as shown in Figure 3-1.

Figure 3-1 Serial Data Format



An additional parity bit can be added to the serial character. This bit appears after the last data bit and before the stop bit(s) in the character structure in order to provide the DW_apb_uart with the ability to perform simple error checking on the received data.

The DW_apb_uart Line Control Register ("LCR" on page 131) is used to control the serial character characteristics. The individual bits of the data word are sent after the start bit, starting with the least-significant bit (LSB). These are followed by the optional parity bit, followed by the stop bit(s), which can be 1, 1.5, or 2.

Note

The STOP bit duration implemented by DW_apb_uart can appear longer due to:

- Idle time inserted between characters for some configurations
- Baud clock divisor values in the transmit direction

For details on idle time between transmitted transfers, refer to "Back-to-Back Character Stream Transmission" on page 62.

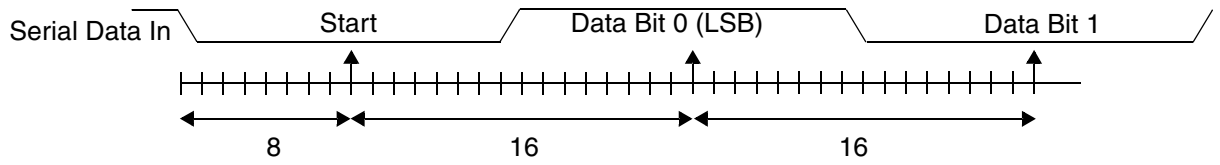
All the bits in the transmission are transmitted for exactly the same time duration; the exception to this is the half-stop bit when 1.5 stop bits are used. This duration is referred to as a Bit Period or Bit Time; one Bit Time equals sixteen baud clocks.

To ensure stability on the line, the receiver samples the serial input data at approximately the midpoint of the Bit Time once the start bit has been detected. Because the exact number of baud clocks is known for which each bit was transmitted, calculating the midpoint for sampling is not difficult; that is, every sixteen baud clocks after the midpoint sample of the start bit.

Together with serial input debouncing, this sampling helps to avoid the detection of false start bits. Short glitches are filtered out by debouncing, and no transition is detected on the line. If a glitch is wide enough to avoid filtering by debouncing, a falling edge is detected. However, a start bit is detected only if the line is again sampled low after half a bit time has elapsed.

Figure 3-2 shows the sampling points of the first two bits in a serial character.

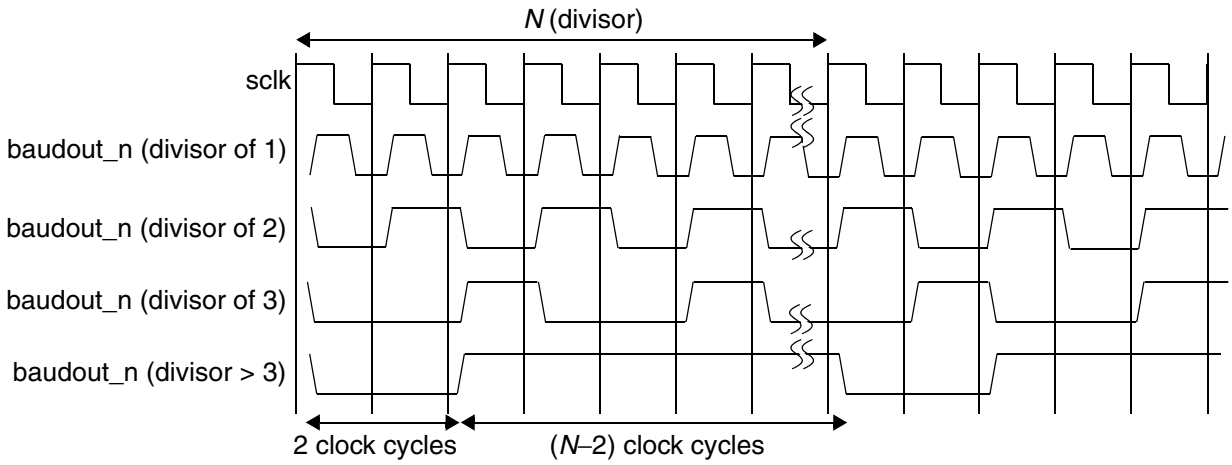
Figure 3-2 Receiver Serial Data Sample Points



As part of the 16550 standard, an optional baud clock reference output signal (baudout_n) provides timing information to receiving devices that require it. The baud rate of the DW_apb_uart is controlled by the serial clock—sclk or pclk in a single clock implementation—and the Divisor Latch Register (DLH and DLL).

Figure 3-3 shows the timing diagram for the baudout_n output for different divisor values.

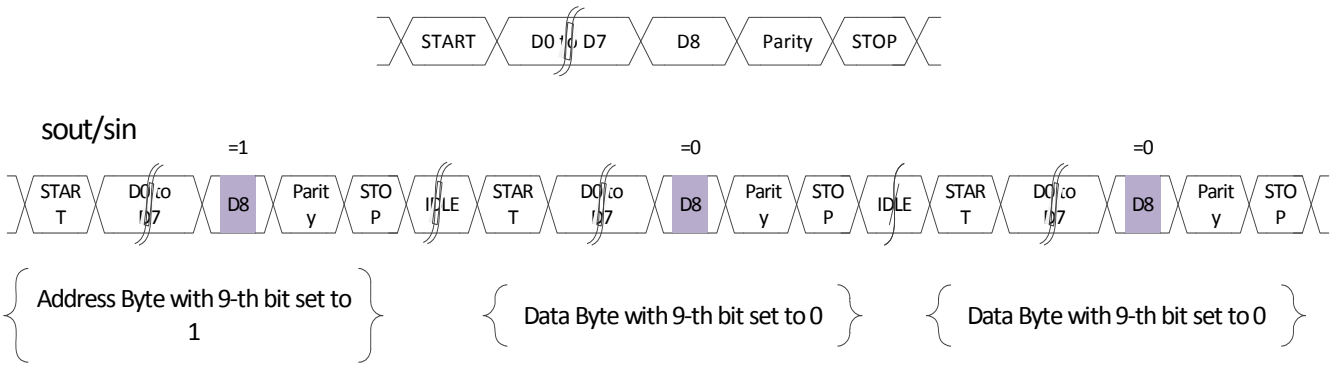
Figure 3-3 Baud Clock Reference Timing Diagram



3.2 9-bit Data Transfer

The DW_apb_uart can be configured to have 9-bit data transfer in both transmit and receive mode. The 9th bit in the character appears after the 8th bit and before the parity bit in the character. Figure 3-4 shows the serial transmission for a character in which D8 represents the 9th bit and also shows general serial transmission in the 9-bit mode.

Figure 3-4 9-Bit Character



By enabling 9-bit data transfer mode, DW_apb_uart can be used in multi-drop systems where one master is connected to multiple slaves in a system. The master communicates with one of the slaves. When the master wants to transfer a block of data to a slave, it first sends an address byte to identify the target slave. The address byte is differentiated from the data byte by setting the 9th bit of the data byte to 1. If the 9th bit is set to 0, then the character represents an address byte. All the slave systems compare the address byte with their own address and only the target slave (in which the address has matched) is enabled to receive data from the master. The master then starts transmitting data bytes to the target slave. The non-addressed slave systems ignore the incoming data until a new address byte is received.

In Figure 3-4, note that one address is followed by 2 data bytes. The address byte goes out with the 9^{th} bit (D8) set to 1 and the data bytes go out with 9^{th} bit (D8) set to 0. The parity bit is an optional field.

Configuration of the DW_apb_uart for 9-bit data transfer does the following:

- LCR_EXT[0] bit is used to enable or disable the 9-bit data transfer.
- LCR_EXT[1] bit is used to choose between hardware and software based address match in the case of receive.
- LCR_EXT[2] bit is used to enable to send the address in the case of transmit.
- LCR_EXT[3] bit is used to choose between hardware and software based address transmission.
- TAR and RAR registers are used to transmit address and to match the received address, respectively.
- THR, RBR, STHR and SRBR registers are of 9-bit which is used to do the data transfers in 9-bit mode.
- LSR[8] bit is used to indicate the address received interrupt.

> **Note** The 9-bit data mode is supported only when the DWC-APB-Advanced-Source source license exists.

3.2.1 Transmit Mode

DW_apb_uart supports two types of transmit modes:

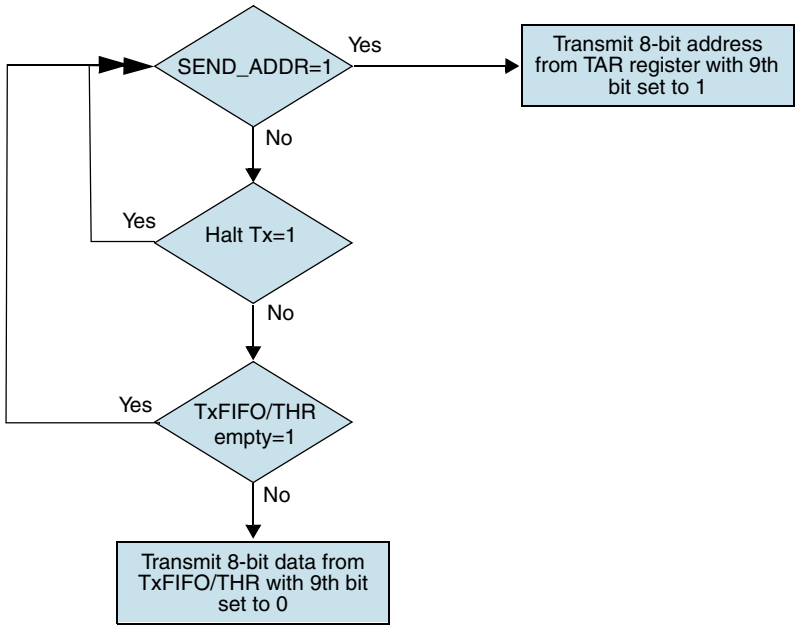
- Transmit Mode 0 (when (LCR_EXT[3]) is set to 0)
- Transmit Mode 1 (when (LCR_EXT[3]) is set to 1)

3.2.1.1 Transmit Mode 0

In transmit mode 0, the address is programmed in the Transmit Address Register (TAR) register and data is written into the Transmit Holding Register (THR) or the Shadow Transmit Holding Register (STHR). The 9^{th} bit of the THR and STHR register is not applicable in this mode.

Figure 3-5 illustrates the transmission of address and data based on SEND_ADDR (LCR_EXT[2]), Halt Tx, and TxFIFO/THR empty conditions.

Figure 3-5 Auto Address Transmit Flow Chart



The address of the target slave to which the data is to be transmitted is programmed in the TAR register. You must enable the SEND_ADDR (LCR_EXT[2]) bit to transmit the target slave address present in the TAR register on the serial UART line with 9^{th} data bit set to 1 to indicate that the address is being sent to the slave. The DW_apb_uart clears the SEND_ADDR bit after the address character starts transmitting on the UART line.

The data required to transmit to the target slave is programmed through Transmit Holding Register (THR). The data is transmitted on the UART line with 9th data bit set to 0 to indicate data is being sent to the slave.

If the application is required to fill the data bytes in the TxFIFO before sending the address on the UART line (before setting LCR_EXT[2]=1), then it is recommended to set the “Halt Tx” to 1 such that DW_apb_uart does not start sending out the data in the TxFIFO as data byte. Once the TxFIFO is filled, then program SEND_ADDR (LCR_EXT[2]) to 1 and then set "Halt Tx" to 0.

3.2.1.2 Transmit Mode 1

In transmit mode 1, THR and STHR registers are of 9-bit wide and both address and data are programmed through the THR and STHR registers. The DW_apb_uart does not differentiate between address and data, and both are taken from the TxFIFO. The SEND_ADDR (LCR_EXT[2]) bit and Transmit address register (TAR) are not applicable in this mode. The software must pack the 9th bit with 1/0 depending on whether address/data has to be sent.

3.2.2 Receive Mode

The DW_apb_uart supports two receive modes:

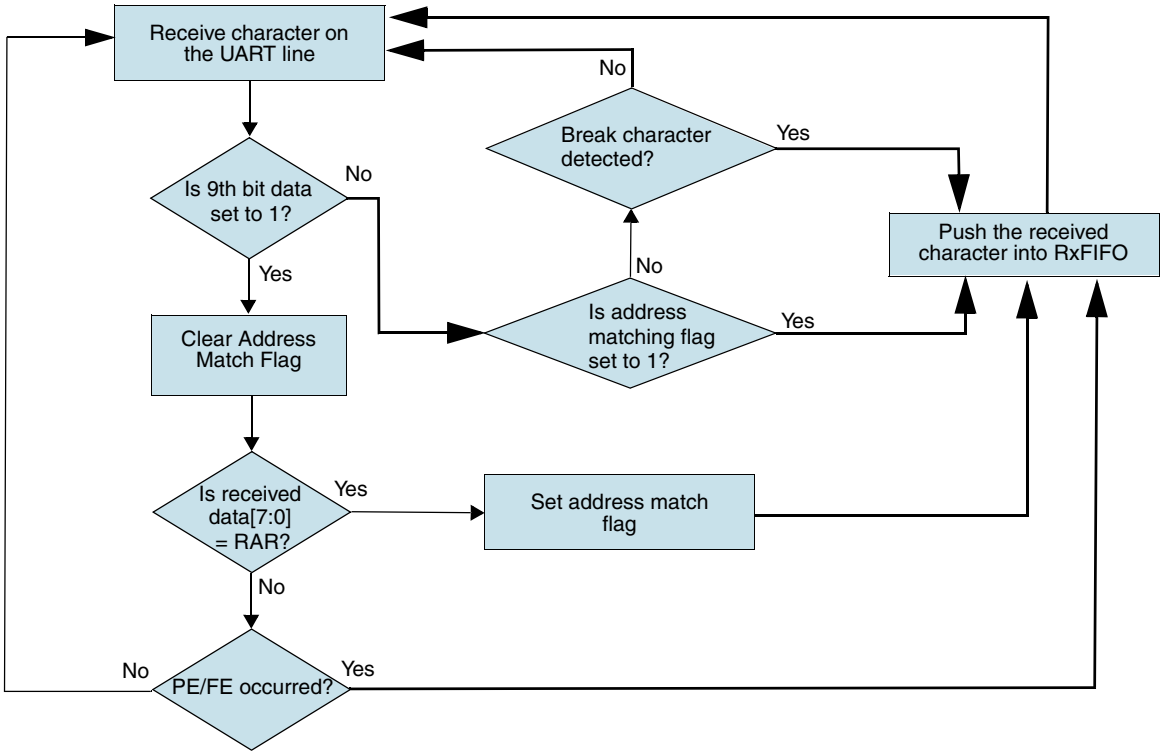
- Hardware Address Match Receive Mode (when ADDR_MATCH (LCR_EXT[1]) is set to 1)
- Software Address Match Receive Mode (when ADDR_MATCH (LCR_EXT[1]) is set to 0)

3.2.2.1 Hardware Address Match Receive Mode

In the hardware address match receive mode, the DW_apb_uart matches the received character with the address programmed in the Receive Address register (RAR), if the 9th bit of the received character is set to 1. If the received address is matched with the programmed address in RAR register, then subsequent data bytes (with 9th bit set to 0) are pushed into the RxFIFO. If the address matching fails, then DW_apb_uart controller discards further data characters until a matching address is received.

Figure 3-6 illustrates the flow chart for the reception of data bytes based on the address matching feature.

Figure 3-6 Hardware Address Match Receive Mode



DW_apb_uart receives the character irrespective of whether the 9th bit data is set to 1. If 9th bit of the received character is set to 1, then it clears internal address match flag and then compares the received 8-bit character information with the address programmed in the RAR register.

If the received address character matches with the address programmed in the RAR register, then the address match flag is set to 1 and the received character is pushed to the RxFIFO in FIFO-mode or to RBR register in non-FIFO mode and the ADDR_RCVD bit in LSR register is set to indicate that the address has been received.

In case of parity or if a framing error is found in the received address character and if the address is not matched with the RAR register, then the received address character is still pushed to RxFIFO or RBR register with ADDR_RCVD and PE/FE error bit set to 1.

The subsequent data bytes (9th bit of received character is set to 0) are pushed to the Rx_FIFO in FIFO mode or to the RBR register in non-FIFO mode until the new address character is received.

If any break character is received, DW_apb_uart treats it as a special character and pushes to the RxFIFO or RBR register based on the FIFO_MODE irrespective of address match flag.

Note The break character can be used to alert the complete system in case all slaves are in sleep mode (entered in to the low power mode). Therefore, the break character is treated as special character.

3.2.2.2 Software Address Match Receive Mode

In this mode of operation, the DW_apb_uart does not perform the address matching for the received address character (9^{th} bit data set to 1) with the RAR register. The DW_apb_uart always receives the 9-bit data and pushes in to RxFIFO in FIFO mode or to the RBR register in non-FIFO mode. The user must compare the address whenever address byte is received and indicated through ADDR_RCVD bit in the Line Status register. The user can flush/reset the RxFIFO in case of address not matched through 'RCVR FIFO Reset' bit in FIFO control register (FCR).

3.3 RS485 Serial Protocol

The RS485 standard supports serial communication over a twisted pair configuration, such as RS232. The difference between the RS232 and RS485 standards is its use of a balanced line for transmission. This usage is also known as the differential format that sends the same signal on two separate lines with phase delay and then compares the signals at the end, subtracts any noise, and adds them to regain signal strength. This process allows the RS485 standard to be viable over significantly longer distances than its short range RS232 counterpart.

DW_apb_uart supports the RS485 serial protocol that enables transfer of serial data using the RS485 interface. The driver enable (DE) and receiver enable (RE) signals are generated for enabling the RS485 interface support. The de and re signals are hardware generated and the assertion/de-assertion times for these signals are programmable. The active level of these signals are configurable.

Configuration of the DW_apb_uart for RS485 interface does the following:

1. Bit 0 of the Transceiver Control Register (TCR) enables or disables the RS485 mode.
2. Bit 1 and bit 2 of TCR are used to select the polarity of RE and DE signals.
3. Bit [4:3] of the TCR selects the type of transfer in RS485 mode.
4. Driver output enable (DE_EN) and Receiver output enable (RE_EN) registers are used for software control of DE and RE signals.
5. Driver output Enable Timing (DET) register is used to program the assertion and deassertion timings of DE signal.
6. TurnAround Timing (TAT) register is used to program the turnaround time from DE to RE and RE to DE.

RS485 interface mode is supported only when the source license DWC-APB-Advanced-Source exists.

3.3.1 DE Assertion and De-assertion Timing

The assertion and deassertion timings of the DE signal are controlled through the DET register:

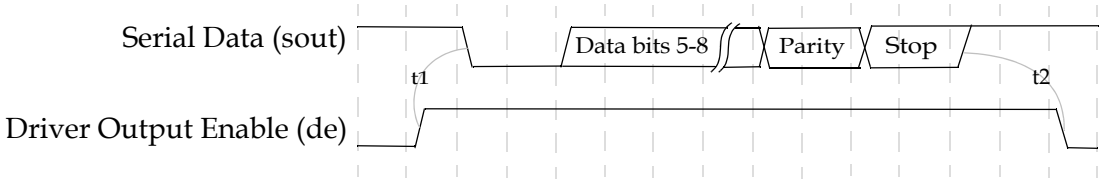
- DE assertion time (DET[7:0]): The assertion time is the time between the activation of the DE signal and the beginning of the START bit. The value represented is in terms of serial clock cycles.

- DE de-assertion time (DET[15:8]): The de-assertion time is the time between the end of the last stop bit, in a transmitted character, and the de-activation of the DE signal. The value represented is in terms of serial clock cycles.

Hardware ensures that these values are met for DE assertion and DE deassertion before/after active data transmission.

In Figure 3-7, t1 represents DE assertion time and t2 represents DE de-assertion time. Note that for simplicity only one data is illustrated in Figure 3-7; however, DE will not get de-asserted if there are more data characters in transmit FIFO. DE gets de-asserted only after all the data characters are transmitted.

Figure 3-7 DE Assertion and De-Assertion



3.3.2 RS485 Modes

DW_apb_uart consists of the following RS485 modes based on the XFER_MODE field in the Transceiver Control Register (TCR) register:

- Full Duplex Mode – In this mode, XFER_MODE of TCR is set to 0.
- Software-Controlled Half Duplex Mode – In this mode, XFER_MODE of TCR is set to 1.
- Hardware-Controlled Half Duplex Mode – In this mode, XFER_MODE of TCR is set to 2

3.3.2.1 Full Duplex Mode

The full duplex mode supports both transmit and receive transfers simultaneously.

In Full Duplex mode, the de signal:

- Goes active if both these conditions are satisfied:
  - When the DE Enable (DE_EN[0]) field of Driver Output Enable Register is set to 1.
  - Transmitter Holding Register is not empty in non-FIFO mode or transmitter FIFO is not empty in FIFO mode.
- Goes inactive if both these conditions are satisfied
  - When the current ongoing transmitting serial transfer is completed.
  - Either DE Enable (DE_EN[0]) of Driver Output Enable Register is set to 0, transmitter FIFO is empty in FIFO mode or Transmitter Holding Register is empty in non-FIFO mode.

In Full Duplex mode, the re signal:

- Goes active when RE Enable (RE_EN[0]) of Receiver Output Enable Register is set to 1.

- Goes inactive when RE Enable (RE_EN[0]) of Receiver Output Enable Register is set to 0.

The user can choose when to transmit or when to receive. Both 're' and 'de' can be simultaneously asserted or de-asserted at any time. DW_apb_uart does not impose any turnaround time between transmit and receive ('de to re') or receive to transmit ('re to de') in this mode. This mode can directly be used in full duplex operation where separate differential pair of wires is present for transmit and receive.

3.3.2.2 Software-Controlled Half Duplex Mode

The software-controlled half duplex mode supports either transmit or receive transfers at a time but not both simultaneously. The switching between transmit to receive or receive to transmit is through programming the Driver output enable (DE_EN) and Receiver output enable (RE_EN) registers.

In software-controlled Half Duplex mode, the de signal:

- Goes active if the following conditions are satisfied:
  - The DE Enable (DE_EN[0]) field of the Driver Output Enable Register is set to 1.
  - Transmitter Holding Register is not empty in non-FIFO mode or transmitter FIFO is not empty in FIFO mode.
  - If any receive transfer is ongoing, then the signal waits until receive has finished, and after the turnaround time counter ('re to de') has elapsed.
- Goes inactive if the following conditions are satisfied:
  - The current ongoing transmitting serial transfer is completed.
  - The DE Enable (DE_EN[0]) field of Driver Output Enable Register is set to 0.
  - Either transmitter FIFO is empty in FIFO mode or Transmitter Holding Register is empty in non-FIFO mode.

In software-controlled half duplex mode, the re signal:

- Goes active if the following conditions are satisfied:
  - When RE Enable (RE_EN[0]) field of Receiver Output Enable Register is set to 1.
  - If any transmit transfer is ongoing, then the signal waits until transmit has finished and after the turnaround time counter ('de to re') has elapsed.
- Goes in-active under the following conditions:
  - The current ongoing receive serial transfer is completed.
  - When RE Enable (RE_EN[0]) of Receiver Output Enable Register is set to 0.

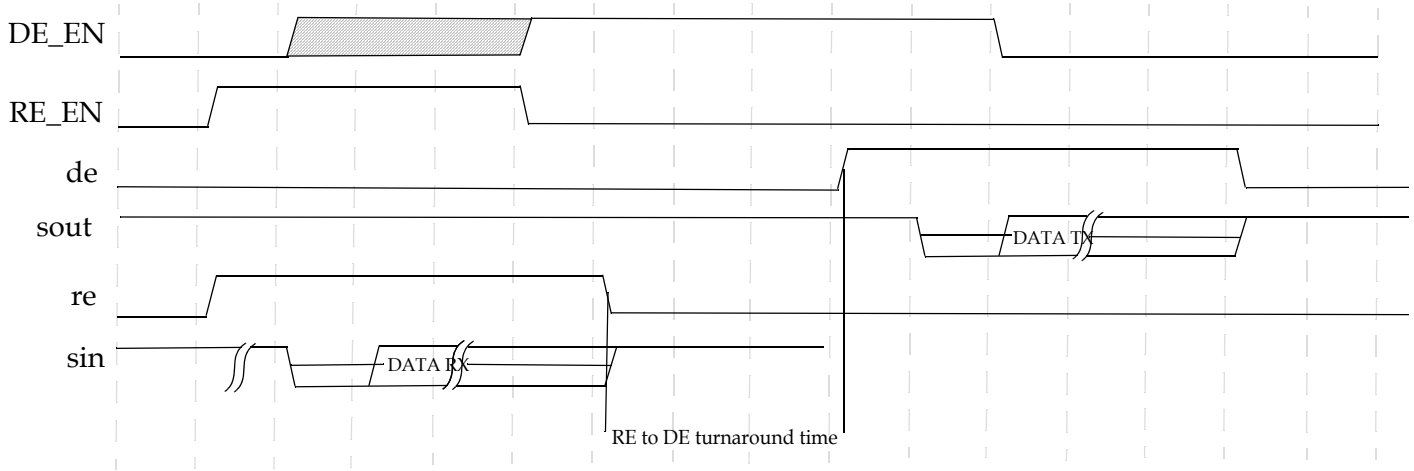
The user must enable either DE or RE but not both at any point of time. As 're' and 'de' signals are mutually exclusive, the user must ensure that both of them are not programmed to be active at any point of time.

In this mode, the hardware ensures that a proper turnaround time is maintained while switching from 're' to 'de' or from 'de' to 're' (value of turnaround is obtained from the TAT register, in terms of serial clock cycles) as shown in Figure 3-8 and Figure 3-9.

3.3.2.2.1 RE to DE Turnaround Time

DW_apb_uart inserts the wait state (as programmed in TAT[31:16] times serial clock) before switching to transmit mode from receive mode as shown in the Figure 3-8 (applicable only when TCR[4:3] =1 or 2 (XFER_MODE).)

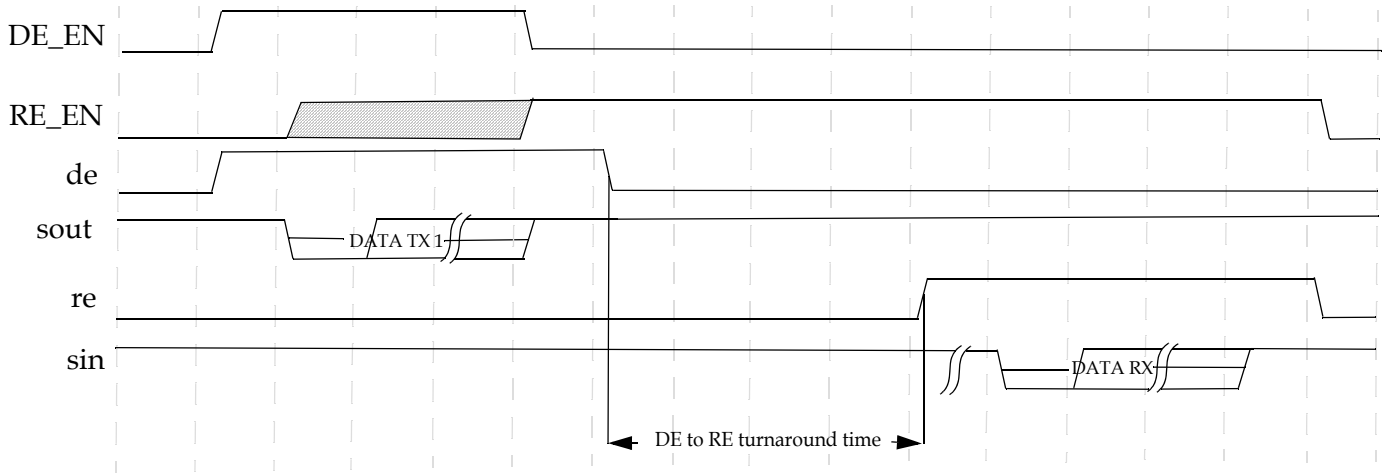
Figure 3-8 RE to DE Turnaround Time



3.3.2.3 DE to RE Turnaround Time

DW_apb_uart inserts the wait state (as programmed in TAT[15:0] times serial clock) before switching to receive mode from transmit mode as shown in the Figure 3-9 (applicable only when TCR[4:3] =1 or 2 (XFER_MODE).)

Figure 3-9 DE to RE Turnaround Time



3.3.2.4 Hardware-Controlled Half Duplex Mode

The hardware-controlled half duplex mode supports either transmit or receive transfers at a time but not both simultaneously. If both 'DE Enable' and 'RE Enable' bits of Driver output enable (DE_EN) and Receiver output enable (RE_EN) registers are enabled, the switching between transmit to receive or receive to transmit is automatically done by the hardware based on the empty condition of Tx-FIFO.

In hardware-controlled half duplex mode, the de signal:

- Goes active if the following conditions are satisfied:
  - The DE Enable (DE_EN[0]) field of Driver Output Enable Register is set to 1.
  - Transmitter Holding Register is not empty in non-FIFO mode or transmitter FIFO is not empty in FIFO mode.
  - If any receive transfer is ongoing, then the signal waits until receive is finished and after the turnaround time counter ('re to de') has elapsed.
- Goes inactive if the following conditions are satisfied
  - The current ongoing transmitting serial transfer is completed.
  - Either transmitter FIFO is empty in FIFO mode or Transmitter Holding Register is empty in non-FIFO mode or the DE Enable (DE_EN[0]) of Driver output Enable Register is set to 0.

In hardware-controlled half duplex mode, the re signal:

- Goes active if the following conditions are satisfied:
  - When RE Enable (RE_EN[0]) field of Receiver Output Enable Register is set to 1.
  - Either transmitter FIFO is empty in FIFO mode or Transmitter Holding Register is empty in non-FIFO mode.
  - If any transmit transfer is ongoing, then the signal waits until transmit is finished and after the turnaround time counter ('de to re') has elapsed.
- Goes inactive under the following conditions:
  - The current ongoing receive serial transfer has completed.
  - Either transmitter FIFO is non-empty in FIFO mode or Transmitter Holding Register is non empty in non-FIFO mode or the RE Enable (RE_EN[0]) of Receiver output Enable Register is set to 0.

In this mode, the hardware ensures that a proper turnaround time is maintained while switching from 're' to 'de' or from 'de' to 're' (value of turnaround is obtained from the TAT register, in terms of serial clock cycles) as shown in Figure 3-8 and Figure 3-9.

3.3.3 Sample Scenarios

Consider a scenario in which the DW_apb_uart controller is receiving 3 characters and another UART device is sending those characters. While the 1st character is being received by the DW_apb_uart controller, if the software writes into the TX FIFO of the DW_apb_uart controller, then at the end of the first character DW_apb_uart controller will switch the mode from receive to transmit. DW_apb_uart will de-assert 're' and assert 'de' signal. This will make the DW_apb_uart controller not to receive the subsequent characters.

Hence, in hardware switching half duplex mode, the user has to ensure that complete receive data has been received before writing in to the Tx-FIFO to avoid missing of receive characters.

Following sections explain the behavior of DW_apb_uart in XFER_MODE=2 for different scenarios.

3.3.3.1 Normal Scenario of Transmission

Figure 3-10 is a sample scenario for normal transmission.

Figure 3-10 Scenario When XFER_MODE=2

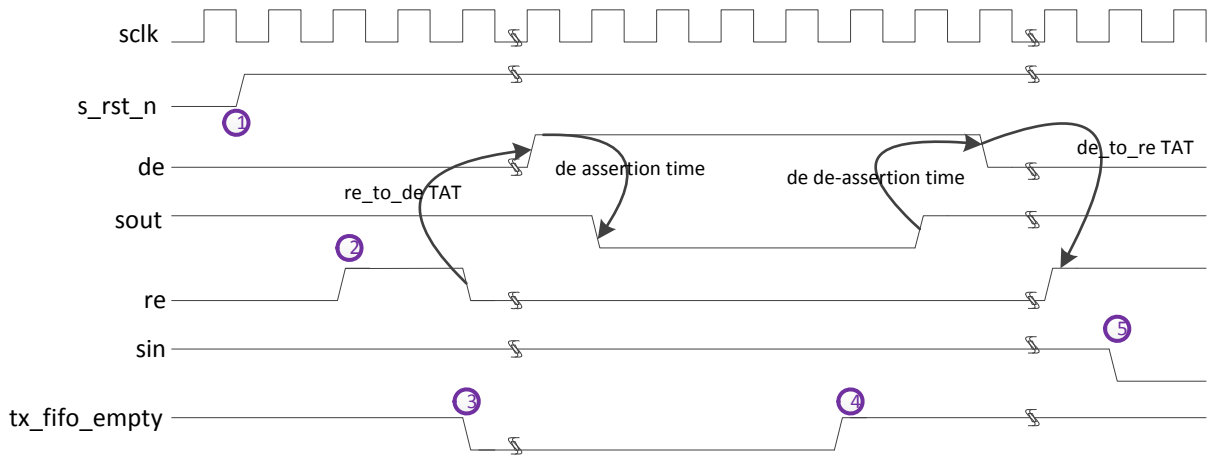


Figure 3-10 shows the following activities at various points in this scenario:

1. At this point, reset is removed, and de and re signals are driven to their configured reset values (UART_DE_POL/UART_RE_POL).
2. At this point, the software programs DE_EN and RE_EN register to 1. At this point in time, tx_fifo_empty * is 1 indicating that there is no data in TX FIFO. Hence, the 'de' signal remains de-asserted and 're' gets asserted.

* tx_fifo_empty is internal signal of DW_apb_uart

3. At this point, the software fills the TX FIFO and there is no ongoing Receive transfer. Therefore, the ‘re' signal goes low. However, the DW_apb_uart controller waits until 're_to_de' TAT value before asserting 'de' signal. After the 'de' gets asserted, the transmission of character starts considering the 'de-asserting timing'.
4. At this point, TX FIFO becomes empty. After transmitting the current character, DW_apb_uart will de-assert the 'de' signal (after de de-assertion time). DW_apb_uart controller waits until 'de_to_re' TAT values before asserting 're' signal back.
5. At this point, DW_apb_uart controller starts receiving the character.

3.3.3.2 Scenario When Receive is in Progress While TX FIFO is Being Filled

In this scenario, TX FIFO is filled when a character is being received. In this case, DW_apb_uart is expected to wait till the current character is finished before changing the role and start transmitting.

Figure 3-11 Receive in Progress, When TX FIFO is Filled

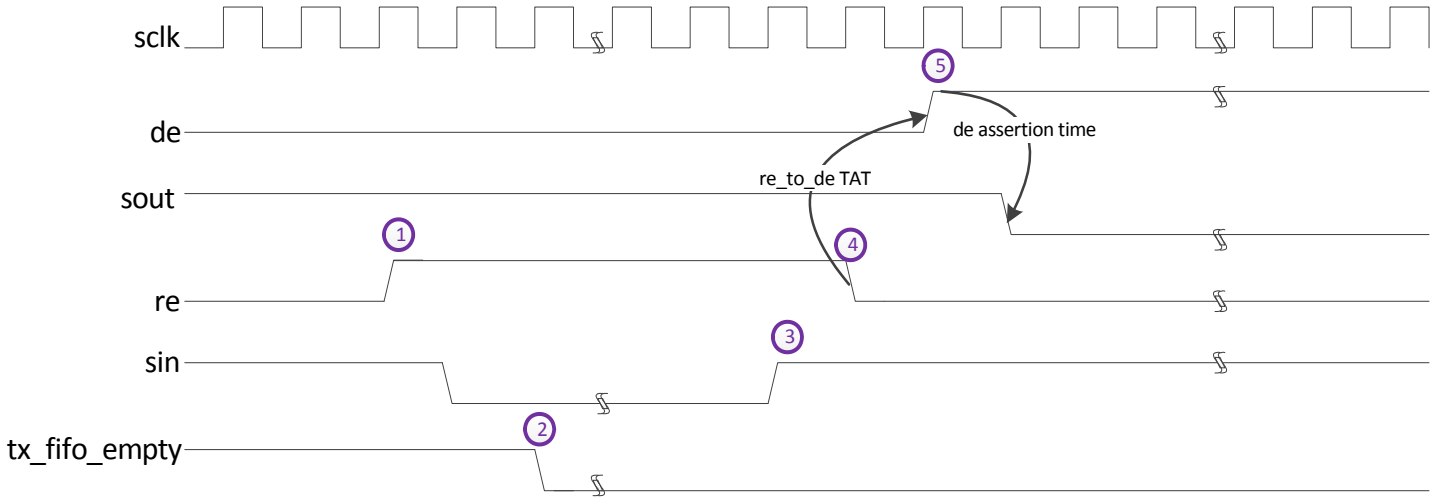


Figure 3-11 shows the following activities at various points in this scenario:

1. The software programs DE_EN and RE_EN to 1, thereby asserting the 're' signal. After this, the DW_apb_uart controller starts receiving the character.
2. The software programs TX FIFO thereby making 'tx_fifo_empty' to go low. However, the DW_apb_uart controller waits until the current character is received before asserting the 'de' signal.
3. The incoming character is fully received.
4. The 're' signal gets de-asserted, after the STOP bit is fully received.
5. After the 're_to_de' TAT, the 'de' signal gets asserted and the DW_apb_uart controller starts transmitting after DET timings.

3.3.3.3 TX FIFO Filled Before Enabling DE_EN and RE_EN Registers

In this case, TX FIFO is filled prior to enabling DE_EN or RE_EN. The DW_apb_uart controller enables the 'de' instead of 're' in this case because TX FIFO already has the data to transmit.

Figure 3-12 TX FIFO is Filled Before Enabling DE/RE

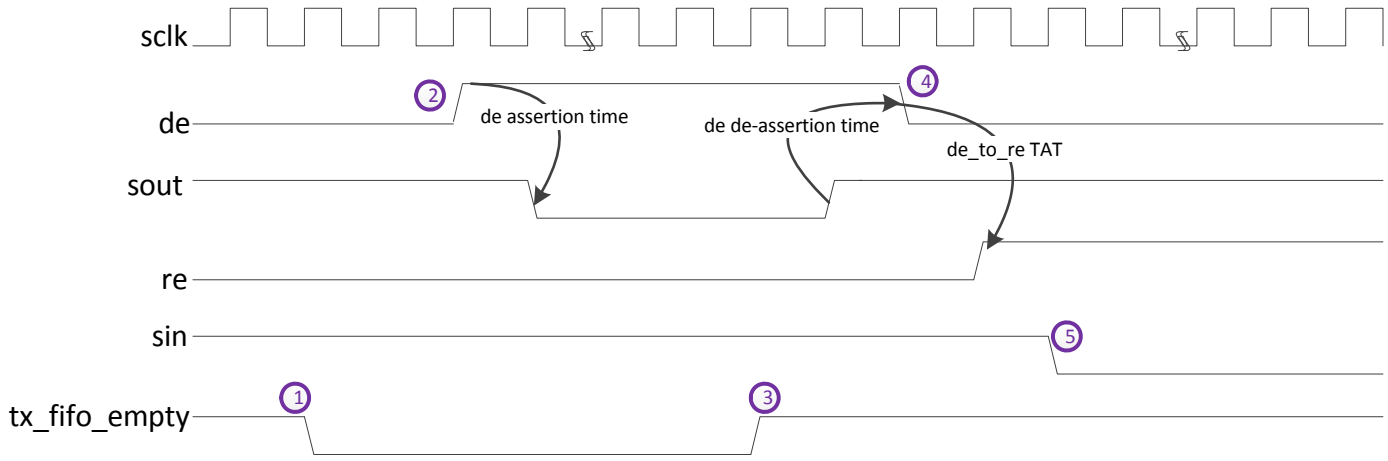


Figure 3-12 shows the following activities at various points in this scenario:

1. The software programs the TX FIFO thereby making 'tx_fifo_empty' signal to go low.
2. The software programs 'DE_EN' and 'RE_EN' to 1. As the data is already present in the TX FIFO, DW_apb_uart controller asserts the 'de' signal. DW_apb_uart stars sending the character after the DET timings.
3. TX FIFO becomes empty.
4. The 'de' signal gets de-asserted after the DET timing. After 'de_to_re' TAT, 're' signal gets asserted.
5. DW_apb_uart controller starts receiving the incoming character.

3.4 Fractional Baud Rate Support

DW_apb_uart supports fractional baud rate that enables a user to program the fractional part of the divisor value to generate fractional baud rate that results in reduced frequency error. The UART interface usage has been evolving to include ever increasing baud rate speeds. The DW_apb_uart needs to be software configurable to handle the baud rates within 2% frequency error.

The Baud rate of DW_apb_uart is controlled by sclk in asynchronous serial clock (CLOCK_MODE=2) implementation or pclk in single clock implementation (CLOCK_MODE=1) and the Divisor Latch Register (DLH and DLL).

The baud rate is determined by the following factors:

- Serial clock operating frequency (sclk in Asynchronous serial clock implementation or pclk in single clock implementation)
- The desired baud rate.
- The baud rate generator divisor value, DIVISOR (composed of DLH & DLL registers).
- The acceptable Baud-rate error, %ERROR

The equation to calculate the baud rate is as follows:

$$\text{Baud Rate} = \frac{\text{Serial Clock Operating Frequency}}{(16 \times \text{Baud Rate})} \quad (1)$$

Where,

DIVISOR – Number (in hexadecimal) to program the DLL and DLH.

Serial clock frequency – Frequency at sclk or pclk pin of DW_apb_uart.

From Equation (1), DIVISOR can be calculated as:

$$\text{DIVISOR} = \frac{\text{Serial Clock Operating Frequency}}{(16 \times \text{Baud Rate})} \quad (2)$$

Also from Equation (1), it can also be shown that:

$$\text{Serial clock frequency} = \text{Baud Rate} \times 16 \times \text{DIVISOR} \quad (3)$$

The Error between the Baud rate and Baud rate (selected) is given as:

$$\text{Percentage ERROR} = \frac{|\text{Baud Rate} - \text{Baud Rate (selected)}|}{\text{Baud Rate}} \times 100 \quad (4)$$

Note Fractional Baud rate is supported only when the source license DWC-APB-Advanced-Source exists.

Configuration of the DW_apb_uart for Fractional Baud Rate does the following:

- The configurable parameter DLF_SIZE is used to choose the width of the register that stores fractional part of the divisor.
- The fractional value of the divisor is programmed in the Divisor Latch Fraction Register (DLF) register.

The programmable fractional baud rate divisor enables a finer resolution of baud clock than the conventional integer divider. The programmable fractional baud clock divider allows for the programmability of both an integer divisor as well as fractional component. The average frequency of the baud clock from the fractional baud rate divisor is dependent upon both the integer divisor and the fractional component, thereby providing a finer resolution to the average frequency of the baud clock.

$$\text{Baud Rate Divisor} = \frac{\text{Serial Clock Frequency}}{(16 \times \text{Required Baud Rate})} = \text{BRD}_\text{I} + \text{BRD}_\text{F} \quad (5)$$

Where,

BRD_I - Integer part of the divisor.

BRD_F - Fractional part of the divisor.

3.4.1 Fractional Division Used to Generate Baud Clock

Fractional division of clock is used by the N/N+1 divider, where N is the integer part of the divisor. N/N+1 division works on the basis of achieving the required average timing over a long period by alternating the

division between two numbers. If N=1 and ratio of N/N+1 is same, which means equal number of divide by 1 and divide by 2 over a period of time, average time period would come out to be divided by 1.5. Varying the ratio of N/N+1 any value can be achieved above 1 and below 2.

3.4.2 Calculating the Fractional Value Error

Following is a sample for calculating the fractional value error.

Consider the following values:

- Required Baud Rate (RBR) = 4454400
- Serial Clock (SCLK) = 133MHz
- DLF_SIZE = 4

Then, as per equation (5), Baud Rate Divisor (BRD) is as follows:

$$\text{BRD} = \frac{133}{16 \times 4454400} = 1.866132364 \quad (6)$$

In (6), the integer and fractional parts are as follows:

- Integer part (BRD_I) = 1
- Fractional part (BRD_F) = 0.866132364

Therefore, Baud Rate Divisor Latch Fractional Value (DLF) is as follows:

$$\text{DLF} = \text{BRD}_\text{F} \times 2^{\text{DLF\_SIZE}} = 0.866132364 \times 16 = 13.858117824 = 14 \text{ (roundoff value)} \quad (7)$$

The Generated Baud Rate Divider (GD) is as follows:

$$\text{GD} = \text{BRD}_\text{I} + \frac{\text{DLF}}{2^{\text{DLF\_SIZE}}} = 1 + \frac{14}{16} = 1.875 \quad (8)$$

Therefore, the Generated Baud Rate (GBR) is as follows:

$$\text{GBR} = \frac{\text{Serial Clock}}{(16 \times \text{GD})} = \frac{133}{16 \times 1.875} = 4433333.333 \quad (9)$$

Now the error is calculated as follows:

$$\text{Error} = \frac{\text{GBR} - \text{RBR}}{\text{RBR}} = 0.004729 \quad (11)$$

The error percentage is as follows:

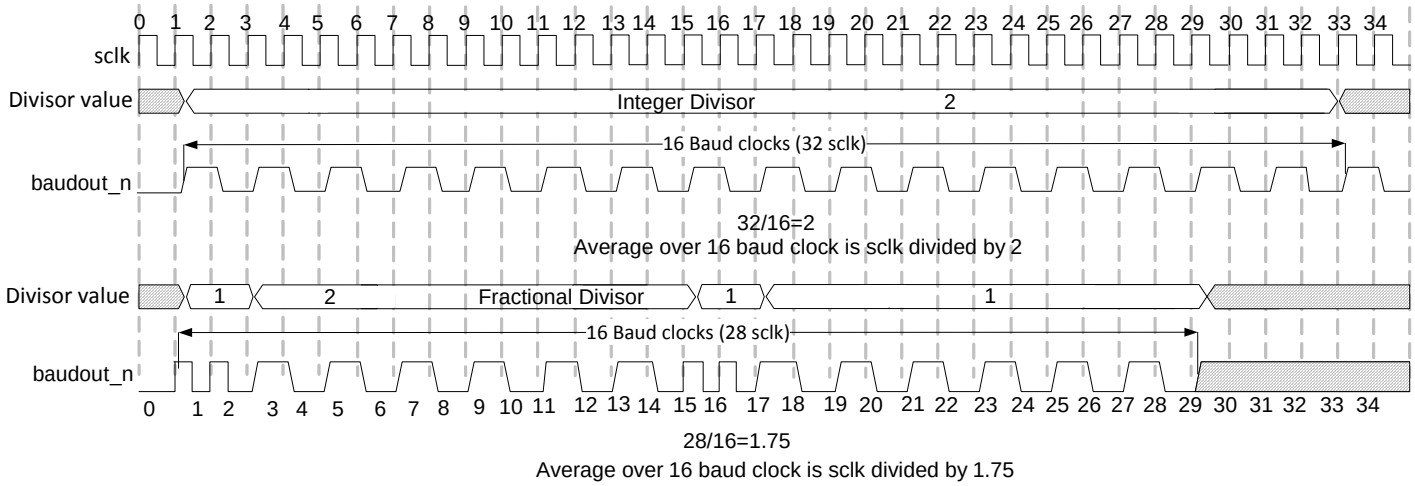
$$\text{Error \%} = 0.004729 \times 100 = 0.473 \quad (12)$$

3.4.2.1 Timing Waveforms

If serial clock is 25 MHz and Baud rate required = 892857, divisor comes out to be 1.75. Without a fractional division a value of 1 or 2 will result in baud rate of 1562500 or 781250 which is more than 2% frequency error. However, if we divide 12 clocks by 2 and then 4 clocks by 1, over an average period of 16 clocks we'll achieve division by 1.75. Using this divisor baud rate of 892857 can be achieved.

As shown in the Figure 3-13, the fractional baud clock is generated between N(1) and N+1(2) values to generate the fractional baud rate of 1.75 to achieve the divisor baud rate of 892857 with 0% frequency error compared to 12.49% frequency error in integer baud clock generator.

Figure 3-13 Example of Integer and Fractional Division Over 16 Clock Periods



3.5 IrDA 1.0 SIR Protocol

The Infrared Data Association (IrDA) 1.0 Serial Infrared (SIR) mode supports bi-directional data communications with remote devices using infrared radiation as the transmission medium. IrDA 1.0 SIR mode specifies a maximum baud rate of 115.2 Kbaud.

Attention Information provided on IrDA SIR mode in this section assumes that the reader is fully familiar with the IrDa Serial Infrared Physical Layer Specifications. This specification can be obtained from the following website:
http://www.irda.org

The data format is similar to the standard serial—sout and sin—data format. Each data character is sent serially in this order:

1. Begins with a start bit
2. Followed by 8 data bits
3. Ends with at least one stop bit

Thus, the number of data bits that can be sent is fixed. No parity information can be supplied, and only one stop bit is used in this mode. Trying to adjust the number of data bits sent or enable parity with the Line Control Register (LCR) has no effect.

Configuration of the DW_apb_uart for IrDA 1.0 SIR does the following:

- Bit 6 of the Mode Control Register (MCR) enables or disables the IrDA 1.0 SIR mode.
- Disabling IrDA SIR mode causes the logic to not be implemented; the mode cannot be activated, which reduces total gate counts.
- When IrDA SIR mode is enabled and active, serial data is transmitted and received on the sir_out_n and sir_in ports, respectively.

Note To enable SIR mode, write the appropriate value to the MCR register before writing to the LCR register. For details of the recommended programming sequence, refer to "Programing Examples" on page 177.

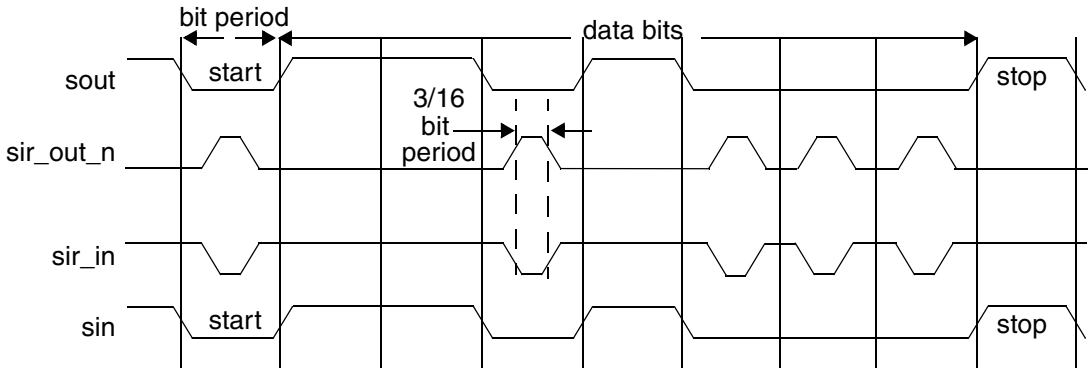
Transmission or non-transmission of a single infrared pulse indicates the following:

- Transmitting a single infrared pulse indicates logic 0
- Non-transmission of a pulse indicates logic 1

The width of each pulse is 3/16ths of a normal serial bit time. Thus, each new character begins with an infrared pulse for the start bit. However, received data is inverted from transmitted data due to infrared pulses energizing the photo transistor base of the IrDA receiver, which pulls its output low. This inverted transistor output is then fed to the DW_apb_uart sir_in port, which gives it the correct UART polarity.

Figure 3-14 shows the timing diagram for the IrDA SIR data format in comparison to standard serial format.

Figure 3-14 IrDA SIR Data Format



As previously mentioned, the DW_apb_uart can be configured to support a low-power reception mode. When the DW_apb_uart is configured in this mode, it is possible to receive SIR pulses of 1.41 microseconds

(minimum pulse duration), as well as nominal 3/16 of a normal serial bit time. In order to use this low-power reception mode, you must program the Low Power Divisor Latch (LPDLL/LPDLH) registers.

For all sclk frequencies greater than or equal to 7.37MHz, pulses of 1.41uS are detectable; these pulses comply with the requirements of the Low Power Divisor Latch registers. However, there are several values of sclk that do not allow detection of such a narrow pulse, as indicated in Table 3-1.

Table 3-1 Narrow Pulse Exceptions

SCLK	Low Power Divisor Latch Register Value	Min Pulse Width for Detection *
1.84MHz	1	3.77uS
3.69MHz	2	2.086uS
5.53MHz	3	1.584uS

* 10% has been added to the internal pulse width signal to cushion the effect of pulse reduction due to the synchronization and data integrity logic so that a pulse slightly narrower than these may be detectable.

When IrDA SIR mode is enabled, the DW_apb_uart operates in a manner similar to when the mode is disabled, with one exception: data transfers can only occur in half-duplex fashion when IrDA SIR mode is enabled. This is because the IrDA SIR physical layer specifies a minimum of 10ms delay between transmission and reception; this 10ms delay must be generated by software.

3.6 FIFO Support

You can configure the DW_apb_uart to implement FIFOs that buffer transmit and receive data; this is illustrated in Figure 1-2 on page 19. If FIFO support is not selected, then no FIFOs are implemented and only a single receive data byte and transmit data byte can be stored at a time in the RBR and THR registers; this implies a 16450-compatible mode of operation. However, in this mode of operation, most of the enhanced features are unavailable.

In FIFO mode, the FIFOs can be selected to be either of the following:

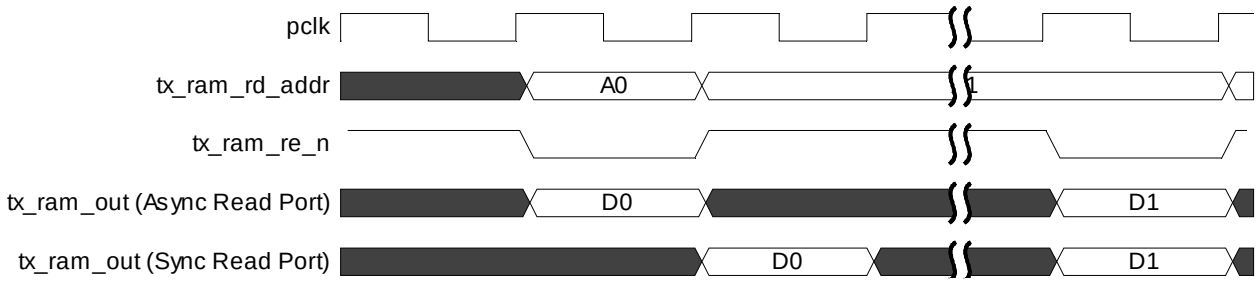
- External customer-supplied FIFO RAMs
- Internal DesignWare D-flip-flop-based RAMs (DW_ram_r_w_s_dff)

If the configured FIFO depth is greater than 256, the FIFO memory selection is restricted to be external. Additionally, selecting internal memory restricts the Memory Read Port Type to D-flip-flop-based Synchronous read port RAMs.

When external RAM support is chosen, either synchronous or asynchronous RAMs can be used. Asynchronous RAM provides read data during the clock cycle that has the memory address and read signals active, for sampling on the next rising clock edge. Synchronous single stage RAM registers the data at the current address out and is not available until the next clock cycle; that is, the second rising clock edge.

Figure 3-15 shows the timing diagram for both asynchronous and synchronous RAMs.

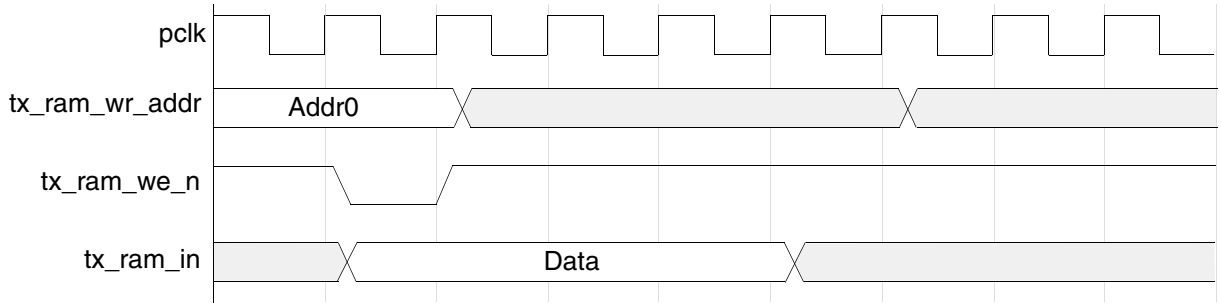
Figure 3-15 Timing for RAM Reads



Note This timing diagram illustrated in Figure 3-15 assumes the RAM has a chip select port that is tied to an active value; therefore, the chip is always enabled. This is why the second synchronous read data appears at the same cycle as the asynchronous read data; that is, the address for the second read has been sampled along with the chip select on an earlier edge. Once the tx_ram_re_n output enable asserts the data, the value on the register output is seen on that same cycle.

Similarly, you can use synchronous RAM for writes, which registers the data at the current address out. Figure 3-16 shows the timing diagram for RAM writes.

Figure 3-16 Timing for RAM Writes



When FIFO support is selected, an optional programmable FIFO Access mode is available for test purposes, which allows:

- Receive FIFO to be written by master
- Transmit FIFO to be read by master

When FIFO Access mode is not selected, none of the corresponding logic is implemented and the mode cannot be enabled, reducing overall gate counts.

When FIFO Access mode has been selected it can be enabled with the FIFO Access Register (FAR[0]). Once enabled, the control portions of the transmit and receive FIFOs are reset and the FIFOs are treated as empty.

Data can be written to the transmit FIFO as normal; however no serial transmission occurs in this mode—normal operation halted—and thus no data leave the FIFO. The data that has been written to the transmit FIFO can be read back with the Transmit FIFO Read (TFR) register, which when read gives the current data at the top of the transmit FIFO.

Similarly, data can be read from the receive FIFO as normal. Since the normal operation of the DW_apb_uart is halted in this mode, data must be written to the receive FIFO so the data can be read back.

Data is written to the receive FIFO using the Receive FIFO Write (RFW) register. The upper two bits of the 10-bit register are used to write framing error and parity error detection information to the receive FIFO, as follows:

- RFW[9] indicates framing error
- RFW[8] indicates parity error

Although these bits cannot be read back through the Receive Buffer Register, they can be checked by reading the Line Status Register and checking the corresponding bits when the data in question is at the top of the receive FIFO.

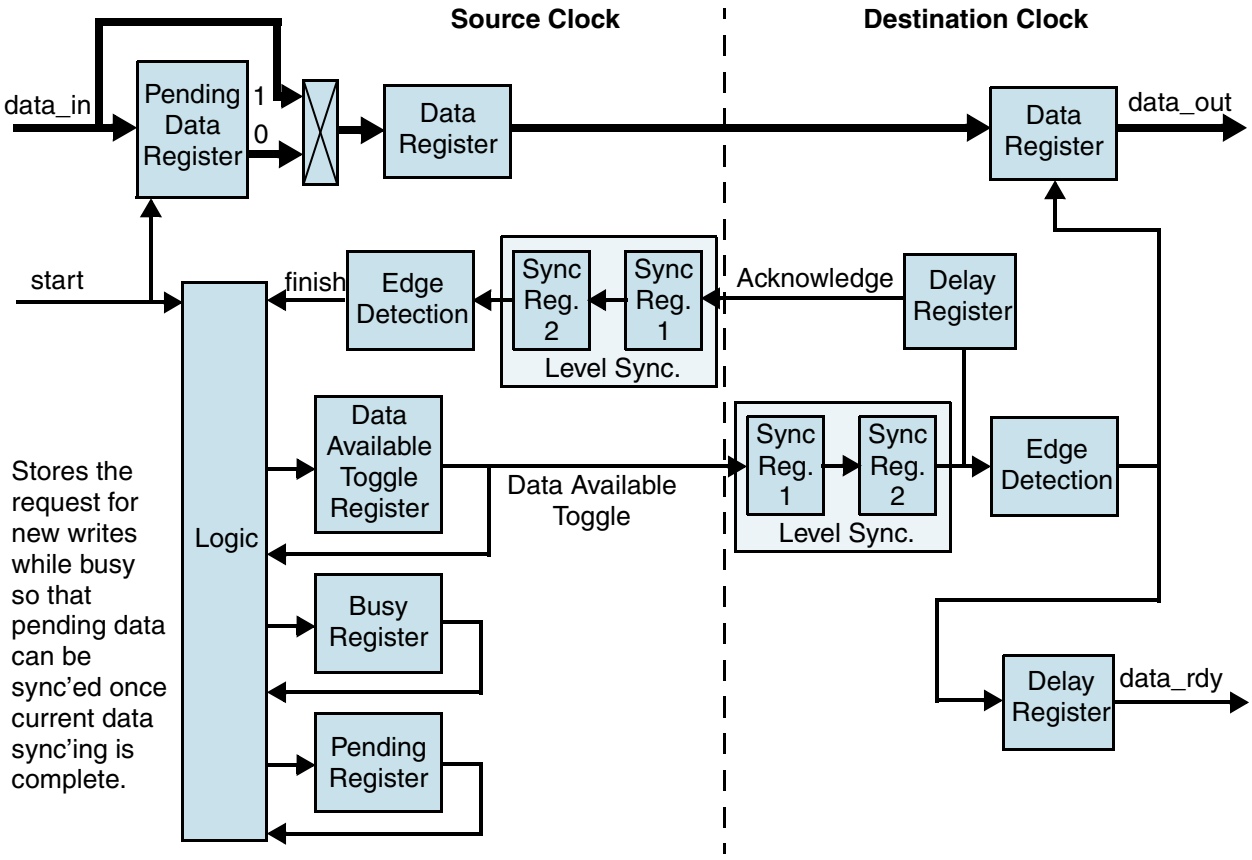
3.7 Clock Support

The DW_apb_uart can be configured to have either one system clock (pclk) or two system clocks (pclk and sclk). The second asynchronous serial clock (sclk) accommodates accurate serial baud rate settings, as well as APB bus interface requirements. When using a single-system clock, available system clock settings for accurate baud rates are greatly restricted.

When a two-clock design is chosen, a synchronization module is implemented for synchronization of all control and data across the two-system clock boundaries; this is illustrated in Figure 1-1 on page 16.

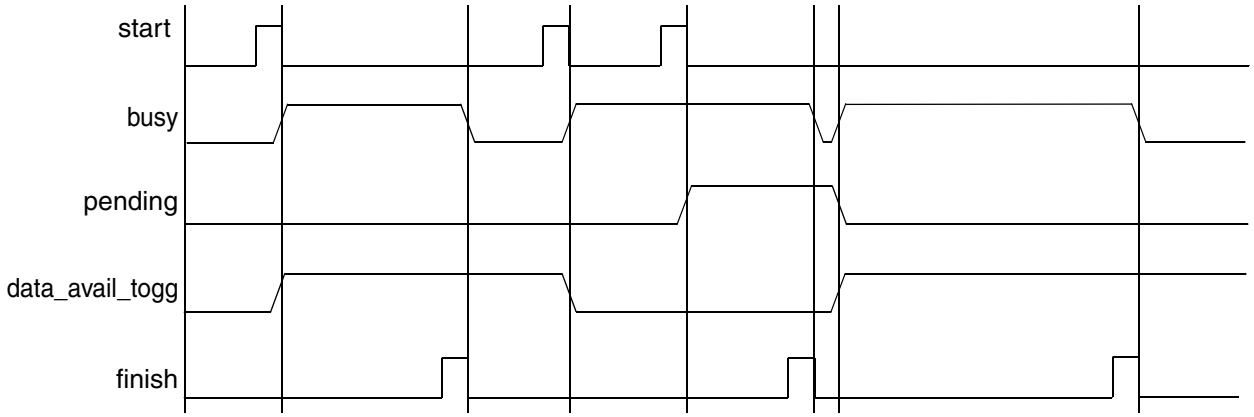
The RTL diagram for the data synchronization module is shown in Figure 3-17; this module can have pending data capability.

Figure 3-17 RTL Diagram of Data Synchronization Module



The timing diagram shown in Figure 3-18 on page 60 shows the data synchronization process.

Figure 3-18 Timing Diagram for Data Synchronization Module



The arrival of new source domain data is indicated by the assertion of start. Since data is now available for synchronization, the process is started and busy status is set. If start is asserted while busy and pending data capability has been selected, the new data is stored.

When no longer busy, the synchronization process starts on the stored pending data. Otherwise the busy status is removed when the current data has been synchronized to the destination domain and the process continues. If only one clock is implemented, all synchronization logic is absent and signals are simply passed through this module.

There are two types of signal synchronization:

- Data-synchronized signals – full synchronization handshake takes place on signals
- Level-synchronized signals – signals are passed through two destination clock registers

Both synchronization types incur additional data path latencies. However, this additional latency has no negative affect on received or transmitted data, other than to limit how much faster sclk can be in relation to pclk for back-to-back serial communications with no idle assertion.

A serial clock that exceeds this limit does not leave enough time for a complete incoming character to be received and pushed into the receiver FIFO. To ensure that you do not exceed the limit, the following equation must hold true:

((2 * *pclk_cycles*) + 4) < (39 * (Baud Divisor))

Where:

pclk_cycles is expressed in sclk cycles

For example, if the Baud Divisor is programmed to 1 and a serial clock is 18 times faster than the pclk signal, the equation becomes:

((2 * 18) + 4) < (39 * 1) ≥ 40 < 39

Thus the equation does not hold true, and the ratio 18:1 (sclk:pclk) exceeds the limit at this Baud rate.

Here are a few things to keep in mind:

- A divisor greater than 1 at a clock ratio of 18:1 (sclk:pclk) does not cause data corruption issues due to synchronization, as the synchronization process has more time to transfer the received data to the peripheral clock domain before the next character bit is received.

  In most cases, however, the pclk signal is faster than sclk, so this should never be an issue.

- There is slightly more time required after initial serial control register programming before serial data can be transmitted or received.
- The serial clock modules must have time to see new register values and reset their respective state machines. This total time is guaranteed to be no more than eight clock cycles of the slower of the two system clocks. Therefore, no data should be transmitted or received before this maximum time expires, after initial configuration.

  Each NOP usually takes one bus cycle to retire. However, the actual number of NOPs that need to be inserted in the assembly code is dependent on the maximum number of instructions that can be retired in a single cycle. So for example, if the processor uses a 4-dispatch pipe, then four NOPs could potentially retire in one bus cycle. Assuming that the next opcode (NOP) is fetched as per the slower

clock—with eight clock cycles of the slower clock as the reference—a minimum of thirty-two NOPs need to be included in the assembly code after a software reset.

In systems where only one clock is implemented, there are no additional latencies.

3.8 Back-to-Back Character Stream Transmission

This section describes:

- Scenarios under which the DW_apb_uart is capable of transmitting back-to-back characters on the serial interface, with no idle time between them
- Worst-case idle time that exists between back-to-back characters

When the Transmit FIFO contains multiple data entries, the DW_apb_uart transmits the characters in the FIFO back-to-back on the serial bus. However, if the CLOCK_MODE configuration parameter equals 2, synchronization delays in the DW_apb_uart can cause an IDLE period between the end of the current STOP bit and the beginning of the next START bit; this appears as an extended STOP bit duration on the serial bus.

3.8.1 Dual Clock Mode

When the CLOCK_MODE parameter equals 2—indicating an asynchronous relationship between pclk and sclk—the DW_apb_uart has a synchronization delay between the transmitter in the sclk domain and the TX FIFO in the pclk domain when querying if another character is ready for transmission. The transmitter begins the handshake one baud clock cycle before the end of the current STOP bit. The duration of the synchronization delay is given by the following equations:

$$sync\_delay = (1sclk + 3pclk) + 1pclk + (1pclk + 3sclk)$$

$$sync\_delay = 4sclk + 5pclk$$

If the *sync_delay* duration is longer than one baud clock period, an IDLE period will be inserted between the end of a STOP bit and the beginning of the next START bit.

To prevent insertion of the IDLE period, the following condition must be true:

$$sync\_delay \leq bclk\_period$$

The baud clock period is given by the following equation:

$$bclk\_period = \{DLH,DLL\} * sclk$$

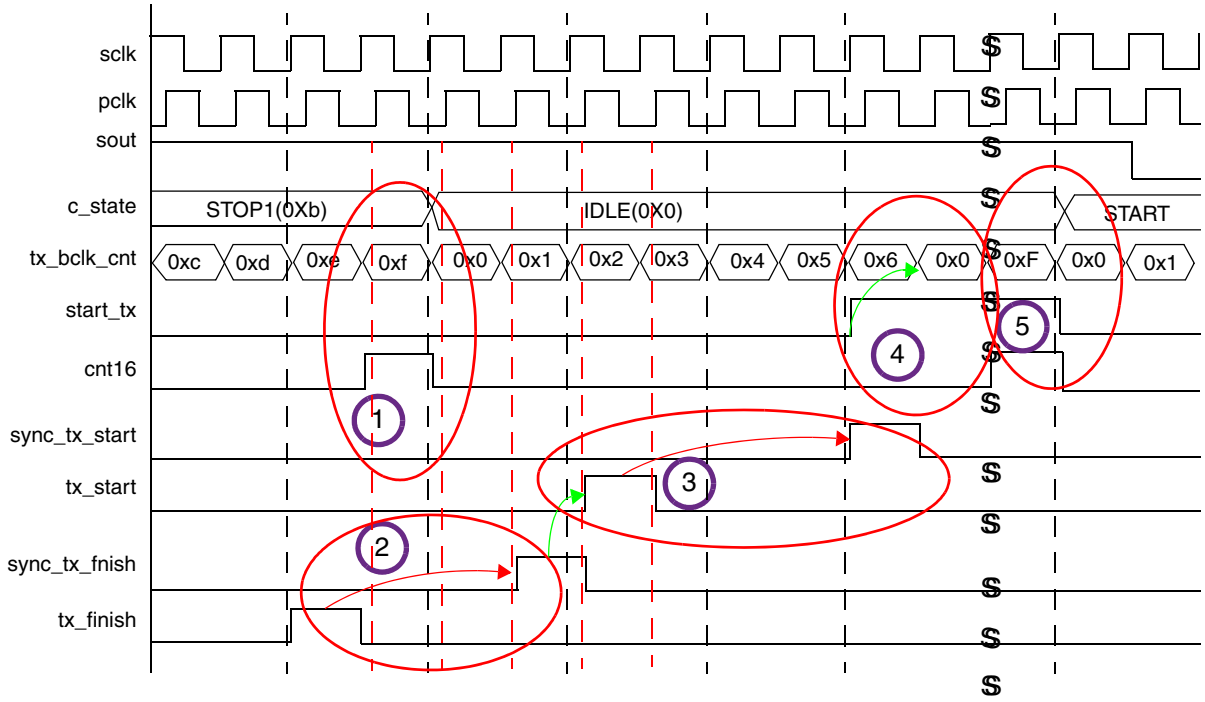
The worst case timing of the inserted IDLE period is given by:

$$worst\_case\_idle\_duration = sync\_delay + (15 * bclk\_period)$$

The *worst_case_idle_duration* can be added to the programmed STOP bit duration to give the overall STOP bit period.

Figure 3-19 illustrates an example of character finish to character start delay.

Figure 3-19 Character Finish to Character Start Delay



1. The baud divisor is set to 1 ({DLH, DLL} = 1), so every sclk is a baud clock cycle. The transmit state machine changes state every sixteen baud clocks—eight in the case of a half STOP bit. At this point in Figure 3-19, after 16 baud clock cycles of the STOP1 state, the state machine enters the IDLE state on the next cycle because start_tx is not yet asserted.
2. One baud clock before the end of the STOP state, the transmit state machine decodes that the current character is complete and asserts tx_finish, which is synchronized to the pclk domain to become sync_tx_finish; this synchronization accounts for the "1sclk + 3pclk" term in *sync_delay*.
3. In the pclk domain, there is a one-pclk cycle delay—"1pclk" term in *sync_delay*—before the signal tx_start is asserted from the assertion of sync_tx_finish. Tx_start must then be synchronized to the sclk domain—"1pclk + 3sclk" term in *sync_delay*—to instruct the state machine to commence the START bit of the next character.
4. Start_tx asserts in the sclk domain, and causes the baud clock counter (tx_bclk_cnt) to go to 0.
5. Once sixteen baud clocks have been counted, the state machine can transition into the START state, and one cycle later sout is de-asserted.

3.8.2 Single Clock Mode

If CLOCK_MODE equals 1, there is no idle time between back-to-back characters if data is ready in the transmit FIFO. In this case, because *sync_delay* equals one pclk as described in "Dual Clock Mode", the requirement to avoid idle time between consecutive characters is met for all {DLH,DLL} values.

$$sync\_delay \leq \{DLH,DLL\}*sclk$$

For example, when {DLH, DLL} equals 1 (bearing in mind that when CLOCK_MODE = 1 : pclk = sclk), then

$$1\ pclk \leq 1*pclk$$

3.9 Interrupts

Assertion of the DW_apb_uart interrupt output signal (intr)—a positive-level interrupt—occurs whenever one of the several prioritized interrupt types are enabled and active.

When an interrupt occurs, the master accesses the IIR register.

The following interrupt types can be enabled with the IER register:

- Receiver Error
- Receiver Data Available
- Character Timeout (in FIFO mode only)
- Transmitter Holding Register Empty at/below threshold (in Programmable THRE interrupt mode)
- Modem Status
- Busy Detect Indication

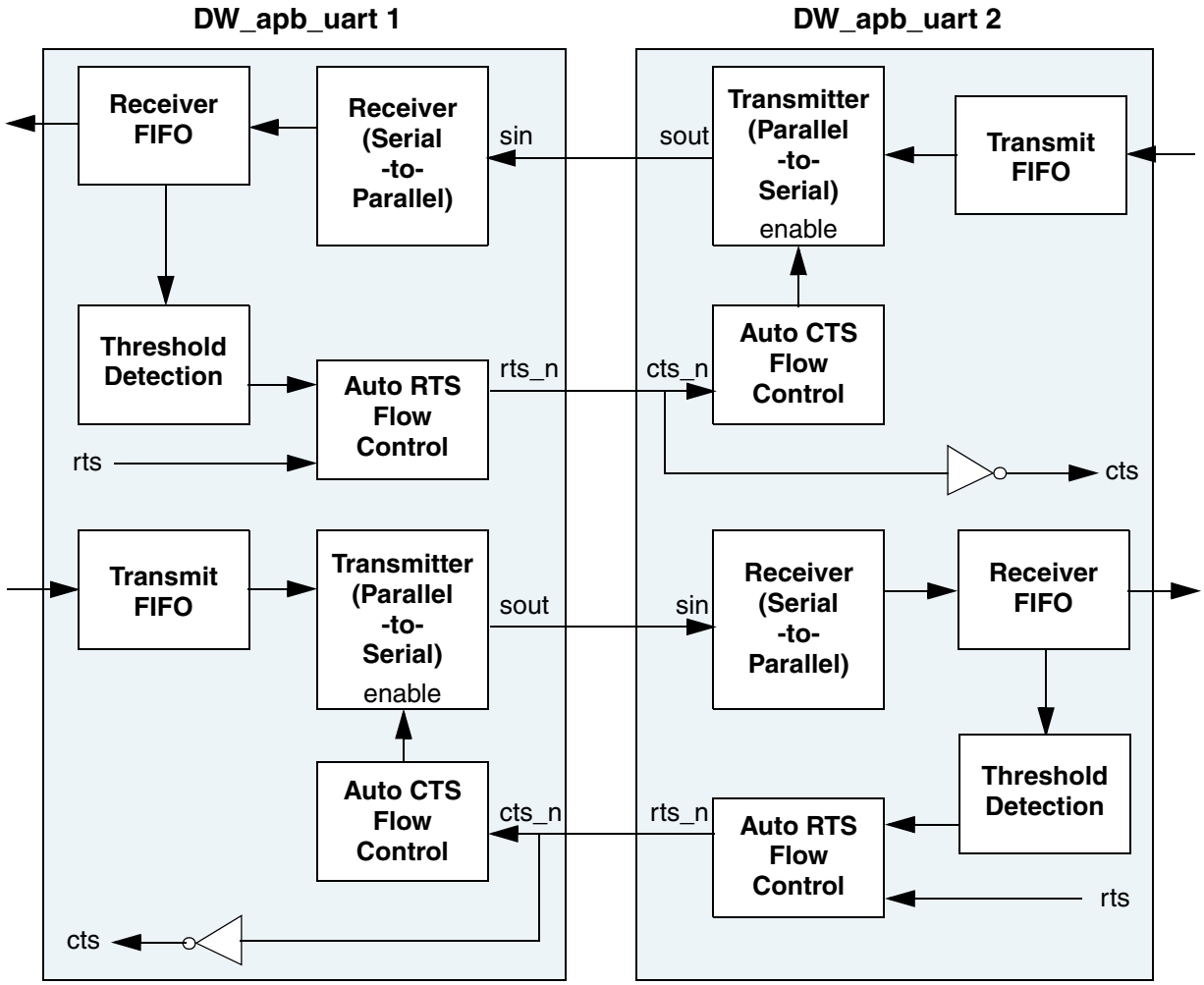
These interrupt types are covered in more detail in Table 6-2 on page 127.

3.10 Auto Flow Control

The DW_apb_uart can be configured to have a 16750-compatible Auto RTS and Auto CTS serial data flow control mode available; if FIFOs are not implemented, this mode cannot be selected. When Auto Flow Control is not selected, none of the corresponding logic is implemented and the mode cannot be enabled, reducing overall gate counts. When Auto Flow Control mode is selected, it can be enabled with the Modem Control Register (MCR[5]).

Figure 3-20 shows a block diagram of the Auto Flow Control functionality.

Figure 3-20 Auto Flow Control Block Diagram



Auto RTS and Auto CTS are described as follows:

- **Auto RTS –** Becomes active when the following occurs:
  - Auto Flow Control is selected during configuration
  - FIFOs are implemented
  - RTS (MCR[1] bit and MCR[5]bit are both set)
  - FIFOs are enabled (FCR[0]) bit is set)
  - SIR mode is disabled (MCR[6] bit is not set)

  When Auto RTS is enabled, the rts_n output is forced inactive (high) when the receiver FIFO level reaches the threshold set by FCR[7:6], but only if the RTC flow-control trigger is disabled. Otherwise, the rts_n output is forced inactive (high) when the FIFO is almost full, where “almost full” refers to two available slots in the FIFO. When rts_n is connected to the cts_n input of another UART device,

the other UART stops sending serial data until the receiver FIFO has available space; that is, until it is completely empty.

The selectable receiver FIFO threshold values are:

- 1
- ¼
- ½
- 2 less than full

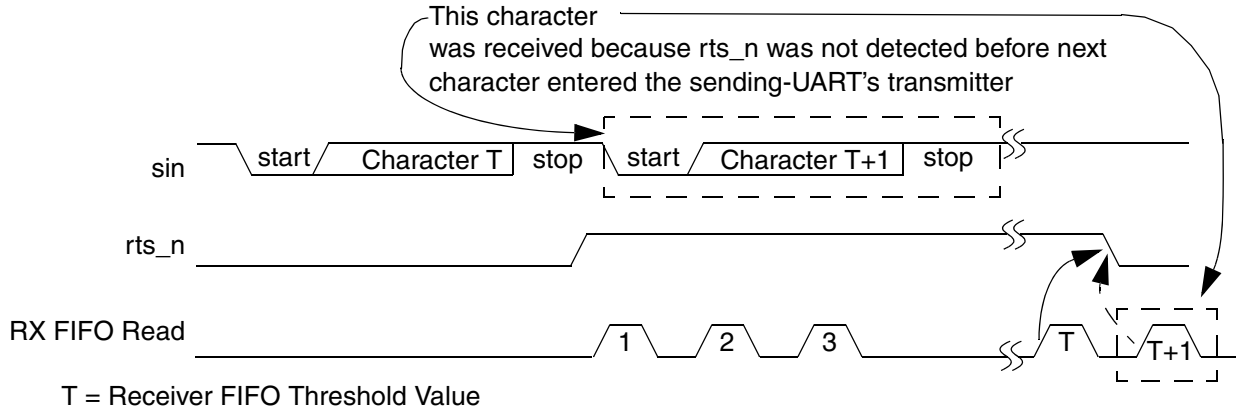
Since one additional character can be transmitted to the DW_apb_uart after rts_n has become inactive—due to data already having entered the transmitter block in the other UART—setting the threshold to "2 less than full" allows maximum use of the FIFO with a safety zone of one character.

Once the receiver FIFO becomes completely empty by reading the Receiver Buffer Register (RBR), rts_n again becomes active (low), signalling the other UART to continue sending data.

Note Even if everything else is selected and the correct MCR bits are set, if the FIFOs are disabled through FCR[0] or the UART is in SIR mode (MCR[6] is set to 1), Auto Flow Control is also disabled. When Auto RTS is not implemented or disabled, rts_n is controlled solely by MCR[1].

Figure 3-21 shows a timing diagram of the Auto RTS operation.

Figure 3-21 Auto RTS Timing



- **Auto CTS** – becomes active when the following occurs:
  - Auto Flow Control is selected during configuration
  - FIFOs are implemented
  - AFCE (MCR[5] bit = 1)
  - FIFOs are enabled through FIFO Control Register FCR[0] bit
  - SIR mode is disabled (MCR[6] bit = 0)

  When Auto CTS is enabled (active), the DW_apb_uart transmitter is disabled whenever the cts_n input becomes inactive (high); this prevents overflowing the FIFO of the receiving UART.

If the cts_n input is not inactivated before the middle of the last stop bit, another character is transmitted before the transmitter is disabled. While the transmitter is disabled, the transmitter FIFO can still be written to, and even overflowed.

Therefore, when using this mode, the following happens:

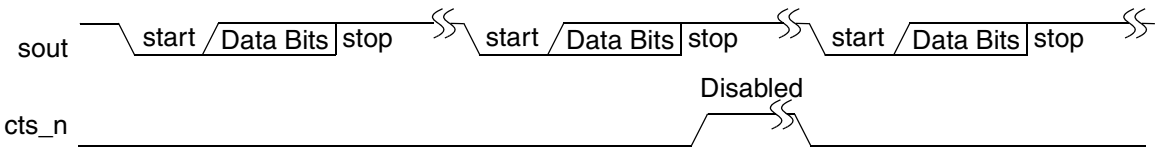
- UART status register can be read to check if transmit FIFO is full (USR[1] set to 0)
- Current FIFO level can be read using TFL register
- Programmable THRE Interrupt mode must be enabled to access “FIFO full” status using Line Status Register (LSR)

When using the “FIFO full” status, software can poll this before each write to the Transmitter FIFO; for details, refer to “Programmable THRE Interrupt” on page 68. When the cts_n input becomes active (low) again, transmission resumes.

Note When everything else is selected, if the FIFOs are disabled using FCR[0], Auto Flow Control is also disabled. When Auto CTS is not implemented or disabled, the transmitter is unaffected by cts_n.

Figure 3-22 illustrates a timing diagram that shows the Auto CTS operation.

Figure 3-22 Auto CTS Timing



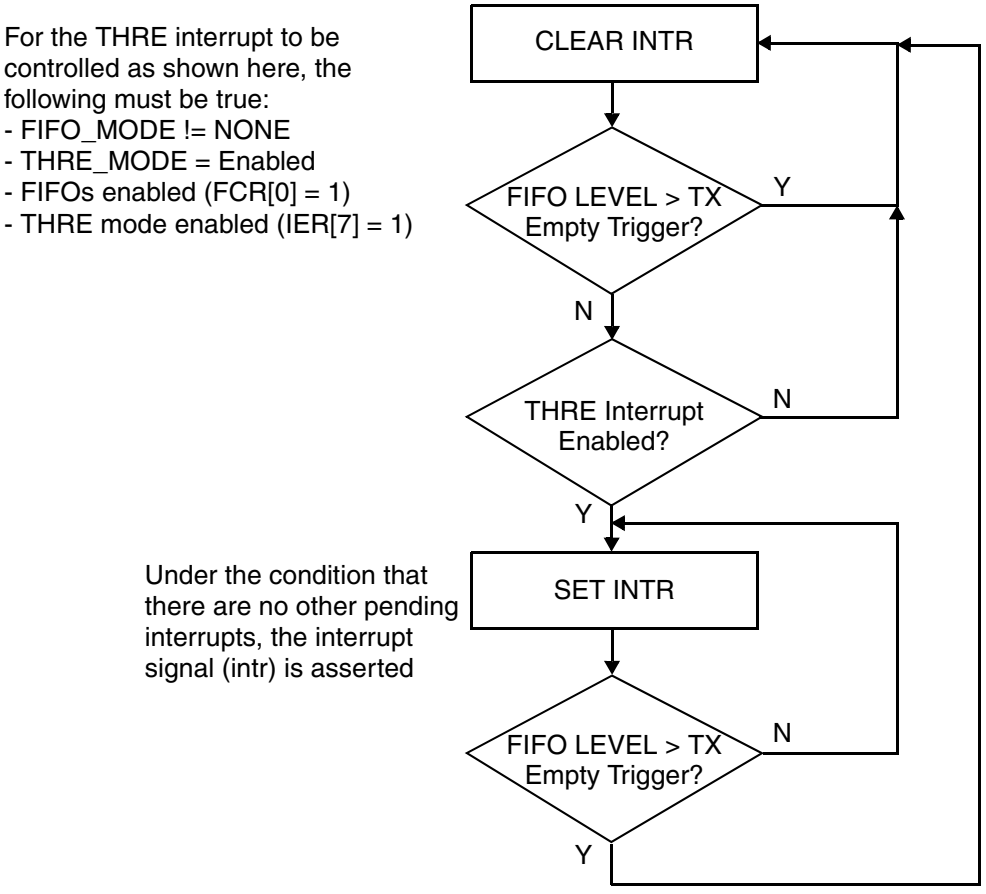
3.11 Programmable THRE Interrupt

The DW_apb_uart can be configured for a Programmable THRE Interrupt mode in order to increase system performance; if FIFOs are not implemented, then this mode cannot be selected.

- When Programmable THRE Interrupt mode is not selected, none of the logic is implemented and the mode cannot be enabled, reducing the overall gate counts.
- When Programmable THRE Interrupt mode is selected, it can be enabled using the Interrupt Enable Register (IER[7]).

When FIFOs and THRE mode are implemented and enabled, the THRE Interrupts and dma_tx_req_n are active at, and below, a programmed transmitter FIFO empty threshold level, as opposed to empty, as shown in the flowchart in Figure 3-23.

Figure 3-23 Flowchart of Interrupt Generation for Programmable THRE Interrupt Mode



The threshold level is programmed into FCR[5:4]. Available empty thresholds are:

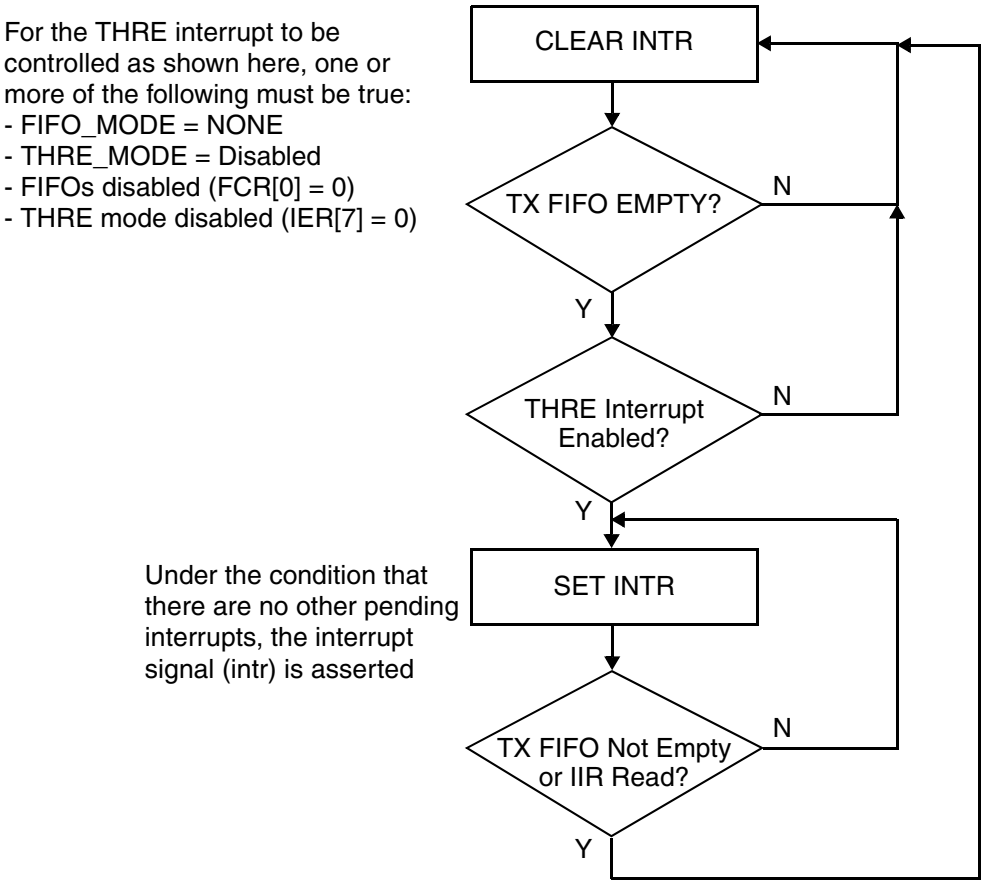
- empty
- 2
- ¼
- ½

Selection of the best threshold value depends on the system's ability to begin a new transmission sequence in a timely manner. However, one of these thresholds should be optimal for increasing system performance by preventing the transmitter FIFO from running empty. For threshold setting details, refer to "FCR" on page 129.

In addition to the interrupt change, the Line Status Register (LSR[5]) also switches from indicating that the transmitter FIFO is empty to the FIFO being full. This allows software to fill the FIFO for each transmit sequence by polling LSR[5] before writing another character. The flow then allows the transmitter FIFO to be filled whenever an interrupt occurs and there is data to transmit, rather than waiting until the FIFO is completely empty. Waiting until the FIFO is empty causes a reduction in performance whenever the system is too busy to respond immediately. Further system efficiency is achieved when this mode is enabled in combination with Auto Flow Control.

Even if everything else is selected and enabled, if the FIFOs are disabled using the FCR[0] bit, the Programmable THRE Interrupt mode is also disabled. When not selected or disabled, THRE interrupts and the LSR[5] bit function normally, signifying an empty THR or FIFO. Figure 3-24 illustrates the flowchart of THRE interrupt generation when not in programmable THRE interrupt mode.

Figure 3-24 Flowchart of Interrupt generation when not in Programmable THRE Interrupt Mode



3.12 Clock Gate Enable

The DW_apb_uart can be configured to have a clock gate enable output.

- When the clock gate enable option is not selected, no logic is implemented, which reduces the overall gate count.
- When the clock gate enable option is selected, the clock gate enable signal(s)—uart_lp_req_pclk for single clock implementations or uart_lp_req_pclk and uart_lp_req_sclk for two clock implementations—is used to indicate the following:
  - Transmit and receive pipeline is clear (no data).
  - No activity has occurred.
  - Modem control input signals have not changed in more than one character time—the time taken to TX/RX a character—so that clocks can be gated.

    A character is made up of:

    start_bit + *data_bits* + *parity* (optional) + *stop_bit(s)*)

The assertion of clock gate enable signals is an indication that the UART is inactive, so clocks may be gated in order to put the device in a low-power (lp) mode. Therefore, the following must be true for at least one character time for the assertion of the clock gate enable signal(s) to occur:

- No data in the RBR (in non-FIFO mode) or the RX FIFO is empty (in FIFO mode)
- No data in the THR (in non-FIFO mode) or the TX FIFO is empty (in FIFO mode)
- sin/sir_in and sout/sir_out_n are inactive (sin/sir_in are kept high and sout is high or sir_out_n is low) indicating no activity
- No change on the modem control input signals

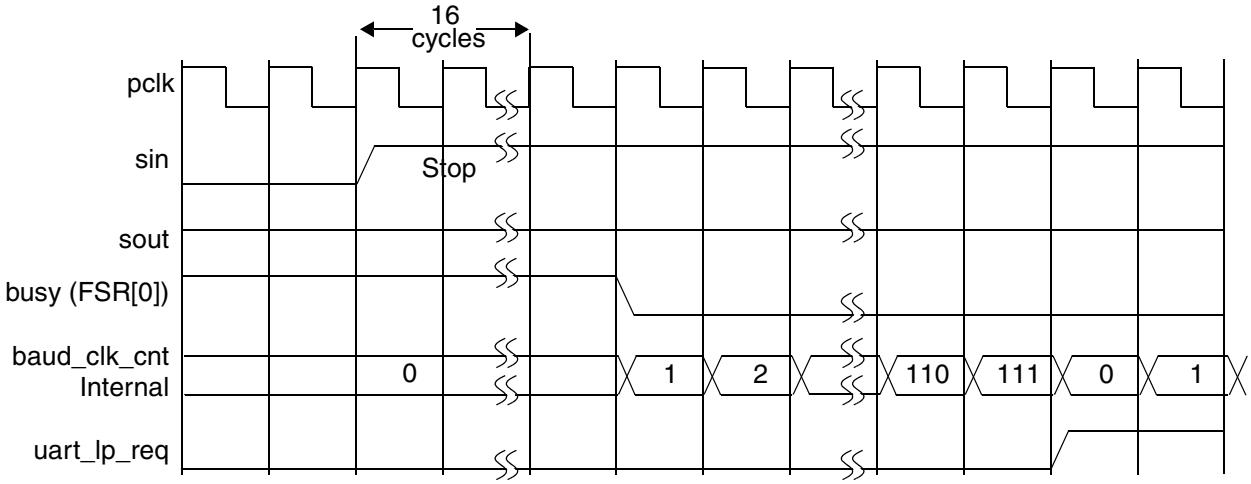
Note, the clock gate enable assertion does not occur in the following modes of operation:

- Loopback mode
- FIFO access mode
- When transmitting a break

For example, assume a DW_apb_uart that is configured to have a single clock (pclk) and is programmed to transmit and receive characters of 7 bits (1 start bit, 5 data bits and 1 stop bit) and the baud clock divisor is set to 1. Therefore, the uart_lp_req_pclk signal is asserted if the transmit and receive pipeline is clear, no activity has occurred and the modem control input signals have not changed for 112 (7×16) pclk cycles.

Figure 3-25 illustrates this example.

Figure 3-25 Clock Gate Enable Timing



When the assertion criteria are no longer met, the clock gate enable signal(s) are de-asserted and the clock(s) is resumed under any of these conditions:

- Either sin signal or sir_in signal goes low
- Write to any of registers is performed
- Modem control input signals have changed when DW_apb_uart is in low-power (sleep) mode

The clock gate enable signals are de-asserted asynchronously on arrival of above mentioned events because a clock is not available to synchronize the events. Therefore, user can decide to include 2-flop syncs externally before the clock gate cell to avoid metastability issues for clock-gate latch.

Note A read to any register does not de-assert the clock gate enable signal(s). The pclk clock needs to be enabled to read any of the registers in low-power mode.

The time taken for the clock(s) to resume is important in preventing receive data synchronization problems, due to the DW_apb_uart RX block sampling:

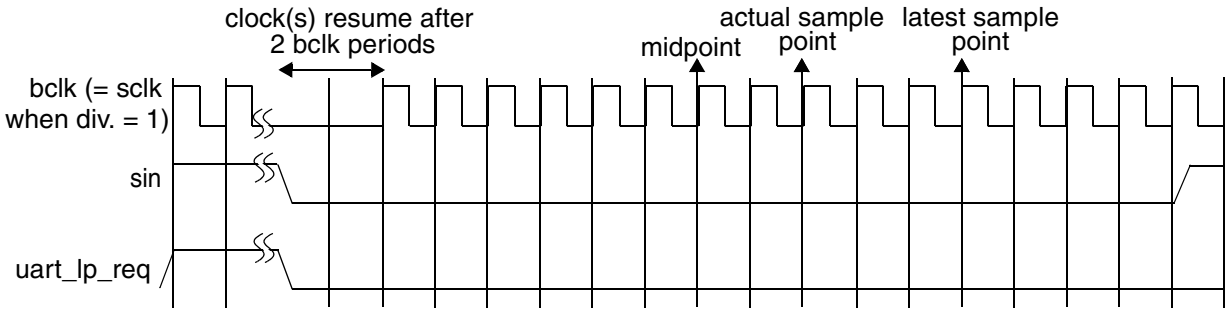
1. At mid-point of each bit period—after approximately 8 baud clocks—in UART (RS323) mode.
2. After that, every 16 baud clocks for a baud divisor of 1 that is 16 sclks; for a single clock implementation, this is 16 pclks.

Thus, if eight or more sclk periods pass before the serial clock starts up again, the DW_apb_uart can get out of synchronization with the serial data it is receiving; that is, the receiver can sample into the second bit period, and if it is still 0, the receiver uses this as the start bit, and so on.

In order to avoid this problem, the clock should be resumed within five clock periods of the baud clock, which is the same as sclk if the baud divisor is set to 1; this is worst-case. If the divisor is greater, it gives a greater number of sclk cycles available before the clock must resume. This means a sample point at the 13 baud clock (at the latest) out of the 16 that are transmitted for each bit period of the character in non-SIR mode.

Figure 3-26 shows the timing diagram that illustrates the previous scenario.

Figure 3-26 Resuming Clock(s) After Low Power Mode Timing



This synchronization problem is magnified in SIR mode because the pulse width is only 3/16 of a bit period—three baud clocks, which for a divisor of 1 is three sclks; thus, the pulse can be missed completely. The clocks must resume before three baud clock periods elapse. However, if the first character received while in sleep mode is used only for wake-up reasons and the actual character value is unimportant, this may not become a problem.

When the DW_apb_uart is configured to have two clocks, if the timing of the received signal is not affected by the synchronization problem, then the minimum time to receive a character—if the baud divisor is 1—is 112 sclks:

$$1\ start\_bit + 5\ data\_bits + 1\ stop\_bit = 7 \times 16 = 112$$

Therefore, the pclk must be available before 112 sclk cycles pass in order for the received character to be synchronized to the pclk domain and stored in the RBR (in non-FIFO mode) or the RX FIFO (in FIFO mode).

3.13 DMA Support

The DW_apb_uart supports DMA signalling with the use of the dma_tx_req_n and dma_rx_req_n output signals to indicate:

- When data can be read
- When transmit FIFO is empty

For more information on the dma_tx_req_n and dma_rx_req_n signals, refer to "Handshaking Interface Operation" on page 80.

3.13.1 DMA Modes

The DW_apb_uart uses two DMA channels—one for transmit data and one for receive data. There are two DMA modes:

- mode 0 – bit 3 of FIFO Control Register set to 0
- mode 1 – bit 3 of FIFO Control Register set to 1

Note Only DMA mode 0 is available when FIFOs are not implemented or disabled.

3.13.1.1 DMA Mode 0

DMA mode 0 supports single DMA data transfers at a time.

In mode 0, the dma_tx_req_n signal:

- Goes active-low under the following conditions:
  - When Transmitter Holding Register is empty in non-FIFO mode
  - When transmitter FIFO is empty in FIFO mode with Programmable THRE interrupt mode disabled
  - When transmitter FIFO is at or below programmed threshold with Programmable THRE interrupt mode enabled
- Goes inactive when:
  - Single character has been written into Transmitter Holding Register or transmitter FIFO with Programmable THRE interrupt mode disabled
  - Transmitter FIFO is above threshold with Programmable THRE interrupt mode enabled

In mode 0, the dma_rx_req_n signal:

- Goes active-low when single character is available in Receiver FIFO or Receive Buffer Register
- Goes inactive when Receive Buffer Register or Receiver FIFO are empty, depending on FIFO mode

3.13.1.2 DMA Mode 1

DMA mode 1 supports multi-DMA data transfers, where multiple transfers are made continuously until the receiver FIFO has been emptied or the transmit FIFO has been filled.

In mode 1, the dma_tx_req_n signal is asserted:

- When transmitter FIFO is empty with Programmable THRE interrupt mode disabled
- When transmitter FIFO is at or below programmed threshold with Programmable THRE interrupt mode enabled

In mode 1, the dma_tx_req_n signal is de-asserted when the transmitter FIFO is completely full.

In mode 1, the dma_rx_req_n signal is asserted:

- When Receiver FIFO is at or above programmed trigger level
- When character timeout has occurred; ERBFI does not need to be set

In mode 1, the dma_rx_req_n signal is de-asserted when the receiver FIFO becomes empty.

3.13.1.3 Additional DMA Interface

If required for a DMA controller—such as the DW_ahb_dmac—you can use the DMA_EXTRA parameter to configure he DW_apb_uart for additional DMA interface signals. In this case, asserting the fixed DMA

signals—dma_tx_req_n and dma_rx_req_n—is similar to what was detailed in DMA Mode 0 and DMA Mode 1.

When configured for additional DMA signals, the dma_tx_req_n signal is asserted under the following conditions:

- When the Transmitter Holding Register is empty in non-FIFO mode
- When the transmitter FIFO is empty in FIFO mode with Programmable THRE interrupt mode disabled
- When the transmitter FIFO is at, or below the programmed threshold with Programmable THRE interrupt mode enabled.

When configured for additional DMA signals, the dma_rx_req_n signal is asserted under the following conditions:

- When a single character is available in Receive Buffer Register in non-FIFO mode
- When Receiver FIFO is at or above programmed trigger level in FIFO mode

With the presence of the additional handshaking signals, the UART does not have to rely on internal status and level values to recognize the completion of a request and hence remove the request. Instead, the de-assertion of the DMA transmit and receive request is controlled by the assertion of the DMA transmit and receive acknowledge respectively.

When the UART is configured for additional DMA signals, responsibility of the data flow (transfer lengths) falls on the DMA (DW_ahb_dmac) and is controlled by the programmed burst transaction lengths. Thus, there is no need for DMA modes, and programming the FCR[3] has no effect.

3.13.1.4 Example DMA Flow

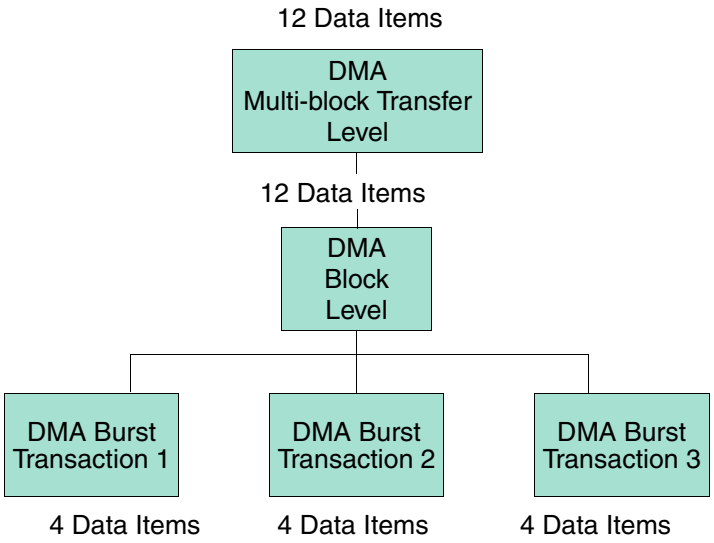
The extra handshaking signals are explained in the following DMA flow for a DW_apb_uart that is configured with FIFOs and Programmable THRE interrupt mode.

As a block flow control device, the DMA Controller is programmed by the processor with the number of data items (block size) that are to be transmitted or received by the DW_apb_uart; this is programmed into the BLOCK_TS field of the CTL*x* register.

The block is broken into a number of transactions, each initiated by a request from the DW_apb_uart. The DMA Controller must also be programmed with the number of data items (in this case, DW_apb_uart FIFO entries) to be transferred for each DMA request. This is also known as the burst transaction length, and is programmed into the SRC_MSIZE/DEST_MSIZE fields of the DW_ahb_dmac CTL*x* register for source and destination, respectively.

Figure 3-27 shows a single block transfer, where the block size programmed into the DMA Controller is 12 and the burst transaction length is set to 4.

Figure 3-27 Breakdown of DMA Transfer into Burst Transactions



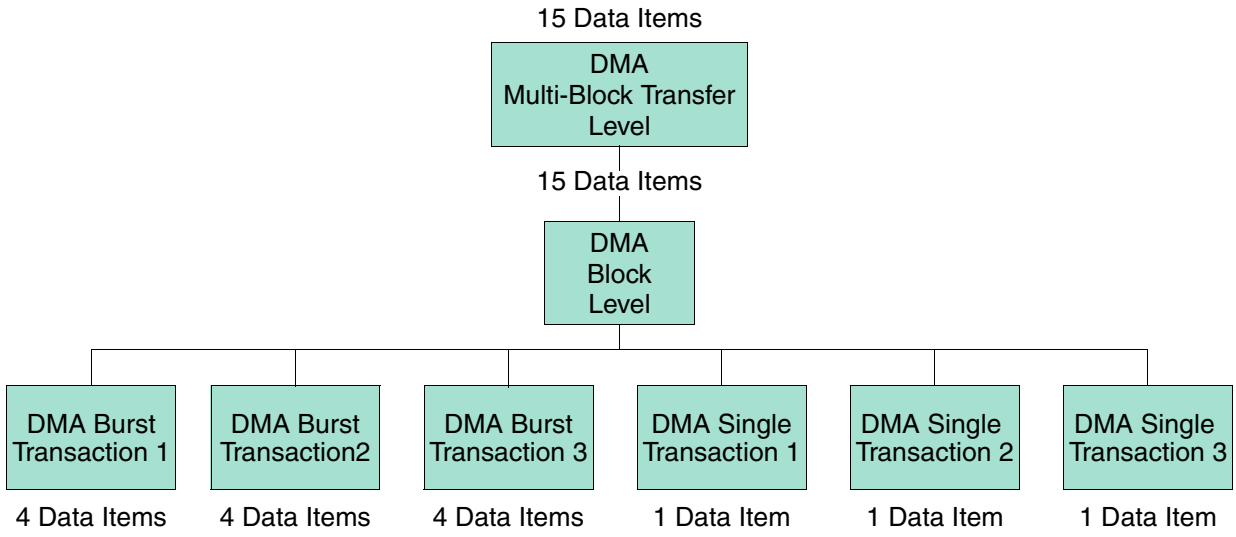
Block Size : DMA.CTL*x*.BLOCK_TS=12
Number of data items per source burst transaction : DMA.CTL*x*.SRC_MSIZE = 4
For a FIFO depth of 16: UART.FCR[7:6] = 01 = FIFO 1/4 full = DMA.CTL*x*.SRC_MSIZE
(for more information, refer to discussion on page 79)

In this case, the block size is a multiple of the burst transaction length. Therefore, the DMA block transfer consists of a series of burst transactions. If the DW_apb_uart makes a transmit request to this channel, four data items are written to the DW_apb_uart transmit FIFO. Similarly, if the DW_apb_uart makes a receive request to this channel, four data items are read from the DW_apb_uart receive FIFO. Three separate requests must be made to this DMA channel before all twelve data items are written or read.

Note The source and destination transfer width settings in the DW_ahb_dmac – DMA.CTLx.SRC_TR_WIDTH and DMA.CTLx.DEST_TR_WIDTH – should be set to 3'b000 because the DW_apb_uart FIFOs are 8 bits wide.

When the block size programmed into the DMA Controller is not a multiple of the burst transaction length, as shown in Figure 3-28, a series of burst transactions followed by single transactions are needed to complete the block transfer.

Figure 3-28 Breakdown of DMA Transfer into Single and Burst Transactions



Block Size : DMA.CTLx.BLOCK_TS=15
Number of data items per burst transaction : DMA.CTLx.DEST_MSIZE = 4
For a FIFO depth of 16: UART.FCR[5:4] = 10 =FIFO 1/4 full = 4 = DMA.CTLx.DEST_MSIZE
(for more information, refer to discussion on page 78)

3.13.2 Transmit Watermark Level and Transmit FIFO Underflow

During DW_apb_uart serial transfers, transmit FIFO requests are made to the DW_ahb_dmac whenever the number of entries in the transmit FIFO is less than or equal to the decoded level of the Transmit Empty Trigger (TET) of the FCR register (bits 5:4); this is known as the watermark level. The DW_ahb_dmac responds by writing a burst of data to the transmit FIFO buffer, of length CTL*x*.DEST_MSIZE.

Data should be fetched from the DMA often enough for the transmit FIFO to perform serial transfers continuously; that is, when the FIFO begins to empty, another DMA request should be triggered. Otherwise the FIFO runs out of data (underflow). To prevent this condition, you must set the watermark level correctly.

3.13.3 Choosing Transmit Watermark Level

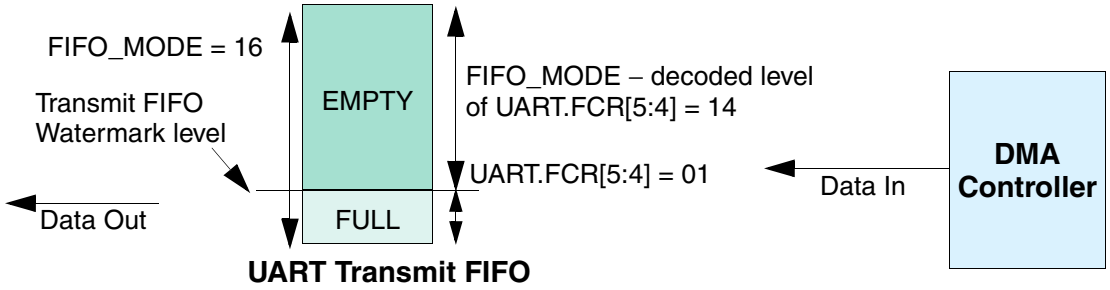
Consider the example where the following assumption is made:

DMA.CTL*x*.DEST_MSIZE = FIFO_DEPTH - UART.FCR[5:4]

The number of data items to be transferred in a DMA burst is equal to the empty space in the Transmit FIFO. Consider two different watermark level settings.

3.13.3.1 Case 1: FCR[5:4] = 01 — decodes to 2

Figure 3-29 Case 1 Watermark Levels



- Transmit FIFO watermark level = decoded level of UART.FCR[5:4] = 2
- DMA.CTL*x*.DEST_MSIZE = FIFO_MODE – UART.FCR[5:4] = 14
- UART transmit FIFO_MODE = 16
- DMA.CTL*x*.BLOCK_TS = 56

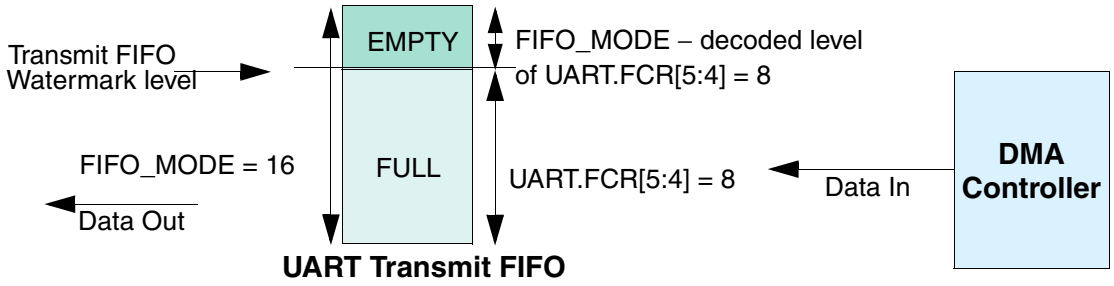
Therefore, the number of burst transactions needed equals the block size divided by the number of data items per burst:

DMA.CTL*x*.BLOCK_TS/DMA.CTL*x*.DEST_MSIZE = 56/14 = 4

The number of burst transactions in the DMA block transfer is 4., but the watermark level—decoded level of UART.FCR[5:4]—is quite low. Therefore, the probability of a UART underflow is high where the UART serial transmit line needs to transmit data, but where there is no data left in the transmit FIFO. This occurs because the DMA has not had time to service the DMA request before the transmit FIFO becomes empty.

3.13.3.2 Case 2: FCR[5:4] = 11 — FIFO 1/2 full (decodes to 8)

Figure 3-30 Case 2 Watermark Levels



- Transmit FIFO watermark level = decoded level of UART.FCR[5:4] = 8
- DMA.CTLx.DEST_MSIZE = FIFO_MODE - UART.FCR[5:4] = 8
- UART transmit FIFO_MODE = 16
- DMA.CTLx.BLOCK_TS = 56

Number of burst transactions in Block:

DMA.CTLx.BLOCK_TS/DMA.CTLx.DEST_MSIZE = 56/8 = 7

In this block transfer, there are seven destination burst transactions in a DMA block transfer, but the watermark level—decoded level of UART.FCR[5:4]—is high. Therefore, the probability of a UART underflow is low because the DMA controller has enough time to service the destination burst transaction request before the UART transmit FIFO becomes empty.

Thus, the second case has a lower probability of underflow at the expense of more burst transactions per block. This provides a potentially greater amount of AMBA bursts per block and worse bus utilization than Case 1.

Therefore, the goal in choosing a watermark level is to minimize the number of transactions per block, while at the same time keeping the probability of an underflow condition to an acceptable level. In practice, this is a function of the ratio of:

rate of UART data transmission : *rate of DMA response to destination burst requests*

For example, both of the following increases the rate at which the DMA controller can respond to burst transaction requests:

- Promoting channel to highest priority channel in DMA
- Promoting DMA master interface to highest priority master in AMBA layer

This in turn enables the user to decrease the watermark level, which improves bus utilization without compromising the probability of an underflow occurring.

3.13.4 Selecting DEST_MSIZE and Transmit FIFO Overflow

As can be seen from Figure 3-30 on page 77, programming DMA.CTL*x*.DEST_MSIZE to a value greater than the watermark level that triggers the DMA request can cause overflow when there is not enough space in the UART transmit FIFO to service the destination burst request. Therefore, use the following in order to avoid overflow:

DMA.CTL*x*.DEST_MSIZE <= UART.FIFO_DEPTH – decoded level of UART.FCR[5:4] **(1)**

In Case 2: FCR[5:4] = 11 — FIFO 1/2 full (decodes to 8), the amount of space in the transmit FIFO at the time the burst request is made is equal to the destination burst length, DMA.CTL*x*.DEST_MSIZE. Thus, the transmit FIFO can be full, but not overflowed, at the completion of the burst transaction.

Therefore, for optimal operation, DMA.CTL*x*.DEST_MSIZE should be set at the FIFO level that triggers a transmit DMA request; that is:

DMA.CTLx.DEST_MSIZE = UART.FIFO_DEPTH – decoded level of UART.FCR[5:4] **(2)**

This is the setting used in Figure 3-28 on page 76.

Adhering to equation (2) reduces the number of DMA bursts needed for a block transfer, which in turn improves AMBA bus utilization.

Note The transmit FIFO is not full at the end of a DMA burst transfer if the UART has successfully transmitted one data item or more on the UART serial transmit line during the transfer.

3.13.5 Receive Watermark Level and Receive FIFO Overflow

During DW_apb_uart serial transfers, receive FIFO requests are made to the DW_ahb_dmac whenever the number of entries in the receive FIFO is at or above the decoded level of Receiver Trigger (RT) of the FCR[7:6]. This is known as the watermark level. The DW_ahb_dmac responds by fetching a burst of data from the receive FIFO buffer of length CTLx.SRC_MSIZE.

Data should be fetched by the DMA often enough for the receive FIFO to accept serial transfers continuously; that is, when the FIFO begins to fill, another DMA transfer is requested. Otherwise, the FIFO fills with data (overflow). To prevent this condition, you must correctly set the watermark level.

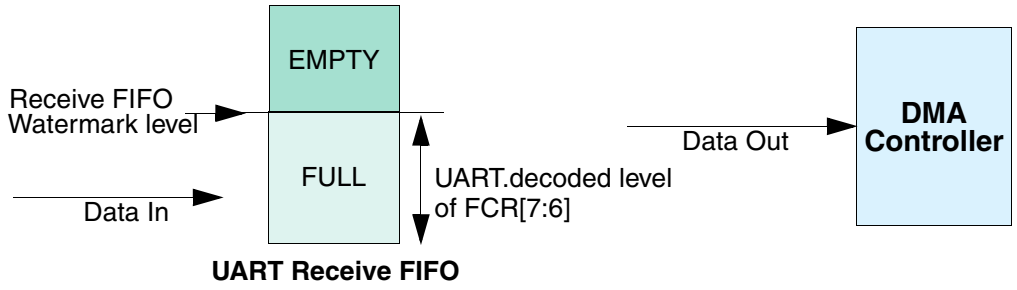
3.13.6 Choosing the Receive Watermark Level

Similar to choosing the transmit watermark level described earlier, the receive watermark level—decoded level of FCR[7:6]—should be set to minimize the probability of overflow. It is a trade-off between the number of DMA burst transactions required per block versus the probability of an overflow occurring.

3.13.7 Selecting SRC_MSIZE and Receive FIFO Underflow

As can be seen in Figure 3-31, programming a source burst transaction length greater than the watermark level can cause underflow when there is not enough data to service the source burst request. Therefore, equation (3) below must be adhered to in order to avoid underflow.

Figure 3-31 UART Receive FIFO



If the number of data items in the receive FIFO is equal to the source burst length at the time the burst request is made – DMA.CTL*x*.SRC_MSIZE – the receive FIFO can be emptied, but not underflowed, at the completion of the burst transaction. For optimal operation, DMA.CTL*x*.SRC_MSIZE should be set at the watermark level; that is:

DMA.CTL*x*.SRC_MSIZE = decoded level of FCR[7:6] **(3)**

Adhering to equation (3) reduces the number of DMA bursts in a block transfer, and this in turn can improve AMBA bus utilization.

Note The receive FIFO is not empty at the end of the source burst transaction if the UART has successfully received one data item or more on the UART serial receive line during the burst.

3.13.8 Handshaking Interface Operation

- **dma_tx_req_n, dma_rx_req_n** – The request signals for source and destination—dma_tx_req_n and dma_rx_req_n—are activated when their corresponding FIFOs reach the watermark levels.

  The DW_ahb_dmac uses edge detection of the dma_tx_req_n signal/dma_rx_req_n to identify a request on the channel. Upon reception of the dma_tx_ack_n/dma_rx_ack_n signal from the DW_ahb_dmac to indicate the burst transaction is complete, the DW_apb_uart de-asserts the burst request signals—dma_tx_req_n/dma_rx_req_n—until dma_tx_ack_n/dma_rx_ack_n is de-asserted by the DW_ahb_dmac.

  When the DW_apb_uart samples that dma_tx_ack_n/dma_rx_ack_n is de-asserted, it can re-assert the dma_tx_req_n/dma_rx_req_n of the request line if their corresponding FIFOs exceed their watermark levels—back-to-back burst transaction. If this is not the case, the DMA request lines remain de-asserted.

Figure 3-32 shows a timing diagram of a burst transaction where pclk = hclk.

Figure 3-32 Burst Transaction – pclk = hclk

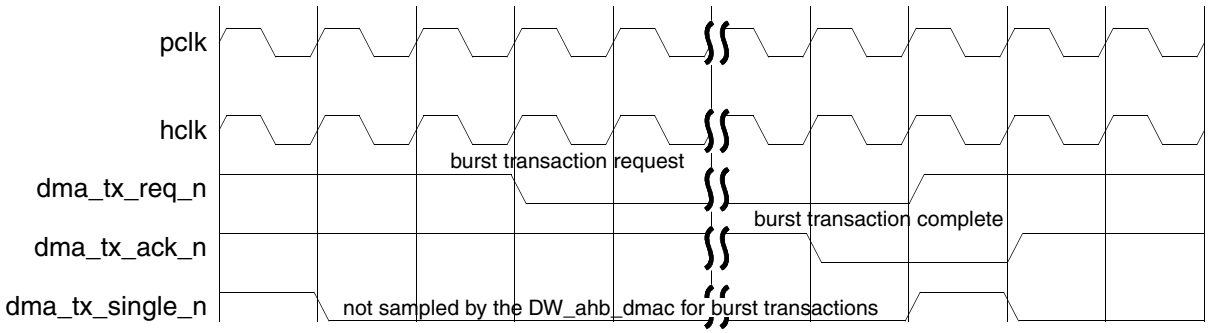
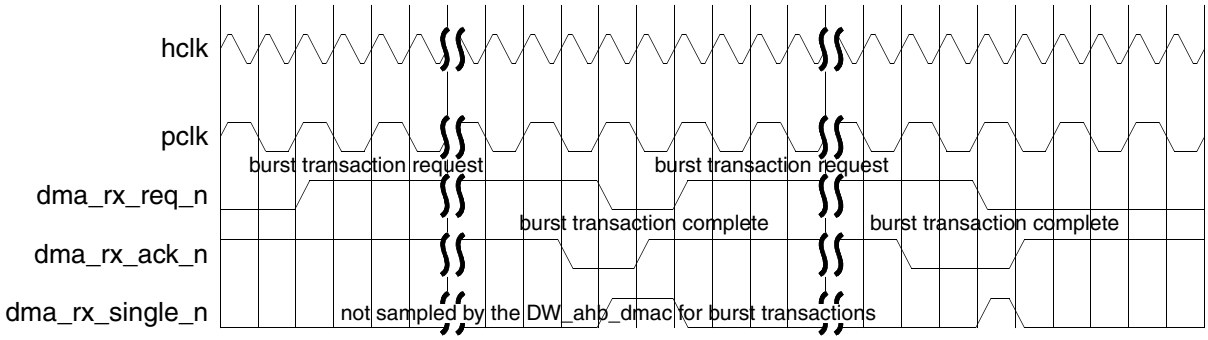


Figure 3-33 shows two back-to-back burst transactions where the hclk frequency is twice the pclk frequency.

Figure 3-33 Back-to-Back Burst Transactions – hclk = 2*pclk



The handshaking loop is as follows:

a. dma_tx_req_n/dma_rx_req_n asserted by DW_apb_uart
b. dma_tx_ack_n/dma_rx_ack_n asserted by DW_ahb_dmac
c. dma_tx_req_n/dma_rx_req_n de-asserted by DW_apb_uart
d. dma_tx_ack_n/dma_rx_ack_n de-asserted by DW_ahb_dmac
e. dma_tx_req_n/dma_rx_req_n re-asserted by DW_apb_uart, if back-to-back transaction is required

Note The burst transaction request signals, dma_tx_req_n and dma_rx_req_n, are generated in the DW_apb_uart off pclk and sampled in the DW_ahb_dmac by hclk. The acknowledge signals, dma_tx_ack_n and dma_rx_ack_n, are generated in the DW_ahb_dmac off hclk and sampled in the DW_apb_uart of pclk. The handshaking mechanism between the DW_ahb_dmac and the DW_apb_uart supports quasi-synchronous clocks; that is, hclk and pclk must be phase-aligned, and the hclk frequency must be a multiple of the pclk frequency.

- Note the following:
  - Once asserted, the burst request lines—dma_tx_req_n/dma_rx_req_n—remain asserted until their corresponding dma_tx_ack_n/dma_rx_ack_n signal is received, even if the respective FIFOs drop below their watermark levels during the burst transaction.
  - The dma_tx_req_n/dma_rx_req_n signals are de-asserted when their corresponding dma_tx_ack_n/dma_rx_ack_n signals are asserted, even if the respective FIFOs exceed their watermark levels.

- **dma_tx_single_n, dma_rx_single_n**
  - dma_tx_single_n – status signal that is asserted when there is at least one free entry in the transmit FIFO; it is cleared when the transmit FIFO is full.
  - dma_rx_single_n – status signal that is asserted when there is at least one valid data entry in the receive FIFO; it is cleared when the receive FIFO is empty.

These signals are needed by only the DW_ahb_dmac for the case where the block size—CTL*x*.BLOCK_TS—that is programmed into the DW_ahb_dmac is not a multiple of the burst transaction length—CTL*x*.SRC_MSIZE, CTL*x*.DEST_MSIZE—shown in Figure 3-28 on page 76. In this case, the DMA single outputs inform the DW_ahb_dmac that it is still possible to perform single data item transfers, so it can access all data items in the transmit/receive FIFO and complete the DMA block transfer. Otherwise, the DMA single outputs from the DW_apb_uart are not sampled by the DW_ahb_dmac.

This is illustrated in the following example.

Receive FIFO Channel of the DW_apb_uart:

DMA.CTLx.SRC_MSIZE = decoded level of UART.FCR[7:6] = 4

DMA.CTLx.BLOCK_TS = 12

Block transfer:

DMA.CTLx.SRC_MSIZE = decoded level of UART.FCR[7:6] = 4

DMA.CTLx.BLOCK_TS = 15

For the example in Figure 3-27 on page 75, with the block size set to 12, the dma_rx_req_n signal is asserted when four data items are present in the receive FIFO. The dma_rx_req_n signal is asserted three times during the DW_apb_uart serial transfer, ensuring that all 12 data items are read by the DW_ahb_dmac. All DMA requests read a block of data items and no single DMA transactions are required. The block transfer is made up of three burst transactions.

The first 12 data items are transferred using three burst transactions. But when the last three data frames enter the receive FIFO, the dma_rx_req_n signal is not activated because the FIFO level is below the watermark level. The DW_ahb_dmac samples dma_rx_single_n and completes the DMA block transfer using three single transactions. The block transfer is made up of three burst transactions, followed by three single transactions.

Figure 3-34 shows a single transaction. The handshaking loop is as follows:

a. dma_tx_single_n/dma_rx_single_n asserted by DW_apb_uart
b. dma_tx_ack_n/dma_rx_ack_n asserted by DW_ahb_dmac
c. dma_tx_single_n/dma_rx_single_n de-asserted by DW_apb_uart
d. dma_tx_ack_n/dma_rx_ack_n de-asserted by DW_ahb_dmac

Figure 3-34 Single Transaction

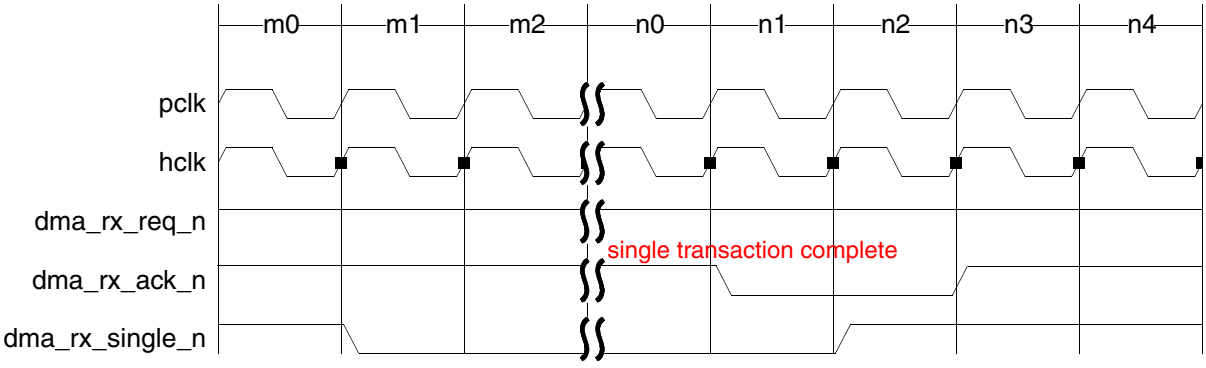
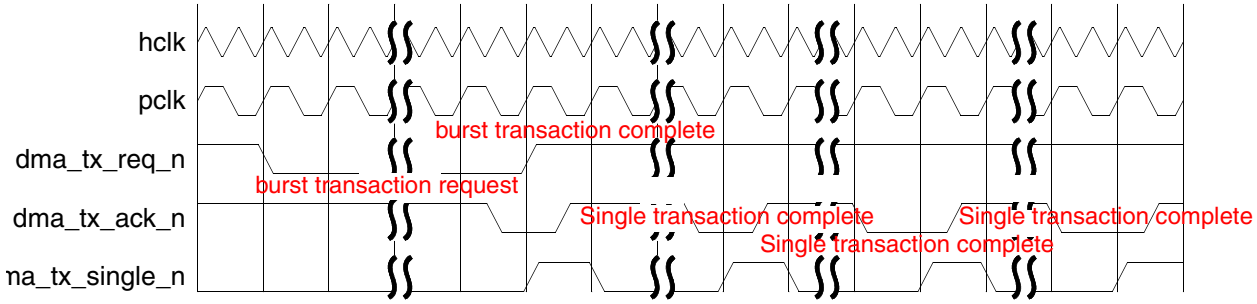


Figure 3-35 shows a burst transaction, followed by three back-to-back single transactions, where the hclk frequency is twice the pclk frequency.

Figure 3-35 Burst Transaction + 3 Back-to-Back Singles – hclk = 2*pclk



Note The single transaction request signals, dma_tx_single_n and dma_rx_single_n, are generated in the DW_apb_uart on the pclk edge and sampled in DW_ahb_dmac on hclk. The acknowledge signals, dma_tx_ack_n and dma_rx_ack_n, are generated in the DW_ahb_dmac on the hclk edge hclk and sampled in the DW_apb_uart on pclk. The handshaking mechanism between the DW_ahb_dmac and the DW_apb_uart supports quasi-synchronous clocks; that is, hclk and pclk must be phase aligned and the hclk frequency must be a multiple of pclk frequency.

3.13.9 Potential Deadlock Conditions in DW_apb_uart/DW_ahb_dmac Systems

There is a risk of a deadlock occurring if both of the following are true:

- DW_ahb_dmac is used to access UART FIFOs
- DMA burst transaction length is set to value smaller than or equal to DW_apb_uart Rx FIFO threshold
- When DW_apb_uart is used in DMA mode 1 with auto-flow control mode enabled

3.13.9.1 Deadlock When DMA Burst Transaction Length Smaller Than Rx FIFO Threshold

When operating in autoflow control mode with the RTC flow trigger threshold is disabled, the DW_apb_uart de-asserts rts_n when the Rx FIFO threshold is reached, and it asserts it again when the Rx FIFO is empty. At the same time, the DW_apb_uart asserts dma_rx_req_n, requesting a burst transaction from the DW_ahb_dmac.

If the DMA burst transaction length is equal to or greater than the Rx FIFO threshold, the DW_ahb_dmac reads from the Rx FIFO until it is empty, causing rts_n to be re-asserted. This in turn allows more data to be received by the DW_apb_uart and the Rx FIFO to fill again.

However, if the DW_ahb_dmac burst transaction length is smaller than the DW_apb_uart Rx FIFO threshold, some data is left in the DW_apb_uart Rx FIFO after completion of the burst transaction. This prevents the rts_n signal from being asserted.

Because the amount of data in the Rx FIFO is below the threshold, the DW_apb_uart asserts the dma_rx_single_n signal—instead of dma_rx_req_n—requesting a DMA single transaction from the

DW_ahb_dmac. However, unless it is operating in the single transaction region, the DW_ahb_dmac ignores single transaction requests.

A deadlock condition is then reached:

- DW_apb_uart does not receive any extra characters because the rts_n signal is de-asserted; no data can be pushed into the Rx FIFO to fill it up to the threshold level again and generate a new burst transaction request from the DW_ahb_dmac; only single transaction requests can be generated.
- Unless it has reached the single transaction region, the DW_ahb_dmac ignores single transaction requests and does not read from the Rx FIFO; the Rx FIFO cannot be emptied, which prevents the rts_n signal from being asserted again

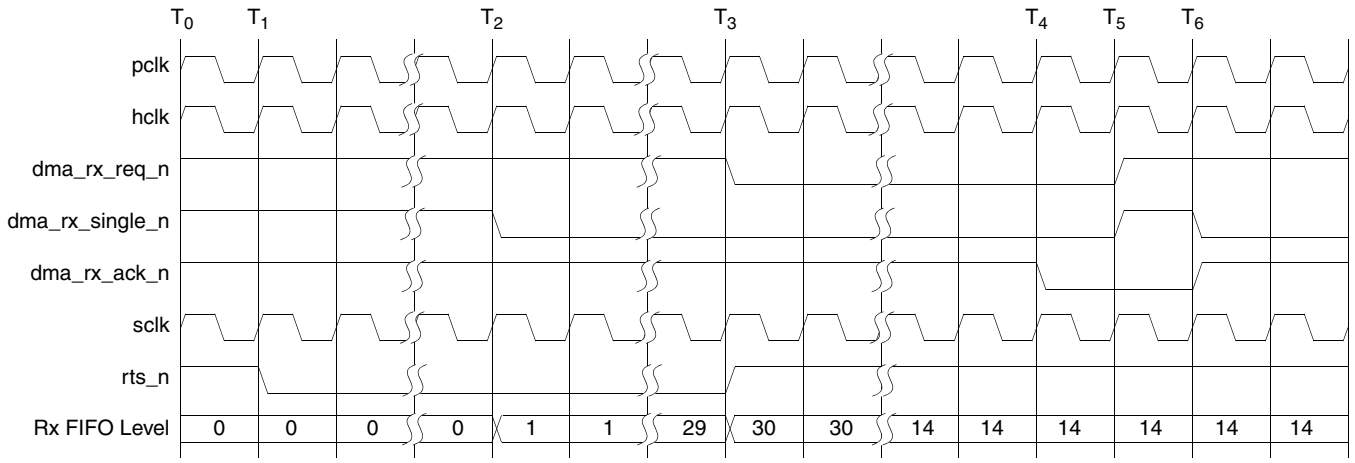
Table 3-2 illustrates this condition.

Table 3-2 DW_apb_uart/DW_ahb_dmac Settings for Deadlock When Transaction Less Than Rx FIFO Threshold

DW_apb_uart Settings	DW_ahb_dmac Settings
■ Component configured for 32 byte deep Rx FIFO ■ Autoflow mode enabled (AFCE=1) ■ Rx FIFO threshold set to “2 less than full”; that is, 30 bytes (RCVR=11)	■ Block size set to 100 bytes (BLOCK_TS=100) ■ Source transaction width set to 1 byte (SRC_TR_WIDTH=1) ■ Source burst transaction length set to 16 (SRC_MSIZE=16)

The timing diagram in Figure 3-36 illustrates the sequence of events that lead to this deadlock condition.

Figure 3-36 Example of DW_apb_uart and DW_ahb_dmac Deadlock Occurrence



Note For the sake of simplicity, pclk, hclk and sclk are shown to be identical; however, this is not a constraint for the occurrence of deadlock. Additionally, in the interest of simplicity, some events are represented as taking place simultaneously; however, in reality this might not be strictly the case and these events can be separated by a small number of clock cycles.

In Figure 3-36, the following events are shown:

- T1 – The DW_apb_uart is programmed and enabled; rts_n is asserted to initiate the reception of characters.
- T2 – The first character is received by the DW_apb_uart and pushed into the RxFIFO; dma_rx_single_n is asserted as a consequence, but because DW_ahb_dmac is not in the single transfer region, this request is ignored.
- T3 – The 30th character is received and pushed into the Rx FIFO. As a consequence:
  - rts_n is de-asserted, stopping any further characters from being received
  - dma_rx_req_n signal is asserted
  - DW_ahb_dmac attends this request and starts reading data from Rx FIFO
- T4 – The 16th character popped from the Rx FIFO is received by the DW_ahb_dma, which asserts dma_rx_ack_n to signal the completion of the DMA burst transaction. Since the DMA burst transaction size is set to 16 and the Rx FIFO threshold is set to 30, there are fourteen characters left in the Rx FIFO after the DMA burst transaction completes.
- T5 – One cycle after dma_rx_ack_n is asserted, the DW_apb_uart de-asserts dma_rx_req_n and dma_rx_single_n as part of the DMA handshaking protocol.
- T6 – One cycle after dma_rx_req_n is de-asserted, the DW_ahb_dmac de-asserts dma_rx_ack_n to complete the DMA handshaking protocol. At the same time, the DW_apb_uart re-asserts dma_req_single_n because there are fourteen characters in the Rx FIFO. For the same reason, rts_n is kept de-asserted; since the DW_ahb_dmac is not in the single transfer region, it ignores the single transaction request and a deadlock is created.

This deadlock condition can be avoided if:

- The Rx FIFO threshold level is set to a value equal to or smaller than the DMA burst transaction size. This ensures the Rx FIFO is always empty after a DMA burst transaction completes and rts_n is asserted accordingly.
- The DMA block size is set to a value smaller than twice the DMA burst transaction length. This guarantees the DW_ahb_dmac enters the single transaction region after the DMA burst transaction completes. It then accepts single transaction requests from the DW_apb_uart, allowing the Rx FIFO to be emptied. In this case, the DMA burst size can be configured to be smaller than the Rx FIFO threshold level.

3.13.9.2 Deadlock When DMA Burst Transaction Length Equal To Rx FIFO Threshold

If the DMA burst transaction length is identical to the DW_apb_uart Rx FIFO threshold, there is risk of a deadlock condition occurring when a character is received after rts_n is de-asserted.

The DW_apb_uart de-asserts rts_n when the Rx FIFO threshold is reached. However, it is possible the component at the other end of the line starts transmitting a new character before it detects the de-assertion of its cts_n input. When this happens, the character transmission completes normally, which means an extra character will be received and pushed into the Rx FIFO (unless it is already full).

At the same time that rts_n is de-asserted, the DW_apb_uart asserts dma_rx_req, requesting a DMA burst transaction from the DW_ahb_dmac. After the DW_ahb_dmac completes this burst transaction—with

length equal to the Rx FIFO threshold—there is one character left in the Rx FIFO, preventing rts_n from being asserted again.

The DW_apb_uart asserts the dma_rx_single_n signal—instead of dma_rx_req_n—requesting a DMA single transaction from the DW_ahb_dmac. However, unless it is operating in the single-transaction region, the DW_ahb_dmac ignores single-transaction requests.

A deadlock condition is then reached:

- The DW_apb_uart does not receive any extra characters because the rts_n signal is de-asserted. No data can be pushed into the Rx FIFO to fill it up to the threshold level again and generate a new burst transaction request from the DW_ahb_dma; only single-transaction requests can be generated.
- Unless it has reached the single-transaction region, the DW_ahb_dmac ignores single-transaction requests and does not read from the Rx FIFO. The Rx FIFO cannot be emptied, which prevents the rts_n signal from being asserted again.

This deadlock condition can be avoided if:

- The Rx FIFO threshold level is set to a value smaller than the DMA burst transaction size. This ensures that the Rx FIFO is always empty after a DMA burst transaction completes, regardless of whether or not one extra character is received and rts_n is asserted accordingly.
- The DMA block size is set to a value smaller than twice the DMA burst transaction length. This guarantees that the DW_ahb_dmac enters the single transaction region after the DMA burst transaction completes. It then accepts single transaction requests from the DW_apb_uart, allowing the Rx FIFO to be emptied.

This deadlock condition is not expected to occur frequently under normal operating conditions. A timeout interrupt would be generated in this case, which can be used to detect the occurrence of this deadlock condition.

3.14 Reset Signals

When configured for asynchronous serial clock operation, the DW_apb_uart includes two separate reset signals, each dedicated to its own clock domain:

- presetn resets logic in pclk clock domain
- s_rst_n resets logic in sclk clock domain

In order to avoid serious operational failures, both clock domains of the DW_apb_uart must be reset before any attempt is made to send or receive data on the serial line; that is, it is an illegal operation to reset just one clock domain of the DW_apb_uart without resetting the other clock domain.

Each reset signal must be de-asserted synchronously with the corresponding clock signal.

When asserting the reset signals, the s_rst_n signal should be asserted before or at the same time as presetn; this prevents any unexpected activity on the serial line that might result from resetting the programming registers without resetting the serial logic.

Similarly, when de-asserting the reset signals, s_rst_n should be de-asserted before presetn is de-asserted. The safest procedure for resetting DW_apb_uart is as follows:

1. Assert s_rst_n and presetn; the sequence of asserting these two signals and their timing relationship with sclk and pclk are not important
2. De-assert s_rst_n synchronously with sclk
3. De-assert presetn synchronously with pclk

Both reset signals should be active for at least three cycles of the respective clock signal.

4 Parameters

This chapter describes the configuration parameters used by the DW_apb_uart. You use coreConsultant or coreAssembler to configure the following parameters and generate the configured code.

Attention	When using coreConsultant or coreAssembler, you can right-click on a parameter label to access a “What’s This” popup dialog that will tell you the details for that particular parameter. The information in each What’s This dialog essentially matches the information in the parameter descriptions below.

4.1 Parameter Descriptions

Table 4-1 lists the DW_apb_uart parameter descriptions.

Table 4-1 Top-Level Parameters

Label	Parameter Definition
APB Data Bus Width	**Parameter Name:** APB_DATA_WIDTH **Legal Values:** 8, 16, 32 **Default Value:** 32 **Dependencies:** None **Description:** Width of APB data bus to which this component is attached. The data width can be set to 8, 16, or 32. Register access is on 32-bit boundaries, and unused bits are held at static 0.
UART FIFO Depth	**Parameter Name:** FIFO_MODE **Legal Values:** NONE, 16, 32, …, 2 KB (2048) **Default Value:** 16 **Dependencies:** None **Description:** Receiver and Transmitter FIFO depth in bytes. A setting of NONE means no FIFOs, which implies the 16450-compatible mode of operation. Most enhanced features are unavailable in the 16450 mode such as the Auto Flow Control and Programmable THRE interrupt modes. Setting a FIFO depth greater than 256 restricts the FIFO Memory to External only. For more details, refer to “FIFO Support” on page 57.

Table 4-1 Top-Level Parameters (Continued)

Label	Parameter Definition
FIFO Memory Type	**Parameter Name:** MEM_SELECT_USER **Legal Values:** ■ 0 – External - User-supplied memory ■ 1 – Internal - DesignWare memory instantiation **Default Value:** External (0) **Dependencies:** Only changeable to Internal if (FIFO_MODE != NONE) and (FIFO_MODE <= 256) **Description:** Selects between external, user-supplied memory or internal DesignWare memory (DW_ram_r_w_s_dff) for the receiver and transmitter FIFOs. FIFO depths greater than 256 restrict FIFO Memory selection to external. In addition, selection of internal memory restricts the Memory Read Port Type to D-flip-flop-based, synchronous read port RAMs.
Asynchronous Serial Clock Support	**Parameter Name:** CLOCK_MODE **Legal Values:** ■ 1 – Disabled - One clock ■ 2 – Enabled - Two clocks **Default Value:** Disabled (1) **Dependencies:** Asynchronous Serial Clock Support is automatically enabled when SIR_LP_MODE = Enabled or when SIR_LP_RX = Enabled. **Description:** When set to Disabled, the DW_apb_uart is implemented with one system clock (pclk). When set to Enabled, two system clocks (pclk and sclk) are implemented in order to accommodate accurate serial baud rate settings, as well as APB bus interface requirements. Selecting Disabled, or a one-system clock, greatly restricts system clock settings available for accurate baud rates. For more details, refer to “Clock Support” on page 59.
Auto Flow Control	**Parameter Name:** AFCE_MODE **Legal Values:** ■ 0 – Disabled - Auto Flow Control not available ■ 1 – Enabled - Auto Flow Control **Default Value:** Disabled (0) Dependencies: Changeable to Enabled only when (FIFO_MODE != NONE) **Description:** Configures the peripheral to have the 16750-compatible auto flow control mode. For more details, refer to “Auto Flow Control” on page 64.

Table 4-1 Top-Level Parameters (Continued)

Label	Parameter Definition
RTC Flow Control Trigger	**Parameter Name:** RTC_FCT **Legal Values:** ■ 0 – RX FIFO Threshold Trigger ■ 1 – RX FIFO Almost-Full Trigger, where “almost full” refers to two available slots in the FIFO **Default Value:** RX FIFO Threshold Trigger (0) **Dependencies:** Changeable only when AFCE_MODE is enabled **Description:** When set to 0, the DW_apb_uart uses the same receiver trigger level—described in FCR.RCVR register—both for generating a DMA request and a handshake signal (rts_n). When set to 1, the DW_apb_uart uses two separate trigger levels for a DMA request and handshake signal (rts_n) in order to maximize throughput on the interface.
Programmable THRE Interrupt Mode	**Parameter Name:** THRE_MODE_USER **Legal Values:** ■ 0 – Disabled - THRE Interrupt mode not available ■ 1 – Enabled - THRE Interrupt mode **Default Value:** Disabled (0) **Dependencies:** Changeable to Enabled only when (FIFO_MODE != NONE) **Description:** Configures the peripheral to have a programmable Transmitter Hold Register Empty (THRE) Interrupt mode. For more information, refer to “Programmable THRE Interrupt” on page 68.
IrDA SIR Mode Support	**Parameter Name:** SIR_MODE **Legal Values:** ■ 0 – Disabled - IrDA SIR mode not available ■ 1 – Enabled - IrDA SIR mode **Default Value:** Disabled (0) **Dependencies:** None **Description:** Configures the peripheral to have IrDA 1.0 SIR infrared mode. For more details, refer to “IrDA 1.0 SIR Protocol” on page 55.
Include Clock Gate Enable Output on I/F?	**Parameter Name:** CLK_GATE_EN **Legal Values:** ■ 0 – No - Clock gate enable is not available ■ 1 – Yes - Clock gate enable **Default Value:** No (0) **Dependencies:** None **Description:** Configures the peripheral to have a clock gate enable output signal on the interface that indicates that the device is inactive so clocks may be gated.

Table 4-1 Top-Level Parameters (Continued)

Label	Parameter Definition
Include FIFO Access Mode?	**Parameter Name:** FIFO_ACCESS **Legal Values:** ■ 0 – No - FIFO access mode is not available ■ 1 – Yes - FIFO access mode **Default Value:** No (0) **Dependencies:** None **Description:** Configures the peripheral to have a programmable FIFO access mode. This is used for test purposes to allow the receive FIFO to be written and the transmit FIFO to be read when FIFOs are implemented and enabled. When FIFOs are not implemented or not enabled it allows the RBR to be written and the THR to be read. For more details, refer to "FIFO Support" on page 57.
Include Additional DMA Signals on I/F?	**Parameter Name:** DMA_EXTRA **Legal Values:** ■ 0 – No - Additional DMA signals not included ■ 1 – Yes - Additional DMA signals included **Default Value:** No (0) **Dependencies:** None **Description:** Configures the peripheral to have four additional DMA signals on the interface so that the device is compatible with the DesignWare DMA controller interface requirements.
Active low DMA Signals?	**Parameter Name:** DMA_POL **Legal Values:** ■ 0 – No - DMA signals set to active-high ■ 1 – Yes - DMA signals set to active-low **Default Value:** Yes (1) **Dependencies:** None **Description:** Selects the polarity of the DMA interface signals.
Low Power IrDA SIR Mode Support	**Parameter Name:** SIR_LP_MODE **Legal Values:** ■ 0 – Disabled - Low-power IrDA SIR mode not available ■ 1 – Enabled - Low-power IrDA SIR mode **Default Value:** Disabled (0) **Dependencies:** This is only changeable when SIR_MODE = Enabled. **Description:** Configures the peripheral to operate in a low-power IrDA SIR mode. As the DW_apb_uart does not support a low-power mode with a counter system to maintain a 1.63us infrared pulse, Asynchronous Serial Clock Support is automatically enabled, and the sclk must be fixed to 1.8432MHz. This provides a 1.63us sir_out_n pulse at 115.2Kbaud.

Table 4-1 Top-Level Parameters (Continued)

Label	Parameter Definition
Support for IrDA SIR Low-Power Reception Capabilities	**Parameter Name:** SIR_LP_RX **Legal Values:** ■ 0 – Disabled ■ 1 – Enabled **Default Value:** Disabled (0) **Dependencies:** This is only changeable when SIR_MODE = Enabled. **Description:** Configures the peripheral to have SIR low-power reception capabilities. Asynchronous Serial Clock support is automatically enabled in this mode.
Include On-chip Debug Output Signals on I/F?	**Parameter Name:** DEBUG **Legal Values:** ■ 0 – No - On-chip debug signals not included ■ 1 – Yes - On-chip debug signals included **Default Value:** No (0) **Dependencies:** None **Description:** Configures the peripheral to have on-chip debug signals on the interface.
Include Baud Clock Reference Output Signal (baudout_n) on I/F?	**Parameter Name:** BAUD_CLK **Legal Values:** ■ 0 – No - baudout_n signal not included ■ 1 – Yes - baudout_n signal included **Default Value:** Yes (1) **Dependencies:** None **Description:** Configures the peripheral to have a baud clock reference output (baudout_n) signal on the interface.
Add Version and ID Registers, Enable FIFO Status, Shadow and Encoded Parameters Register Options?	**Parameter Name:** ADDITIONAL_FEATURES Legal Values: ■ 0 – No - Version and ID Registers; additional register options not included ■ 1 – Yes - Version and ID Registers; additional register options included **Default Value:** No (0) **Dependencies:** None **Description:** Configures the peripheral to have the option to include the FIFO status registers, shadow registers, and encoded parameter register. Also configures the peripheral to have the UART component version and the peripheral ID registers.

Table 4-1 Top-Level Parameters (Continued)

Label	Parameter Definition
Include Software Accessible FIFO Status Registers?	**Parameter Name:** FIFO_STAT **Legal Values:** ■ 0 – No - FIFO Status registers not included ■ 1 – Yes - FIFO Status registers included **Default Value:** No (0) **Dependencies:** This is only changeable when: ■ FIFO_MODE != NONE and ■ ADDITIONAL_FEATURES = YES. **Description:** Configures the peripheral to have three additional FIFO status registers.
Include Additional Shadow Registers for Reducing Software Overhead?	**Parameter Name:** SHADOW **Legal Values:** ■ 0 – No - Additional registers not included ■ 1 – Yes - Additional Shadow registers included **Default Value:** No (0) **Dependencies:** This is only changeable when ADDITIONAL_FEATURES = YES. **Description:** Configures the peripheral to have nine additional registers that shadow some of the existing register bits that are regularly modified by software. These can be used to reduce the software overhead that is introduced by having to perform read-modify writes.
Include Component Parameter Register?	**Parameter Name:** UART_ADD_ENCODED_PARAMS **Legal Values:** ■ 0 – No - component parameter register (CPR) not included ■ 1 – Yes - component parameter register (CPR) included **Default Value:** No (0) **Dependencies:** This is only changeable when ADDITIONAL_FEATURES = YES. **Description:** Configures the peripheral to have a component parameter register (CPR).
Remove Busy Functionality?	**Parameter Name:** UART_16550_COMPATIBLE **Legal Values:** ■ 0 – No - not 16550-compatible ■ 1 – Yes - 16550-compatible **Default Value:** No (0) **Dependencies**: None **Description:** Configures the peripheral to be fully 16550-compatible. This is achieved by not having the busy functionality implemented.

Table 4-1 Top-Level Parameters (Continued)

Label	Parameter Definition
Enable 9-bit Mode Support?	**Parameter Name**: UART_9BIT_DATA_EN **Legal Values**: ■ 0 - Disabled - 9-bit data is not included ■ 1 - Enabled - 9-bit data support is included **Default Value**: Disabled (0) **Dependencies**: DWC-APB-Advanced-Source license required. **Description**: Configures the peripheral to have 9-bits of data per character. The 9^{th} bit of the data byte sent from the master is set to 1 to indicate the address byte while cleared to 0 to indicate the data byte.
RS485 Interface Support	**Parameter Name**: UART_RS485_INTERFACE_EN **Legal Values**: ■ 0 - Disabled - RS485 Interface not included ■ 1 - Enabled - RS485 Interface included **Default Value**: Disabled (0) **Dependencies**: DWC-APB-Advanced-Source license required. **Description**: Configures the peripheral for RS485 Interface support. If enabled, new signals 'de', 're' and 'rs485_en' are included in the interface to support RS485 Transceiver.
Active High RS485 Driver Enable Signal?	**Parameter Name**: UART_DE_POL **Legal Values**: 0 - ActiveLow - 'de' signal set to active-low. 1 - ActiveHigh - 'de' signal set to active-high. **Default Value**: ActiveHigh (1) **Dependencies**: Enabled only when UART_RS485_INTERFACE_EN=1. **Description**: Selects the polarity of the Driver Enable (de) signal.
Active High RS485 Receiver Enable Signal?	**Parameter Name**: UART_RE_POL **Legal Values**: 0 - ActiveLow - 're' signal set to active-low. 1 - ActiveHigh - 're' signal set to active-high. **Default Value**: ActiveHigh (1) **Dependencies**: Enabled only when UART_RS485_INTERFACE_EN=1. **Description**: Selects the polarity of the Receiver Enable (re) signal.

Table 4-1 Top-Level Parameters (Continued)

Label	Parameter Definition
Fractional Baud Rate Divisor Support	**Parameter Name**: FRACTIONAL_BAUD_DIVISOR_EN **Legal Values**: 0 - Disabled - Fractional Baud Rate is not enabled. 1 - Enabled - Fractional Baud Rate is enabled. **Default Value**: Disabled(0) **Dependencies**: DWC-APB-Advanced-Source license required. **Description**: Configures the peripheral to have Fractional Baud Rate Divisor. If enabled, new Fractional divisor latch register (DLF) is included to program the fractional divisor values. For more information about this feature, see “Fractional Baud Rate Support” on page 52.
Fractional Divisor Width	**Parameter Name**: DLF_SIZE **Legal Values**: 4, 5, and 6 **Default Value**: 4 **Dependencies**: FRACTIONAL_BAUD_DIVISOR_EN = 1 **Description**: Specifies the width of the fractional divisor. A high value means more precision but long averaging period. For more information about this feature, see “Fractional Baud Rate Support” on page 52.

5 Signal Descriptions

This chapter details all possible I/O signals in the core. For configurable IP titles, your actual configuration might not contain all of these signals.

Inputs are on the left of the signal diagrams; outputs are on the right.

Attention: For configurable IP titles, do not use this document to determine the exact I/O footprint of the core. It is for reference purposes only.

When you configure the core in coreConsultant, you must access the I/O signals for your actual configuration at workspace/report/IO.html or workspace/report/IO.xml after you have completed the report creation activity. That report comes from the exact same source as this chapter but removes all the I/O signals that are not in your actual configuration. This does not apply to non-configurable IP titles. In addition, all parameter expressions are evaluated to actual values. Therefore, the widths might change depending on your actual configuration.

Some expressions might refer to TCL functions or procedures (sometimes identified as **<functionof>)** that coreConsultant uses to make calculations. The exact formula used by these TCL functions is not provided in this chapter. However, when you configure the core in coreConsultant, all TCL functions and parameters are evaluated completely; and the resulting values are displayed where appropriate in the coreConsultant GUI reports.

The I/O signals are grouped as follows:

5.1 APB Slave Interface Signals

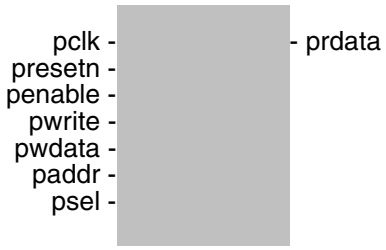


Table 5-1 APB Slave Interface Signals

Port Name	I/O	Description
pclk	I	APB clock used in the APB interface to program registers **Exists**: Always **Active State**: N/A **Synchronous to**: N/A **Registered**: N/A
presetn	I	APB clock-domain reset **Exists**: Always **Active State**: Low **Synchronous to**: pclk on de-assertion, asynchronous on assertion **Registered**: N/A
penable	I	APB enable control used for timing read/write operations **Exists**: Always **Active State**: High **Synchronous to**: pclk **Registered**: No
pwrite	I	APB Write control **Exists**: Always **Active State**: High **Synchronous to**: pclk **Registered**: No
pwdata[(APB_DATA_WIDTH-1):0]	I	APB write data bus **Exists**: Always **Active State**: High **Synchronous to**: pclk **Registered**: No

Table 5-1 APB Slave Interface Signals (Continued)

Port Name	I/O	Description
paddr[(UART_ADDR_SLICE_LHS-1):0]	I	APB address bus. Uses the lower bits of the APB address bus for register decode **Exists**: Always **Active State**: High **Synchronous to**: pclk **Registered**: No
psel	I	APB peripheral select **Exists**: Always **Active State**: High **Synchronous to**: pclk **Registered**: No
prdata[(APB_DATA_WIDTH-1):0]	O	APB read data bus. **Exists**: Always **Active State**: High **Synchronous to**: pclk **Registered**: Yes

5.2 Application Interface Signals

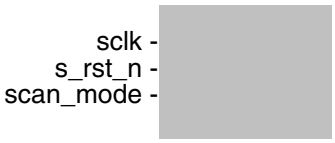


Table 5-2 Application Interface Signals

Port Name	I/O	Description
sclk	I	Serial Interface Clock **Exists**: CLOCK_MODE==2 **Active State**: N/A **Synchronous to**: N/A **Registered**: N/A
s_rst_n	I	Serial Interface Reset **Exists**: CLOCK_MODE==2 **Active State**: Low **Synchronous to**: N/A **Registered**: N/A
scan_mode	I	Scan mode used to ensure that test automation tools can control all asynchronous flop signals. During scan this signal must be set high all the time. In normal operation you must tie this signal low. **Exists**: Always **Active State**: High **Synchronous to**: N/A **Registered**: No

5.3 FIFO Interface Signals

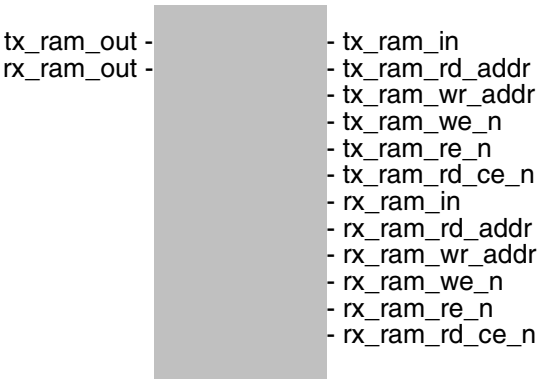


Table 5-3 FIFO Interface Signals

Port Name	I/O	Description
tx_ram_out[(TX_RAM_DATA_WIDTH-1):0]	I	Data to the transmit FIFO RAM **Exists**: FIFO_MODE!=0 && MEM_SELECT==0 **Active State**: High **Synchronous to**: pclk **Registered**: No
rx_ram_out[(RX_RAM_DATA_WIDTH-1):0]	I	Data to the receive FIFO RAM **Exists**: FIFO_MODE!=0 && MEM_SELECT==0 **Active State**: High **Synchronous to**: pclk **Registered**: No
tx_ram_in[(TX_RAM_DATA_WIDTH-1):0]	O	Data from the transmit FIFO RAM **Exists**: FIFO_MODE!=0 && MEM_SELECT==0 **Active State**: High **Synchronous to**: pclk **Registered**: No
tx_ram_rd_addr[(FIFO_ADDR_WIDTH-1):0]	O	Read address pointer for the transmit FIFO RAM **Exists**: FIFO_MODE!=0 && MEM_SELECT==0 **Active State**: High **Synchronous to**: pclk **Registered**: Yes

Table 5-3 FIFO Interface Signals (Continued)

Port Name	I/O	Description
tx_ram_wr_addr[(FIFO_ADDR_WIDTH-1):0]	O	Write address pointer for the transmit FIFO RAM **Exists**: FIFO_MODE!=0 && MEM_SELECT==0 **Active State**: High **Synchronous to**: pclk **Registered**: Yes
tx_ram_we_n	O	Write enable for the transmit FIFO RAM **Exists**: FIFO_MODE!=0 && MEM_SELECT==0 **Active State**: Low **Synchronous to**: pclk **Registered**: No
tx_ram_re_n	O	Read enable for the transmit FIFO RAM wake-up **Exists**: FIFO_MODE!=0 && MEM_SELECT==0 **Active State**: Low **Synchronous to**: pclk **Registered**: No
tx_ram_rd_ce_n	O	Read port chip enable for the transmit FIFO RAM **Exists**: FIFO_MODE!=0 && MEM_SELECT==0 **Active State**: Low **Synchronous to**: pclk **Registered**: Yes
rx_ram_in[(RX_RAM_DATA_WIDTH-1):0]	O	Data from the receive FIFO RAM **Exists**: FIFO_MODE!=0 && MEM_SELECT==0 **Active State**: High **Synchronous to**: pclk **Registered**: No
rx_ram_rd_addr[(FIFO_ADDR_WIDTH-1):0]	O	Read address pointer for the receive FIFO RAM **Exists**: FIFO_MODE!=0 && MEM_SELECT==0 **Active State**: High **Synchronous to**: pclk **Registered**: Yes
rx_ram_wr_addr[(FIFO_ADDR_WIDTH-1):0]	O	Write address pointer for the receive FIFO RAM **Exists**: FIFO_MODE!=0 && MEM_SELECT==0 **Active State**: High **Synchronous to**: pclk **Registered**: Yes

Table 5-3 FIFO Interface Signals (Continued)

Port Name	I/O	Description
rx_ram_we_n	O	Write enable for the receive FIFO RAM **Exists**: FIFO_MODE!=0 && MEM_SELECT==0 **Active State**: Low **Synchronous to**: pclk **Registered**: No
rx_ram_re_n	O	Read enable for the receive FIFO RAM wake-up **Exists**: FIFO_MODE!=0 && MEM_SELECT==0 **Active State**: Low **Synchronous to**: pclk **Registered**: No
rx_ram_rd_ce_n	O	Read port chip enable for receive FIFO RAM **Exists**: FIFO_MODE!=0 && MEM_SELECT==0 **Active State**: Low **Synchronous to**: pclk **Registered**: Yes

5.4 Modem Interface Signals

Table 5-4 Modem Interface Signals

Port Name	I/O	Description
cts_n	I	Clear To Send Modem Status **Exists**: Always **Active State**: Low **Synchronous to**: N/A **Registered**: No
dsr_n	I	Data Set Ready Modem Status input **Exists**: Always **Active State**: Low **Synchronous to**: N/A **Registered**: No
dcd_n	I	Data Carrier Detect Modedm Status input **Exists**: Always **Active State**: Low **Synchronous to**: N/A **Registered**: No
ri_n	I	Ring Indicator Status input **Exists**: Always **Active State**: Low **Synchronous to**: N/A **Registered**: No
dtr_n	O	Modem Control Data Terminal Ready Output **Exists**: Always **Active State**: Low **Synchronous to**: pclk **Registered**: No
rts_n	O	Modem Control Request To Send output **Exists**: Always **Active State**: Low **Synchronous to**: pclk **Registered**: No

Table 5-4 Modem Interface Signals (Continued)

Port Name	I/O	Description
out2_n	O	Modem Control Programmable output 2 **Exists**: Always **Active State**: Low **Synchronous to**: pclk **Registered**: No
out1_n	O	Modem Control Programmable output 1 **Exists**: Always **Active State**: Low **Synchronous to**: pclk **Registered**: No

5.5 DMA Interface Signals

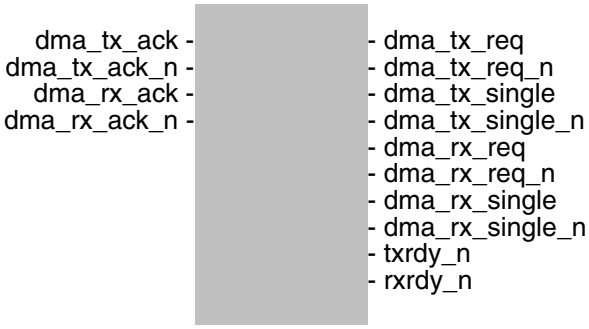


Table 5-5 DMA Interface Signals

Port Name	I/O	Description
dma_tx_ack	I	DMA Transmit Acknowledge (Active High) indicates that the DMA Controller has transmitted the block of data to the DW_apb_uart for transmission. **Exists**: DMA_EXTRA==1 && DMA_POL==0 **Active State**: High **Synchronous to**: pclk **Registered**: No
dma_tx_ack_n	I	DMA Transmit Acknowledge (Active Low)indicates that the DMA Controller has transmitted the block of data to the DW_apb_uart for transmission. **Exists**: DMA_EXTRA==1 && DMA_POL==1 **Active State**: Low **Synchronous to**: pclk **Registered**: No
dma_rx_ack	I	DMA Receive Acknowledge (Active High) indicates that the DMA Controller has transmitted the block of data from the DW_apb_uart. **Exists**: DMA_EXTRA==1 && DMA_POL==0 **Active State**: High **Synchronous to**: pclk **Registered**: No
dma_rx_ack_n	I	DMA Receive Acknowledge i(Active Low) ndicates that the DMA Controller has transmitted the block of data from the DW_apb_uart. **Exists**: DMA_EXTRA==1 && DMA_POL==1 **Active State**: Low **Synchronous to**: pclk **Registered**: No

Table 5-5 DMA Interface Signals (Continued)

Port Name	I/O	Description
dma_tx_req	O	Transmit Buffer Ready (Active High) indicates that the Transmit buffer requires service from the DMA controller. **Exists**: DMA_POL==0 **Active State**: High **Synchronous to**: pclk **Registered**: Yes
dma_tx_req_n	O	Transmit Buffer Ready (Active Low) indicates that the Transmit buffer requires service from the DMA controller. **Exists**: DMA_POL==1 **Active State**: Low **Synchronous to**: pclk **Registered**: Yes
dma_tx_single	O	DMA Transmit FIFO Single (Active High) informs the DMA Controller that there is at least one free entry in the Transmit buffer/FIFO. This output does not request a DMA transfer. **Exists**: DMA_EXTRA==1 && DMA_POL==0 **Active State**: High **Synchronous to**: pclk **Registered**: Yes
dma_tx_single_n	O	DMA Transmit FIFO Single (Active Low) informs the DMA Controller that there is at least one free entry in the Transmit buffer/FIFO. This output does not request a DMA transfer. **Exists**: DMA_EXTRA==1 && DMA_POL==1 **Active State**: Low **Synchronous to**: pclk **Registered**: Yes
dma_rx_req	O	Receive Buffer Ready (Active High) indicates that the Receive buffer requires service from the DMA controller. **Exists**: DMA_POL==0 **Active State**: High **Synchronous to**: pclk **Registered**: Yes
dma_rx_req_n	O	Receive Buffer Ready (Active Low) indicates that the Receive buffer requires service from the DMA controller. **Exists**: DMA_POL==1 **Active State**: Low **Synchronous to**: pclk **Registered**: Yes

Table 5-5 DMA Interface Signals (Continued)

Port Name	I/O	Description
dma_rx_single	O	DMA Receive FIFO Single (Active High) informs the DMA controller that there is at least one free entry in the Receive buffer/FIFO. This output does not request a DMA transfer. **Exists**: DMA_EXTRA==1 && DMA_POL==0 **Active State**: High **Synchronous to**: pclk **Registered**: Yes
dma_rx_single_n	O	DMA Receive FIFO Single (Active Low) informs the DMA controller that there is at least one free entry in the Receive buffer/FIFO. This output does not request a DMA transfer. **Exists**: DMA_EXTRA==1 && DMA_POL==1 **Active State**: Low **Synchronous to**: pclk **Registered**: Yes
txrdy_n	O	This transmit buffer read signal is used for backward compatibility of older DW_apb_uart components to indicate that the Transmit buffer requires service from the DMA controller. **Exists**: DMA_EXTRA==0 **Active State**: Low **Synchronous to**: pclk **Registered**: Yes
rxrdy_n	O	This receive buffer read signal is used for backward compatibility of older DW_apb_uart components to indicate that the Receive buffer requires service from the DMA controller. **Exists**: DMA_EXTRA==0 **Active State**: Low **Synchronous to**: pclk **Registered**: Yes

5.6 Serial Interface Signals

Table 5-6 Serial Interface Signals

Port Name	I/O	Description
sin	I	Serial Input. **Exists**: Always **Active State**: High **Synchronous to**: pclk (in single clock configuration),sclk (in two clock configuration) **Registered**: No
sout	O	Serial Output **Exists**: Always **Active State**: High **Synchronous to**: pclk (in single clock configuration),sclk (in two clock configuration) **Registered**: Yes

5.7 Infrared Interface Signals

Table 5-7 Infrared Interface Signals

Port Name	I/O	Description
sir_in	I	IrDA SIR Input. **Exists**: SIR_MODE==1 **Active State**: High **Synchronous to**: pclk (in single clock configuration),sclk (in two clock configuration) **Registered**: No
sir_out_n	O	IrDA SIR Output **Exists**: SIR_MODE==1 **Active State**: Low **Synchronous to**: pclk (in single clock configuration),sclk (in two clock configuration) **Registered**: Yes

5.8 Clock Control Interface Signals

Table 5-8 Clock Control Interface Signals

Port Name	I/O	Description
uart_lp_req_pclk	O	pclk domain clock gate signal indicates that the UART is inactive, so clocks may be gated to put the device in a low-power(lp) mode. **Exists**: CLK_GATE_EN==1 **Active State**: High **Synchronous to**: pclk **Registered**: Yes
uart_lp_req_sclk	O	sclk domain clock gate signal indicates that the UART is inactive, so clocks may be gated to put the device in a low-power(lp) mode. **Exists**: CLK_GATE_EN==1 && CLOCK_MODE==2 **Active State**: High **Synchronous to**: sclk **Registered**: Yes
baudout_n	O	Transmit clock output **Exists**: BAUD_CLK==1 **Active State**: Low **Synchronous to**: pclk (in single clock configuration),sclk (in two clock configuration) **Registered**: No

5.9 Debug Interface Signals


- debug

Table 5-9 Debug Interface Signals

Port Name	I/O	Description
debug[31:0]	O	On-chip debug signal as follows: debug[31:14] = RAZ debug[13] = RX push indication (RBR or RX FIFO) debug[12] = TX pop indication (THR or TX FIFO) debug[11:10] = Receive Trigger (FCR[7:6]) debug[9:8] = TX Empty Trigger (FCR[5:4]) debug[7] = DMA Mode (FCR[3]) debug[6:1] = Individual interrupt sources: ■ debug[6] = Line status Interrupt ■ debug[5] = Data available Interrupt ■ debug[4] = character Timeout Interrupt ■ debug[3] = THRE interrupt ■ debug[2] = modem status interrupt ■ debug[1] = busy detect interrupt ■ debug[0] = FIFO enable (FCR[0]) NOTE: The debug[1] signal (busy detect interrupt) is never asserted if UART_16550_COMPATIBLE = YES in coreConsultant **Exists**: DEBUG==1 **Active State**: High **Synchronous to**: pclk **Registered**: No

5.10 RS485 Interface Signals

Table 5-10 RS485 Interface Signals

Port Name	I/O	Description
re	O	Receiver Enable Signal. This signal is used to activate and de-activate the Receiver driver. This signal is asserted before the start of receiving serial transfer from DW_apb_uart. This signal is controlled through: ■ Writing into the RE_EN register; this serves as software override option. ■ If RE_EN is programmed to 1, then based on the data available in the TX FIFO DW_apb_uart controller automatically controls the 're' signal. Polarity of this signal is set by RE_POL bit in TCR register. **Exists**: (UART_RS485_INTERFACE_EN==1) **Active State**: High, if TCR[1]=1; Low, if TCR[1]=0 **Synchronous to**: pclk (in single clock configuration),sclk (in two clock configuration) **Registered**: Yes

Table 5-10 RS485 Interface Signals (Continued)

Port Name	I/O	Description
de	O	Driver Enable Signal. This signal is used to activate and de-activate the transmitter driver. This signal is asserted before the start of transmitting serial transfer from DW_apb_uart. This signal is controlled through: ■ Writing into the DE_EN register; this serves as software override option ■ If DE_EN is programmed to 1, then based on the data available in the TX FIFO DW_apb_uart controller automatically controls the 'de' signal. Polarity of this signal is set by DE_POL bit in TCR register. **Exists**: (UART_RS485_INTERFACE_EN==1) **Active State**: High, if TCR[2]=1; Low, if TCR[2]=0 **Synchronous to**: pclk (in single clock configuration),sclk (in two clock configuration) **Registered**: Yes
rs485_en	O	RS485 Enable Signal. This signal indicates whether the DW_apb_uart is enabled for RS485 Mode or RS232 Mode. 0 - RS232 Mode. 1 - RS485 ModeReceiver Enable Signal. **Exists**: (UART_RS485_INTERFACE_EN==1) **Active State**: High **Synchronous to**: pclk **Registered**: Yes

5.11 Interrupt Interface Signals

Table 5-11 Interrupt Interface Signals

Port Name	I/O	Description
intr	O	Interrupt **Exists**: Always **Active State**: High **Synchronous to**: pclk **Registered**: Yes

6 Registers

This chapter describes the programmable registers of the DW_apb_uart.

6.1 Register Memory Map

The DW_apb_uart has a number of internal registers that are accessed through the 5-bit address bus.

Note Since DW_apb_uart registers are only located 32-bit boundaries, paddr[1:0] may be tied low permanently, if so desired. This would allow backward compatibility with standard 16550 UART programmability.

Table 6-1 summarizes the register memory map for the DW_apb_uart:

Table 6-1 DW_apb_uart Memory Map

Name	Address Offset	Width	R/W	Description
RBR	0x00	32 bits	R	Receive Buffer Register **Reset Value:** 0x0 **Dependencies:** LCR[7] bit = 0
THR		32 bits	W	Transmit Holding Register **Reset Value:** 0x0 **Dependencies:** LCR[7] bit = 0
DLL		32 bits	R/W	Divisor Latch (Low) **Reset Value:** 0x0 **Dependencies:** LCR[7] bit = 1
DLH	0x04	32 bits	R/W	Divisor Latch (High) **Reset Value:** 0x0 **Dependencies:** LCR[7] bit = 1
IER		32 bits	R/W	Interrupt Enable Register **Reset Value:** 0x0 **Dependencies:** LCR[7] bit = 0

Table 6-1 DW_apb_uart Memory Map (Continued)

Name	Address Offset	Width	R/W	Description
IIR	0x08	32 bits	R	Interrupt Identification Register **Reset Value:** 0x01
FCR		32 bits	W	FIFO Control Register **Reset Value:** 0x0
LCR	0x0C	32 bits	R/W	Line Control Register **Reset Value:** 0x0
MCR	0x10	32 bits	R/W	Modem Control Register **Reset Value:** 0x0
LSR	0x14	32 bits	R	Line Status Register **Reset Value:** 0x60
MSR	0x18	32 bits	R	Modem Status Register **Reset Value:** 0x0
SCR	0x1C	32 bits	R/W	Scratchpad Register **Reset Value:** 0x0
LPDLL	0x20	32 bits	R/W	Low Power Divisor Latch (Low) Register **Reset Value:** 0x0 **Dependencies:** LCR[7] bit = 1
LPDLH	0x24	32 bits	R/W	Low Power Divisor Latch (High) Register **Reset Value:** 0x0 **Dependencies:** LCR[7] bit = 1
Reserved	0x28 - 0x2C	–	–	–
SRBR	0x30 - 0x6C	32 bits	R	Shadow Receive Buffer Register **Reset Value:** 0x0 **Dependencies:** LCR[7] bit = 0
STHR		32 bits	W	Shadow Transmit Holding Register **Reset Value:** 0x0 **Dependencies:** LCR[7] bit = 0
FAR	0x70	32 bits	R/W	FIFO Access Register **Reset Value:** 0x0
TFR	0x74	32 bits	R	Transmit FIFO Read **Reset Value:** 0x0
RFW	0x78	32 bits	W	Receive FIFO Write **Reset Value:** 0x0
USR	0x7C	32 bits	R	UART Status Register **Reset Value:** 0x6

Table 6-1 DW_apb_uart Memory Map (Continued)

Name	Address Offset	Width	R/W	Description
TFL	0x80	See Description (page 152)	R	Transmit FIFO Level **Width:** *FIFO_ADDR_WIDTH* + 1 **Reset Value:** 0x0
RFL	0x84	See Description (page 152)	R	Receive FIFO Level **Width:** *FIFO_ADDR_WIDTH* + 1 **Reset Value:** 0x0
SRR	0x88	32 bits	W	Software Reset Register **Reset Value:** 0x0
SRTS	0x8C	32 bits	R/W	Shadow Request to Send **Reset Value:** 0x0
SBCR	0x90	32 bits	R/W	Shadow Break Control Register **Reset Value:** 0x0
SDMAM	0x94	32 bits	R/W	Shadow DMA Mode **Reset Value:** 0x0
SFE	0x98	32 bits	R/W	Shadow FIFO Enable **Reset Value:** 0x0
SRT	0x9C	32 bits	R/W	Shadow RCVR Trigger **Reset Value:** 0x0
STET	0xA0	32 bits	R/W	Shadow TX Empty Trigger **Reset Value:** 0x0
HTX	0xA4	32 bits	R/W	Halt TX **Reset Value:** 0x0
DMASA	0xA8	1 bit	W	DMA Software Acknowledge **Reset Value:** 0x0
TCR	0xAC	32 bits	R/W	Transceiver Control Register **Reset Value**: 0x6 **Dependencies**: UART_RS485_INTERFACE_EN=1
DE_EN	0xB0	32 bits	R/W	Driver Output Enable Register. **Reset Value**: 0x0 **Dependencies**: UART_RS485_INTERFACE_EN=1
RE_EN	0xB4	32 bits	R/W	Receiver Output Enable Register. **Reset Value**: 0x0 **Dependencies**: UART_RS485_INTERFACE_EN=1
DET	0xB8	32 bits	R/W	Driver Output Enable Timing Register. **Reset Value**: 0x0 **Dependencies**: UART_RS485_INTERFACE_EN=1

Table 6-1 DW_apb_uart Memory Map (Continued)

Name	Address Offset	Width	R/W	Description
TAT	0xBC	32 bits	R/W	TurnAround Timing Register. **Reset Value**: 0x0 **Dependencies**: UART_RS485_INTERFACE_EN=1
DLF	0xC0	32 bits	R/W	Divisor Latch Fractional Value. **Reset Value**: 0x0 **Dependencies**: FRACTIONAL_BAUD_DIVISOR_EN=1
RAR	0xC4	32 bits	R/W	Receive Address Register **Reset Value**: 0x0 **Dependencies**: UART_9BIT_DATA_EN=1
TAR	0xC8	32 bits	R/W	Transmit Address Register **Reset Value**: 0x0 **Dependencies**: UART_9BIT_DATA_EN=1
LCR_EXT	0xCC	32 bits	R/W	Line Extended Control Register **Reset Value**: 0x0 **Dependencies**: UART_9BIT_DATA_EN=1
–	0xD0 - 0xF0	–	–	–
CPR	0xF4	32 bits	R	Component Parameter Register **Reset Value:** Configuration-dependent
UCV	0xF8	32 bits	R	UART Component Version **Reset Value:** See the Releases table in the AMBA 2 release notes.
CTR	0xFC	32 bits	R	Component Type Register **Reset Value:** 0x44570110

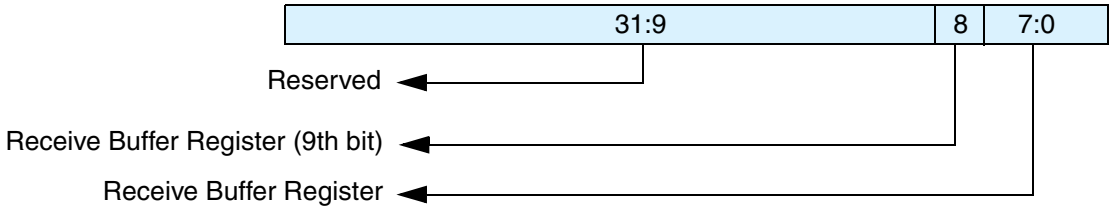
6.2 Register and Field Descriptions

The following subsections describe the data fields of the DW_apb_uart registers.

6.2.1 RBR

- **Name:** Receive Buffer Register
- **Size:** 32 bits
- **Address Offset:** 0x00
- **Read/write access:** read-only

This register can be accessed only when the DLAB bit (LCR[7]) is cleared.

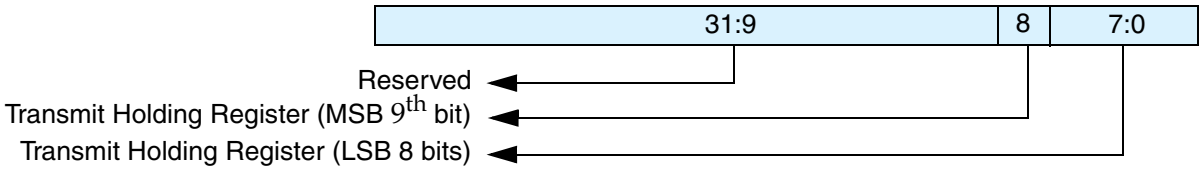


Bits	Name	R/W	Description
31:9	Reserved and read as 0		
8	Receive Buffer register (MSB 9^{th} bit)	R	Data byte received on the serial input port (sin) in UART mode for the MSB 9^{th} bit. It is applicable only when UART_9BIT_DATA_EN=1 **Reset Value**: 0x0
7:0	Receive Buffer Register (LSB 8 bits)	R	Data byte received on the serial input port (sin) in UART mode, or the serial infrared input (sir_in) in infrared mode. The data in this register is valid only if the Data Ready (DR) bit in the Line Status Register (LSR) is set. If in non-FIFO mode (FIFO_MODE = NONE) or FIFOs are disabled (FCR[0] set to 0), the data in the RBR must be read before the next data arrives, otherwise it is overwritten, resulting in an over-run error. If in FIFO mode (FIFO_MODE != NONE) and FIFOs are enabled (FCR[0] set to 1), this register accesses the head of the receive FIFO. If the receive FIFO is full and this register is not read before the next data character arrives, then the data already in the FIFO is preserved, but any incoming data are lost and an over-run error occurs. **Reset Value:** 0x0

6.2.2 THR

- **Name:** Transmit Holding Register
- **Size:** 32 bits
- **Address Offset:** 0x00
- **Read/write access:** write-only

This register can be accessed only when the DLAB bit (LCR[7]) is cleared.

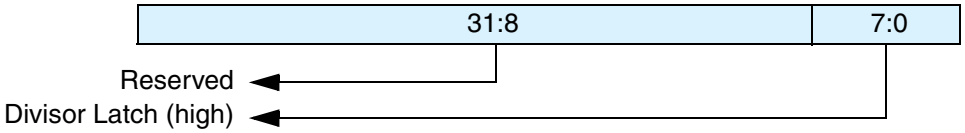


Bits	Name	R/W	Description
31:9	Reserved and read as 0		
8	Transmit Holding Register (MSB 9^{th} bit)	W	Data to be transmitted on the serial output port (sout) in UART mode for the MSB 9^{th} bit. It is applicable only when UART_9BIT_DATA_EN=1. **Reset Value**: 0x0
7:0	Transmit Holding Register (LSB 8 bits)	W	Data to be transmitted on the serial output port (sout) in UART mode or the serial infrared output (sir_out_n) in infrared mode. Data should only be written to the THR when the THR Empty (THRE) bit (LSR[5]) is set. If in non-FIFO mode or FIFOs are disabled (FCR[0] = 0) and THRE is set, writing a single character to the THR clears the THRE. Any additional writes to the THR before the THRE is set again causes the THR data to be overwritten. If in FIFO mode and FIFOs are enabled (FCR[0] = 1) and THRE is set, x number of characters of data may be written to the THR before the FIFO is full. The number x (default=16) is determined by the value of FIFO Depth that you set during configuration. Any attempt to write data when the FIFO is full results in the write data being lost. **Reset Value:** 0x0

6.2.3 DLH

- **Name:** Divisor Latch High
- **Size:** 32 bits
- **Address Offset:** 0x04
- **Read/write access:** read/write

If UART_16550_COMPATIBLE = No, then this register can be accessed only when the DLAB bit (LCR[7]) is set and the UART is not busy—that is, USR[0] is 0; otherwise this register can be accessed only when the DLAB bit (LCR[7]) is set.

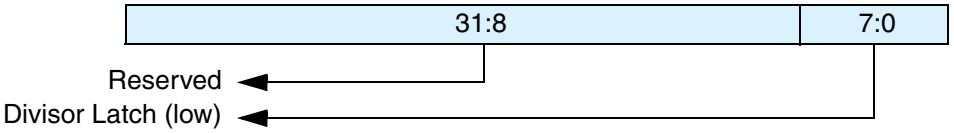


Bits	Name	R/W	Description
31:8	Reserved and read as 0		
7:0	Divisor Latch (High)	R/W	Upper 8-bits of a 16-bit, read/write, Divisor Latch register that contains the baud rate divisor for the UART. The output baud rate is equal to the serial clock (pclk if one clock design, sclk if two clock design (CLOCK_MODE = Enabled) frequency divided by sixteen times the value of the baud rate divisor, as follows: baud rate = (serial clock freq) / (16 * divisor). Note that with the Divisor Latch Registers (DLL and DLH) set to 0, the baud clock is disabled and no serial communications occur. Also, once the DLH is set, at least 8 clock cycles of the slowest DW_apb_uart clock should be allowed to pass before transmitting or receiving data. **Reset Value:** 0x0

6.2.4 DLL

- **Name:** Divisor Latch Low
- **Size:** 32 bits
- **Address Offset:** 0x00
- **Read/write access:** read/write

If UART_16550_COMPATIBLE = No, then this register can be accessed only when the DLAB bit (LCR[7]) is set and the UART is not busy—that is, USR[0] is 0; otherwise this register can be accessed only when the DLAB bit (LCR[7]) is set.

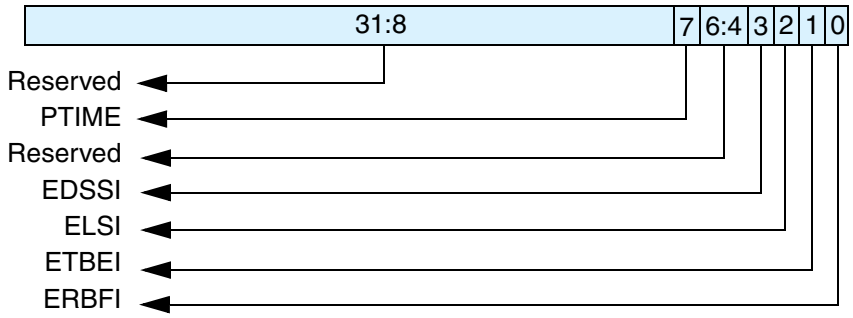


Bits	Name	R/W	Description
31:8	Reserved and read as 0		
7:0	Divisor Latch (Low)	R/W	Lower 8 bits of a 16-bit, read/write, Divisor Latch register that contains the baud rate divisor for the UART. The output baud rate is equal to the serial clock (pclk if one clock design, sclk if two clock design (CLOCK_MODE = Enabled) frequency divided by sixteen times the value of the baud rate divisor, as follows: baud rate = (serial clock freq) / (16 * divisor). Note that with the Divisor Latch Registers (DLL and DLH) set to 0, the baud clock is disabled and no serial communications occur. Also, once the DLL is set, at least 8 clock cycles of the slowest DW_apb_uart clock should be allowed to pass before transmitting or receiving data. **Reset Value:** 0x0

6.2.5 IER

- **Name:** Interrupt Enable Register
- **Size:** 32 bits
- **Address Offset:** 0x04
- **Read/write access:** read/write

This register can be accessed only when the DLAB bit (LCR[7]) is cleared.

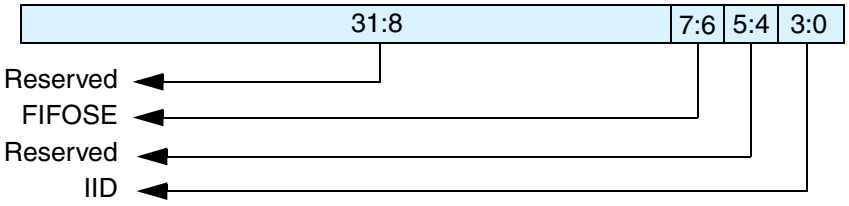


Bits	Name	R/W	Description
31:8	Reserved and read as 0		
7	PTIME	R/W	Programmable THRE Interrupt Mode Enable that can be written to only when THRE_MODE_USER = Enabled, always readable. This is used to enable/disable the generation of THRE Interrupt. ■ 0 – disabled ■ 1 – enabled **Reset Value:** 0x0
6:4	Reserved and read as 0		
3	EDSSI	R/W	Enable Modem Status Interrupt. This is used to enable/disable the generation of Modem Status Interrupt. This is the fourth highest priority interrupt. ■ 0 – disabled ■ 1 – enabled **Reset Value:** 0x0
2	ELSI	R/W	Enable Receiver Line Status Interrupt. This is used to enable/disable the generation of Receiver Line Status Interrupt. This is the highest priority interrupt. ■ 0 – disabled ■ 1 – enabled **Reset Value:** 0x0

Bits	Name	R/W	Description
1	ETBEI	R/W	Enable Transmit Holding Register Empty Interrupt. This is used to enable/disable the generation of Transmitter Holding Register Empty Interrupt. This is the third highest priority interrupt. ■ 0 – disabled ■ 1 – enabled **Reset Value:** 0x0
0	ERBFI	R/W	Enable Received Data Available Interrupt. This is used to enable/disable the generation of Received Data Available Interrupt and the Character Timeout Interrupt (if in FIFO mode and FIFOs enabled). These are the second highest priority interrupts. ■ 0 – disabled ■ 1 – enabled **Reset Value:** 0x0

6.2.6 IIR

- **Name:** Interrupt Identity Register
- **Size:** 32 bits
- **Address Offset:** 0x08
- **Read/write access:** read-only



Bits	Name	R/W	Description
31:8	Reserved and read as 0		
7:6	FIFOs Enabled (or FIFOSE)	R	FIFOs Enabled. This is used to indicate whether the FIFOs are enabled or disabled. ■ 00 – disabled ■ 11 – enabled **Reset Value:** 0x00
5:4	Reserved	N/A	Reserved and read as 0

Bits	Name	R/W	Description
3:0	Interrupt ID (or IID)	R	Interrupt ID. This indicates the highest priority pending interrupt which can be one of the following types: ■ 0000 – modem status ■ 0001 – no interrupt pending ■ 0010 – THR empty ■ 0100 – received data available ■ 0110 – receiver line status ■ 0111 – busy detect ■ 1100 – character timeout The interrupt priorities are split into several levels that are detailed in Table 6-2 on page 127. **Note** An interrupt of type 0111 (busy detect) is never indicated if UART_16550_COMPATIBLE = YES in coreConsultant. Bit 3 indicates an interrupt can only occur when the FIFOs are enabled and used to distinguish a Character Timeout condition interrupt. **Reset Value:** 0x01

Table 6-2 Interrupt Control Functions

Interrupt ID				Interrupt Set and Reset Functions			
Bit 3	Bit 2	Bit 1	Bit 0	Priority Level	Interrupt Type	Interrupt Source	Interrupt Reset Control
0	0	0	1	–	None	None	–
0	1	1	0	Highest	Receiver line status	Overrun/parity/ framing errors, break interrupt, or address received interrupt	Reading the line status register. In addition to LSR read, the Receiver line status is also cleared when RX_FIFO is read.
0	1	0	0	Second	Received data available	Receiver data available (non-FIFO mode or FIFOs disabled) or RCVR FIFO trigger level reached (FIFO mode and FIFOs enabled)	Reading the receiver buffer register (non-FIFO mode or FIFOs disabled) or the FIFO drops below the trigger level (FIFO mode and FIFOs enabled)
1	1	0	0	Second	Character timeout indication	No characters in or out of the RCVR FIFO during the last 4 character times and there is at least 1 character in it during this time	Reading the receiver buffer register

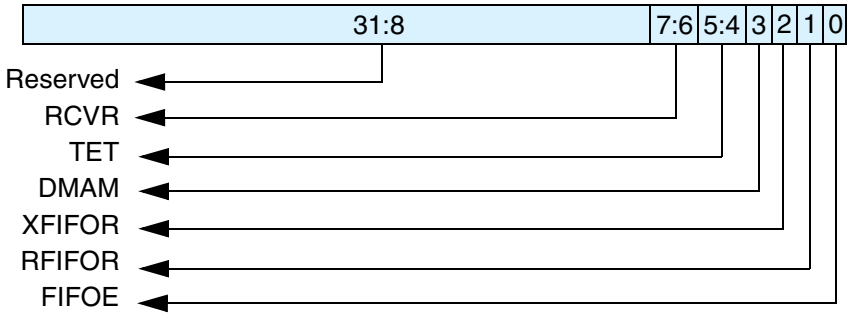
Table 6-2 Interrupt Control Functions (Continued)

Interrupt ID				Interrupt Set and Reset Functions			
Bit 3	**Bit 2**	**Bit 1**	**Bit 0**	**Priority Level**	**Interrupt Type**	**Interrupt Source**	**Interrupt Reset Control**
0	0	1	0	Third	Transmit holding register empty	Transmitter holding register empty (Prog. THRE Mode disabled) or XMIT FIFO at or below threshold (Prog. THRE Mode enabled)	Reading the IIR register (if source of interrupt); or, writing into THR (FIFOs or THRE Mode not selected or disabled) or XMIT FIFO above threshold (FIFOs and THRE Mode selected and enabled).
0	0	0	0	Fourth	Modem status	Clear to send or data set ready or ring indicator or data carrier detect. Note that if auto flow control mode is enabled, a change in CTS (that is, DCTS set) does not cause an interrupt.	Reading the Modem status register
0	1	1	1	Fifth	Busy detect indication	UART_16550_COMPATIBLE = NO and master has tried to write to the Line Control Register while the DW_apb_uart is busy (USR[0] is set to 1).	Reading the UART status register

6.2.7 FCR

- **Name:** FIFO Control Register
- **Size:** 32 bits
- **Address Offset:** 0x08
- **Read/write access:** write-only

This register is valid only when the DW_apb_uart is configured to have FIFOs implemented (FIFO_MODE != NONE). If FIFOs are not implemented, this register does not exist and writing to this register address has no effect.

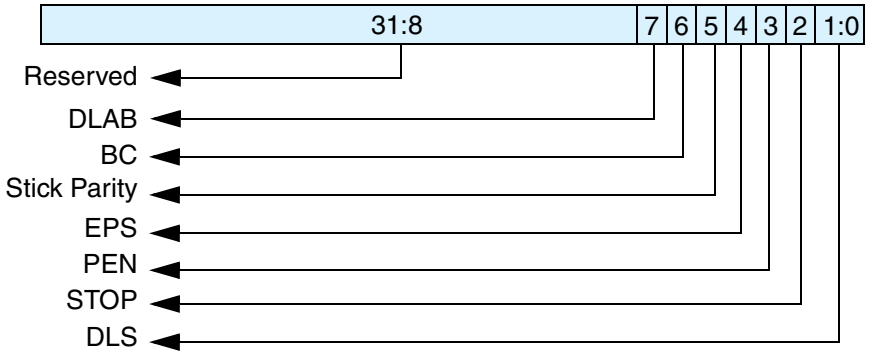


Bits	Name	R/W	Description
31:8	Reserved and read as 0		
7:6	RCVR Trigger (or RT)	W	RCVR Trigger. This is used to select the trigger level in the receiver FIFO at which the Received Data Available Interrupt is generated. In auto flow control mode, this trigger is used to determine when the rts_n signal is de-asserted only when RTC_FCT is disabled. It also determines when the dma_rx_req_n signal is asserted in certain modes of operation. For details on DMA support, refer to "DMA Support" on page 72. The following trigger levels are supported: ■ 00 – 1 character in the FIFO ■ 01 – FIFO ¼ full ■ 10 – FIFO ½ full ■ 11 – FIFO 2 less than full **Reset Value:** 0x0

Bits	Name	R/W	Description
5:4	TX Empty Trigger (or TET)	W	TX Empty Trigger. Writes have no effect when THRE_MODE_USER = Disabled. This is used to select the empty threshold level at which the THRE Interrupts are generated when the mode is active. It also determines when the dma_tx_req_n signal is asserted when in certain modes of operation. For details on DMA support, refer to “DMA Support” on page 72. The following trigger levels are supported: ■ 00 – FIFO empty ■ 01 – 2 characters in the FIFO ■ 10 – FIFO ¼ full ■ 11 – FIFO ½ full **Reset Value:** 0x0
3	DMA Mode (or DMAM)	W	DMA Mode. This determines the DMA signalling mode used for the dma_tx_req_n and dma_rx_req_n output signals when additional DMA handshaking signals are not selected (DMA_EXTRA = No). For details on DMA support, refer to “DMA Support” on page 72. ■ 0 – mode 0 ■ 1 – mode 1 **Reset Value:** 0x0
2	XMIT FIFO Reset (or XFIFOR)	W	XMIT FIFO Reset. This resets the control portion of the transmit FIFO and treats the FIFO as empty. This also de-asserts the DMA TX request and single signals when additional DMA handshaking signals are selected (DMA_EXTRA = YES). Note that this bit is 'self-clearing'. It is not necessary to clear this bit. **Reset Value:** 0x0
1	RCVR FIFO Reset (or RFIFOR)	W	RCVR FIFO Reset. This resets the control portion of the receive FIFO and treats the FIFO as empty. This also de-asserts the DMA RX request and single signals when additional DMA handshaking signals are selected (DMA_EXTRA = YES). Note that this bit is 'self-clearing'. It is not necessary to clear this bit. **Reset Value:** 0x0
0	FIFO Enable (or FIFOE)	W	FIFO Enable. This enables/disables the transmit (XMIT) and receive (RCVR) FIFOs. Whenever the value of this bit is changed both the XMIT and RCVR controller portion of FIFOs is reset. **Reset Value:** 0x0

6.2.8 LCR

- **Name:** Line Control Register
- **Size:** 32 bits
- **Address Offset:** 0x0C
- **Read/write access:** read/write

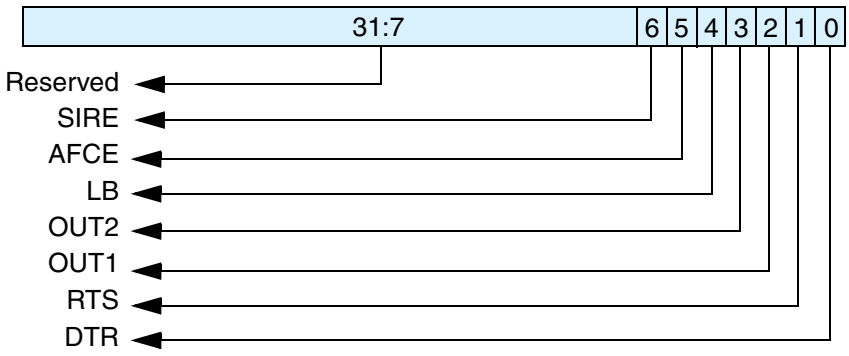


Bits	Name	R/W	Description
31:8	Reserved and read as 0		
7	DLAB	R/W	Divisor Latch Access Bit. If UART_16550_COMPATIBLE = NO, then writeable only when UART is not busy (USR[0] is 0); otherwise always writable, always readable. This bit is used to enable reading and writing of the Divisor Latch register (DLL and DLH/LPDLL and LPDLH) to set the baud rate of the UART. This bit must be cleared after initial baud rate setup in order to access other registers. **Reset Value:** 0x0
6	Break (or BC)	R/W	Break Control Bit.This is used to cause a break condition to be transmitted to the receiving device. If set to 1, the serial output is forced to the spacing (logic 0) state. When not in Loopback Mode, as determined by MCR[4], the sout line is forced low until the Break bit is cleared. If SIR_MODE = Enabled and active (MCR[6] set to 1) the sir_out_n line is continuously pulsed. When in Loopback Mode, the break condition is internally looped back to the receiver and the sir_out_n line is forced low. **Reset Value:** 0x0
5	Stick Parity	R/W	Stick Parity. If UART_16550_COMPATIBLE = NO, then writeable only when UART is not busy (USR[0] is 0); otherwise always writable, always readable. This bit is used to force parity value. When PEN, EPS, and Stick Parity are set to 1, the parity bit is transmitted and checked as logic 0. If PEN and Stick Parity are set to 1 and EPS is a logic 0, then parity bit is transmitted and checked as a logic 1. If this bit is set to 0, Stick Parity is disabled. **Reset Value:** 0x0

Bits	Name	R/W	Description
4	EPS	R/W	Even Parity Select. If UART_16550_COMPATIBLE = NO, then writeable only when UART is not busy (USR[0] is 0); otherwise always writable, always readable. This is used to select between even and odd parity, when parity is enabled (PEN set to 1). If set to 1, an even number of logic 1s is transmitted or checked. If set to 0, an odd number of logic 1s is transmitted or checked. **Reset Value:** 0x0
3	PEN	R/W	Parity Enable. If UART_16550_COMPATIBLE = NO, then writeable only when UART is not busy (USR[0] is 0); otherwise always writable, always readable. This bit is used to enable and disable parity generation and detection in transmitted and received serial character respectively. ■ 0 – parity disabled ■ 1 – parity enabled **Reset Value:** 0x0
2	STOP	R/W	Number of stop bits. If UART_16550_COMPATIBLE = NO, then writeable only when UART is not busy (USR[0] is 0); otherwise always writable, always readable. This is used to select the number of stop bits per character that the peripheral transmits and receives. If set to 0, one stop bit is transmitted in the serial data. If set to 1 and the data bits are set to 5 (LCR[1:0] set to 0) one and a half stop bits is transmitted. Otherwise, two stop bits are transmitted. Note that regardless of the number of stop bits selected, the receiver checks only the first stop bit. ■ 0 – 1 stop bit ■ 1 – 1.5 stop bits when DLS (LCR[1:0]) is 0, else 2 stop bit NOTE: The STOP bit duration implemented by DW_apb_uart may appear longer due to idle time inserted between characters for some configurations and baud clock divisor values in the transmit direction; for details on idle time between transmitted transfers, refer to “Back-to-Back Character Stream Transmission” on page 62. **Reset Value:** 0x0
1:0	DLS (or CLS, as used in legacy)	R/W	Data Length Select. If UART_16550_COMPATIBLE = NO, then writable only when UART is not busy (USR[0] is 0); otherwise always writable and readable. When DLS_E in LCR_EXT is set to 0, this register is used to select the number of data bits per character that the peripheral transmits and receives. The number of bits that may be selected are as follows: ■ 00 – 5 bits ■ 01 – 6 bits ■ 10 – 7 bits ■ 11 – 8 bits **Reset Value:** 0x0

6.2.9 MCR

- **Name:** Modem Control Register
- **Size:** 32 bits
- **Address Offset:** 0x10
- **Read/write access:** read/write



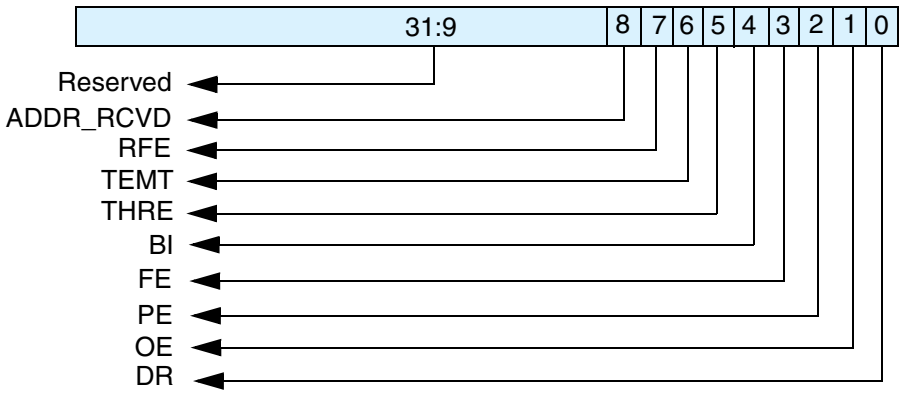
Bits	Name	R/W	Description
31:7	Reserved and read as 0		
6	SIRE	R/W	SIR Mode Enable. Writeable only when SIR_MODE = Enabled, always readable. This is used to enable/disable the IrDA SIR Mode features as described in "IrDA 1.0 SIR Protocol" on page 55. ■ 0 – IrDA SIR Mode disabled ■ 1 – IrDA SIR Mode enabled **Reset Value:** 0x0 **Note** To enable SIR mode, write the appropriate value to the MCR register before writing to the LCR register. For details of the recommended programming sequence, refer to "Programing Examples" on page 177.
5	AFCE	R/W	Auto Flow Control Enable. Writeable only when AFCE_MODE = Enabled, always readable. When FIFOs are enabled and the Auto Flow Control Enable (AFCE) bit is set, Auto Flow Control features are enabled as described in "Auto Flow Control" on page 64. ■ 0 – Auto Flow Control Mode disabled ■ 1 – Auto Flow Control Mode enabled **Reset Value:** 0x0

Bits	Name	R/W	Description
4	LoopBack (or LB)	R/W	LoopBack Bit. This is used to put the UART into a diagnostic mode for test purposes. If operating in UART mode (SIR_MODE != Enabled or not active, MCR[6] set to 0), data on the sout line is held high, while serial data output is looped back to the sin line, internally. In this mode all the interrupts are fully functional. Also, in loopback mode, the modem control inputs (dsr_n, cts_n, ri_n, dcd_n) are disconnected and the modem control outputs (dtr_n, rts_n, out1_n, out2_n) are looped back to the inputs, internally. If operating in infrared mode (SIR_MODE = Enabled AND active, MCR[6] set to 1), data on the sir_out_n line is held low, while serial data output is inverted and looped back to the sir_in line. **Reset Value:** 0x0
3	OUT2	R/W	OUT2. This is used to directly control the user-designated Output2 (out2_n) output. The value written to this location is inverted and driven out on out2_n, that is: ■ 0 – out2_n de-asserted (logic 1) ■ 1 – out2_n asserted (logic 0) Note that in Loopback mode (MCR[4] set to 1), the out2_n output is held inactive high while the value of this location is internally looped back to an input. **Reset Value:** 0x0
2	OUT1	R/W	OUT1. This is used to directly control the user-designated Output1 (out1_n) output. The value written to this location is inverted and driven out on out1_n, that is: ■ 0 – out1_n de-asserted (logic 1) ■ 1 – out1_n asserted (logic 0) Note that in Loopback mode (MCR[4] set to 1), the out1_n output is held inactive high while the value of this location is internally looped back to an input. **Reset Value:** 0x0
1	RTS	R/W	Request to Send. This is used to directly control the Request to Send (rts_n) output. The Request To Send (rts_n) output is used to inform the modem or data set that the UART is ready to exchange data. When Auto RTS Flow Control is not enabled (MCR[5] set to 0), the rts_n signal is set low by programming MCR[1] (RTS) to a high.In Auto Flow Control, AFCE_MODE = Enabled and active (MCR[5] set to 1) and FIFOs enable (FCR[0] set to 1), the rts_n output is controlled in the same way, but is also gated with the receiver FIFO threshold trigger (rts_n is inactive high when above the threshold) only when the RTC Flow Trigger is disabled; otherwise it is gated by the receiver FIFO almost-full trigger, where “almost full” refers to two available slots in the FIFO (rts_n is inactive high when above the threshold). The rts_n signal is de-asserted when MCR[1] is set low. Note that in Loopback mode (MCR[4] set to 1), the rts_n output is held inactive high while the value of this location is internally looped back to an input. **Reset Value:** 0x0

Bits	Name	R/W	Description
0	DTR	R/W	Data Terminal Ready. This is used to directly control the Data Terminal Ready (dtr_n) output. The value written to this location is inverted and driven out on dtr_n, that is: ■ 0 – dtr_n de-asserted (logic 1) ■ 1 – dtr_n asserted (logic 0) The Data Terminal Ready output is used to inform the modem or data set that the UART is ready to establish communications. Note that in Loopback mode (MCR[4] set to 1), the dtr_n output is held inactive high while the value of this location is internally looped back to an input. **Reset Value:** 0x0

6.2.10 LSR

- **Name:** Line Status Register
- **Size:** 32 bits
- **Address Offset:** 0x14
- **Read/write access:** read-only



Bits	Name	R/W	Description
31:9	Reserved and read as 0		
8	ADDR_RCVD	R/W	Address Received bit If 9-bit data mode (LCR_EXT[0]=1) is enabled, this bit is used to indicate that the 9^{th} bit of the receive data is set to 1. This bit can also be used to indicate whether the incoming character is an address or data. ■ 1 - Indicates that the character is an address. ■ 0 - Indicates that the character is data. In the FIFO mode, since the 9^{th} bit is associated with the received character, it is revealed when the character with the 9^{th} bit set to 1 is at the top of the FIFO list. Reading the LSR clears the 9^{th} bit. **NOTE**: You must ensure that an interrupt gets cleared (reading LSR register) before the next address byte arrives. If there is a delay in clearing the interrupt, then software will not be able to distinguish between multiple address related interrupt. **Reset Value**: 0x0

Bits	Name	R/W	Description
7	RFE	R	Receiver FIFO Error bit. This bit is only relevant when FIFO_MODE != NONE AND FIFOs are enabled (FCR[0] set to 1). This is used to indicate if there is at least one parity error, framing error, or break indication in the FIFO. ■ 0 – no error in RX FIFO ■ 1 – error in RX FIFO This bit is cleared when the LSR is read and the character with the error is at the top of the receiver FIFO and there are no subsequent errors in the FIFO. **Reset Value:** 0x0
6	TEMT	R	Transmitter Empty bit. If in FIFO mode (FIFO_MODE != NONE) and FIFOs enabled (FCR[0] set to 1), this bit is set whenever the Transmitter Shift Register and the FIFO are both empty. If in non-FIFO mode or FIFOs are disabled, this bit is set whenever the Transmitter Holding Register and the Transmitter Shift Register are both empty. **Reset Value:** 0x1
5	THRE	R	Transmit Holding Register Empty bit. If THRE_MODE_USER = Disabled or THRE mode is disabled (IER[7] set to 0) and regardless of FIFO's being implemented/enabled or not, this bit indicates that the THR or TX FIFO is empty. This bit is set whenever data is transferred from the THR or TX FIFO to the transmitter shift register and no new data has been written to the THR or TX FIFO. This also causes a THRE Interrupt to occur, if the THRE Interrupt is enabled. If THRE_MODE_USER = Enabled AND FIFO_MODE != NONE and both modes are active (IER[7] set to 1 and FCR[0] set to 1 respectively), the functionality is switched to indicate the transmitter FIFO is full, and no longer controls THRE interrupts, which are then controlled by the FCR[5:4] threshold setting. For more details, see “Programmable THRE Interrupt” on page 68. **Reset Value:** 0x1
4	BI	R	Break Interrupt bit. This is used to indicate the detection of a break sequence on the serial input data. If in UART mode (SIR_MODE = Disabled), it is set whenever the serial input, sin, is held in a logic '0' state for longer than the sum of *start time* + *data bits* + *parity* + *stop bits*. If in infrared mode (SIR_MODE = Enabled), it is set whenever the serial input, sir_in, is continuously pulsed to logic '0' for longer than the sum of *start time* + *data bits* + *parity* + *stop bits*. A break condition on serial input causes one and only one character, consisting of all 0s, to be received by the UART. In FIFO mode, the character associated with the break condition is carried through the FIFO and is revealed when the character is at the top of the FIFO. Reading the LSR clears the BI bit. In non-FIFO mode, the BI indication occurs immediately and persists until the LSR is read. **NOTE:** If a FIFO is full when a break condition is received, a FIFO overrun occurs. The break condition and all the information associated with it—parity and framing errors—is discarded; any information that a break character was received is lost. **Reset Value:** 0x0

Bits	Name	R/W	Description
3	FE	R	Framing Error bit. This is used to indicate the occurrence of a framing error in the receiver. A framing error occurs when the receiver does not detect a valid STOP bit in the received data. In the FIFO mode, since the framing error is associated with a character received, it is revealed when the character with the framing error is at the top of the FIFO. When a framing error occurs, the DW_apb_uart tries to resynchronize. It does this by assuming that the error was due to the start bit of the next character and then continues receiving the other bit; that is, data, and/or parity and stop. It should be noted that the Framing Error (FE) bit (LSR[3]) is set if a break interrupt has occurred, as indicated by Break Interrupt (BI) bit (LSR[4]). This happens because the break character implicitly generates a framing error by holding the sin input to logic 0 for longer than the duration of a character. ■ 0 – no framing error ■ 1 – framing error Reading the LSR clears the FE bit. **Reset Value:** 0x0
2	PE	R	Parity Error bit. This is used to indicate the occurrence of a parity error in the receiver if the Parity Enable (PEN) bit (LCR[3]) is set. In the FIFO mode, since the parity error is associated with a character received, it is revealed when the character with the parity error arrives at the top of the FIFO. It should be noted that the Parity Error (PE) bit (LSR[2]) can be set if a break interrupt has occurred, as indicated by Break Interrupt (BI) bit (LSR[4]). In this situation, the Parity Error bit is set if parity generation and detection is enabled (LCR[3]=1) and the parity is set to odd (LCR[4]=0). ■ 0 – no parity error ■ 1 – parity error Reading the LSR clears the PE bit. **Reset Value:** 0x0
1	OE	R	Overrun error bit. This is used to indicate the occurrence of an overrun error. This occurs if a new data character was received before the previous data was read. In the non-FIFO mode, the OE bit is set when a new character arrives in the receiver before the previous character was read from the RBR. When this happens, the data in the RBR is overwritten. In the FIFO mode, an overrun error occurs when the FIFO is full and a new character arrives at the receiver. The data in the FIFO is retained and the data in the receive shift register is lost. ■ 0 – no overrun error ■ 1 – overrun error Reading the LSR clears the OE bit. **Reset Value:** 0x0

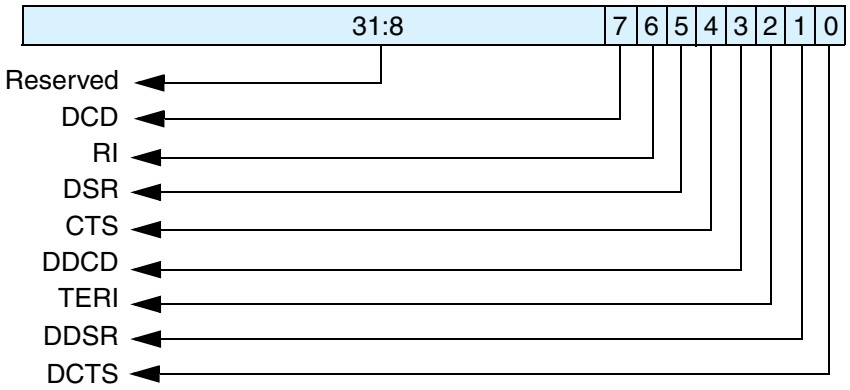
Bits	Name	R/W	Description
0	DR	R	Data Ready bit. This is used to indicate that the receiver contains at least one character in the RBR or the receiver FIFO. ■ 0 – no data ready ■ 1 – data ready This bit is cleared when the RBR is read in non-FIFO mode, or when the receiver FIFO is empty, in FIFO mode. **Reset Value:** 0x0

6.2.11 MSR

- **Name:** Modem Status Register
- **Size:** 32 bits
- **Address Offset:** 0x18
- **Read/write access:** read-only

Whenever bits 0, 1, 2 or 3 are set to logic 1, to indicate a change on the modem control inputs, a modem status interrupt is generated if enabled through the IER, regardless of when the change occurred. The bits of this register can be set after a reset—even though their respective modem signals are inactive—because the

synchronized version of the modem signals have a reset value of 0 and change to value 1 after reset. To prevent unwanted interrupts due to this change, a read of the MSR register can be performed after reset.



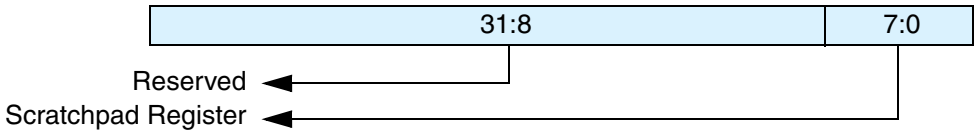
Bits	Name	R/W	Description
31:8	Reserved and read as 0		
7	DCD	R	Data Carrier Detect. This is used to indicate the current state of the modem control line dcd_n. This bit is the complement of dcd_n. When the Data Carrier Detect input (dcd_n) is asserted it is an indication that the carrier has been detected by the modem or data set. ■ 0 – dcd_n input is de-asserted (logic 1) ■ 1 – dcd_n input is asserted (logic 0) In Loopback Mode (MCR[4] set to 1), DCD is the same as MCR[3] (Out2). **Reset Value:** 0x0
6	RI	R	Ring Indicator. This is used to indicate the current state of the modem control line ri_n. This bit is the complement of ri_n. When the Ring Indicator input (ri_n) is asserted it is an indication that a telephone ringing signal has been received by the modem or data set. ■ 0 – ri_n input is de-asserted (logic 1) ■ 1 – ri_n input is asserted (logic 0) In Loopback Mode (MCR[4] set to 1), RI is the same as MCR[2] (Out1). **Reset Value:** 0x0
5	DSR	R	Data Set Ready. This is used to indicate the current state of the modem control line dsr_n. This bit is the complement of dsr_n. When the Data Set Ready input (dsr_n) is asserted it is an indication that the modem or data set is ready to establish communications with the DW_apb_uart. ■ 0 – dsr_n input is de-asserted (logic 1) ■ 1 – dsr_n input is asserted (logic 0) In Loopback Mode (MCR[4] set to 1), DSR is the same as MCR[0] (DTR). **Reset Value:** 0x0

Bits	Name	R/W	Description
4	CTS	R	Clear to Send. This is used to indicate the current state of the modem control line cts_n. This bit is the complement of cts_n. When the Clear to Send input (cts_n) is asserted it is an indication that the modem or data set is ready to exchange data with the DW_apb_uart. ■ 0 – cts_n input is de-asserted (logic 1) ■ 1 – cts_n input is asserted (logic 0) In Loopback Mode (MCR[4] = 1), CTS is the same as MCR[1] (RTS). **Reset Value:** 0x0
3	DDCD	R	Delta Data Carrier Detect. This is used to indicate that the modem control line dcd_n has changed since the last time the MSR was read. ■ 0 – no change on dcd_n since last read of MSR ■ 1 – change on dcd_n since last read of MSR Reading the MSR clears the DDCD bit. In Loopback Mode (MCR[4] = 1), DDCD reflects changes on MCR[3] (Out2). Note, if the DDCD bit is not set and the dcd_n signal is asserted (low) and a reset occurs (software or otherwise), then the DDCD bit is set when the reset is removed if the dcd_n signal remains asserted. **Reset Value:** 0x0
2	TERI	R	Trailing Edge of Ring Indicator. This is used to indicate that a change on the input ri_n (from an active-low to an inactive-high state) has occurred since the last time the MSR was read. ■ 0 – no change on ri_n since last read of MSR ■ 1 – change on ri_n since last read of MSR Reading the MSR clears the TERI bit. In Loopback Mode (MCR[4] = 1), TERI reflects when MCR[2] (Out1) has changed state from a high to a low. **Reset Value:** 0x0
1	DDSR	R	Delta Data Set Ready. This is used to indicate that the modem control line dsr_n has changed since the last time the MSR was read. ■ 0 – no change on dsr_n since last read of MSR ■ 1 – change on dsr_n since last read of MSR Reading the MSR clears the DDSR bit. In Loopback Mode (MCR[4] = 1), DDSR reflects changes on MCR[0] (DTR). Note, if the DDSR bit is not set and the dsr_n signal is asserted (low) and a reset occurs (software or otherwise), then the DDSR bit is set when the reset is removed if the dsr_n signal remains asserted. **Reset Value:** 0x0

Bits	Name	R/W	Description
0	DCTS	R	Delta Clear to Send. This is used to indicate that the modem control line cts_n has changed since the last time the MSR was read. ■ 0 – no change on cts_n since last read of MSR ■ 1 – change on cts_n since last read of MSR Reading the MSR clears the DCTS bit. In Loopback Mode (MCR[4] = 1), DCTS reflects changes on MCR[1] (RTS). Note, if the DCTS bit is not set and the cts_n signal is asserted (low) and a reset occurs (software or otherwise), then the DCTS bit is set when the reset is removed if the cts_n signal remains asserted. **Reset Value:** 0x0

6.2.12 SCR

- **Name:** Scratchpad Register
- **Size:** 32 bits
- **Address Offset:** 0x1C
- **Read/write access:** read/write



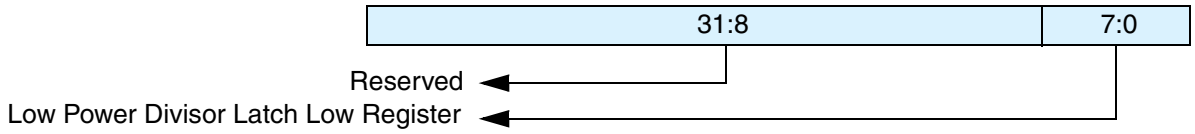
Bits	Name	R/W	Description
31:8	Reserved and read as 0		
7:0	Scratchpad Register	R/W	This register is for programmers to use as a temporary storage space. It has no defined purpose in the DW_apb_uart. **Reset Value:** 0x0

6.2.13 LPDLL

- **Name:** Low Power Divisor Latch Low Register
- **Size:** 32 bits
- **Address Offset:** 0x20
- **Read/write access:** read/write

This register is only valid when the DW_apb_uart is configured to have SIR low-power reception capabilities implemented (SIR_LP_RX = Yes). If SIR low-power reception capabilities are not implemented, this register does not exist and reading from this register address returns 0.

If UART_16550_COMPATIBLE = No, then this register can be accessed only when the DLAB bit (LCR[7]) is set and the UART is not busy—that is, USR[0] is 0; otherwise this register can be accessed only when the DLAB bit (LCR[7]) is set.



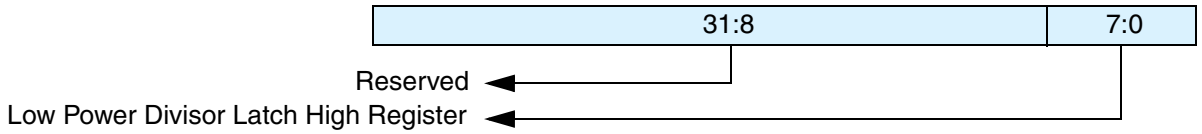
Bits	Name	R/W	Description
31:8	Reserved and read as 0		
7:0	LPDLL	R/W	This register makes up the lower 8-bits of a 16-bit, read/write, Low Power Divisor Latch register that contains the baud rate divisor for the UART, which must give a baud rate of 115.2K. This is required for SIR Low Power (minimum pulse width) detection at the receiver. The output low-power baud rate is equal to the serial clock (sclk) frequency divided by sixteen times the value of the baud rate divisor, as follows: Low power baud rate = (serial clock frequency)/(16* divisor) Therefore, a divisor must be selected to give a baud rate of 115.2K. **NOTE:** When the Low Power Divisor Latch registers (LPDLL and LPDLH) are set to 0, the low-power baud clock is disabled and no low-power pulse detection (or any pulse detection) occurs at the receiver. Also, once the LPDLL is set, at least eight clock cycles of the slowest DW_apb_uart clock should be allowed to pass before transmitting or receiving data. **Reset Value:** 0x0

6.2.14 LPDLH

- **Name:** Low Power Divisor Latch High Register
- **Size:** 32 bits
- **Address Offset:** 0x24
- **Read/write access:** read/write

This register is valid only when the DW_apb_uart is configured to have SIR low-power reception capabilities implemented (SIR_LP_RX = Yes). If SIR low-power reception capabilities are not implemented, this register does not exist and reading from this register address returns 0.

If UART_16550_COMPATIBLE = No, then this register can be accessed only when the DLAB bit (LCR[7]) is set and the UART is not busy—that is, USR[0] is 0; otherwise this register can be accessed only when the DLAB bit (LCR[7]) is set.



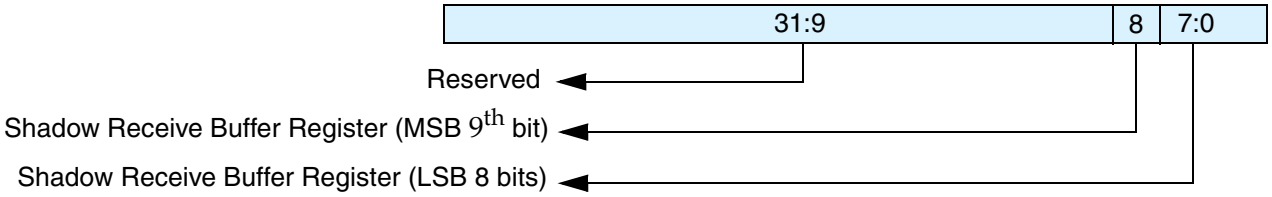
Bits	Name	R/W	Description
31:8	Reserved and read as 0		
7:0	LPDLH	R/W	This register makes up the upper 8-bits of a 16-bit, read/write, Low Power Divisor Latch register that contains the baud rate divisor for the UART, which must give a baud rate of 115.2K. This is required for SIR Low Power (minimum pulse width) detection at the receiver. The output low-power baud rate is equal to the serial clock (sclk) frequency divided by sixteen times the value of the baud rate divisor, as follows: Low power baud rate = (serial clock frequency)/(16* divisor) Therefore, a divisor must be selected to give a baud rate of 115.2K. **NOTE:** When the Low Power Divisor Latch registers (LPDLL and LPDLH) are set to 0, the low-power baud clock is disabled and no low-power pulse detection (or any pulse detection) occurs at the receiver. Also, once the LPDLH is set, at least eight clock cycles of the slowest DW_apb_uart clock should be allowed to pass before transmitting or receiving data. **Reset Value:** 0x0

6.2.15 SRBR

- **Name:** Shadow Receive Buffer Register
- **Size:** 32 bits
- **Address Offset:** 0x30 - 0x6C
- **Read/write access:** read-only

This register is valid only when the DW_apb_uart is configured to have additional shadow registers implemented (SHADOW = YES). If shadow registers are not implemented, this register does not exist and reading from this register address returns 0.

This register can be accessed only when the DLAB bit (LCR[7]) is cleared.



Bits	Name	R/W	Description
31:9	Reserved and read as 0		
8	Shadow Receive Buffer Register (MSB 9th bit)	R	This is a shadow register for the RBR[8] bit. It is applicable only when UART_9BIT_DATA_EN=1. **Reset Value**: 0x0
7:0	Shadow Receive Buffer Register (LSB 8 bits)	R	This is a shadow register for the RBR and has been allocated sixteen 32-bit locations so as to accommodate burst accesses from the master. This register contains the data byte received on the serial input port (sin) in UART mode or the serial infrared input (sir_in) in infrared mode. The data in this register is valid only if the Data Ready (DR) bit in the Line status Register (LSR) is set. If in non-FIFO mode (FIFO_MODE = NONE) or FIFOs are disabled (FCR[0] set to 0), the data in the RBR must be read before the next data arrives, otherwise it is overwritten, resulting in an overrun error. If in FIFO mode (FIFO_MODE != NONE) and FIFOs are enabled (FCR[0] set to 1), this register accesses the head of the receive FIFO. If the receive FIFO is full and this register is not read before the next data character arrives, then the data already in the FIFO are preserved, but any incoming data is lost. An overrun error also occurs. **Reset Value:** 0x0

6.2.16 STHR

- **Name:** Shadow Transmit Holding Register
- **Size:** 32 bits
- **Address Offset:** 0x30 - 0x6C
- **Read/write access:** write

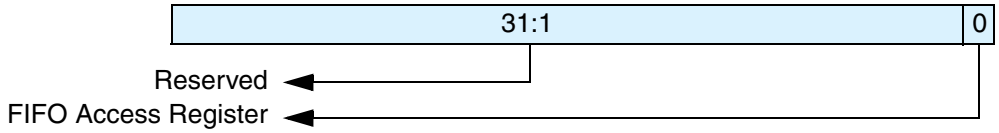
This register is valid only when the DW_apb_uart is configured to have additional shadow registers implemented (SHADOW = YES). If shadow registers are not implemented, this register does not exist, and reading from this register address returns 0.

This register can be accessed only when the DLAB bit (LCR[7]) is cleared.

Bits	Name	R/W	Description
31:9	Reserved and read as 0		
8	Shadow Transmit Holding Register (MSB 9th bit)	W	This is a shadow register for the THR[8] bit. It is applicable only when UART_9BIT_DATA_EN=1 **Reset Value**: 0x0
7:0	Shadow Transmit Holding Register	W	This is a shadow register for the THR and has been allocated sixteen 32-bit locations so as to accommodate burst accesses from the master. This register contains data to be transmitted on the serial output port (sout) in UART mode or the serial infrared output (sir_out_n) in infrared mode. Data should only be written to the THR when the THR Empty (THRE) bit (LSR[5]) is set. If in non-FIFO mode or FIFOs are disabled (FCR[0] set to 0) and THRE is set, writing a single character to the THR clears the THRE. Any additional writes to the THR before the THRE is set again causes the THR data to be overwritten. If in FIFO mode and FIFOs are enabled (FCR[0] set to 1) and THRE is set, x number of characters of data may be written to the THR before the FIFO is full. The number x (default=16) is determined by the value of FIFO Depth that you set during configuration. Any attempt to write data when the FIFO is full results in the write data being lost. **Reset Value:** 0x0

6.2.17 FAR

- **Name:** FIFO Access Register
- **Size:** 32 bits
- **Address Offset:** 0x70
- **Read/write access:** read/write

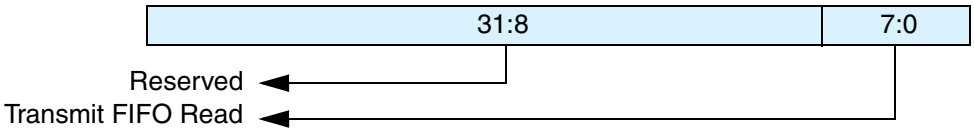


Bits	Name	R/W	Description
31:1	Reserved and read as 0		
0	FIFO Access Register	R/W	Writes have no effect when FIFO_ACCESS = No, always readable. This register is use to enable a FIFO access mode for testing, so that the receive FIFO can be written by the master and the transmit FIFO can be read by the master when FIFOs are implemented and enabled. When FIFOs are not implemented or not enabled it allows the RBR to be written by the master and the THR to be read by the master. ■ 0 – FIFO access mode disabled ■ 1 – FIFO access mode enabled Note, that when the FIFO access mode is enabled/disabled, the control portion of the receive FIFO and transmit FIFO is reset and the FIFOs are treated as empty. **Reset Value:** 0x0

6.2.18 TFR

- **Name:** Transmit FIFO Read
- **Size:** 32 bits
- **Address Offset:** 0x74
- **Read/write access:** read-only

This register is valid only when the DW_apb_uart is configured to have the FIFO access test mode available (FIFO_ACCESS = YES). If not configured, this register does not exist and reading from this register address returns 0.

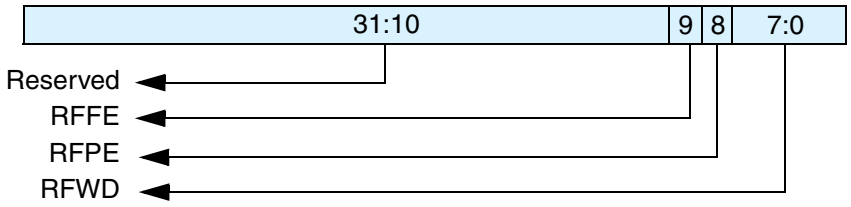


Bits	Name	R/W	Description
31:8	Reserved and read as 0		
7:0	Transmit FIFO Read	R	Transmit FIFO Read. These bits are only valid when FIFO access mode is enabled (FAR[0] is set to 1). When FIFOs are implemented and enabled, reading this register gives the data at the top of the transmit FIFO. Each consecutive read pops the transmit FIFO and gives the next data value that is currently at the top of the FIFO. When FIFOs are not implemented or not enabled, reading this register gives the data in the THR. **Reset Value:** 0x0

6.2.19 RFW

- **Name:** Receive FIFO Write
- **Size:** 32 bits
- **Address Offset:** 0x78
- **Read/write access:** write-only

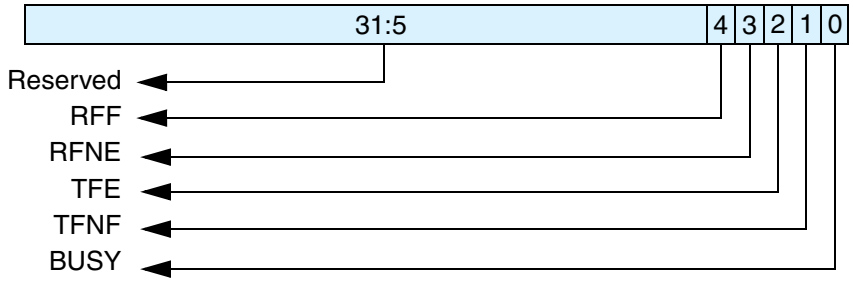
This register is valid only when the DW_apb_uart is configured to have the FIFO access test mode available (FIFO_ACCESS = YES). If not configured, this register does not exist and reading from this register address returns 0.



Bits	Name	R/W	Description
31:10	Reserved and read as 0		
9	RFFE	W	Receive FIFO Framing Error. These bits are only valid when FIFO access mode is enabled (FAR[0] is set to 1). When FIFOs are implemented and enabled, this bit is used to write framing error detection information to the receive FIFO. When FIFOs are not implemented or not enabled, this bit is used to write framing error detection information to the RBR. **Reset Value:** 0x0
8	RFPE	W	Receive FIFO Parity Error. These bits are only valid when FIFO access mode is enabled (FAR[0] is set to 1). When FIFOs are implemented and enabled, this bit is used to write parity error detection information to the receive FIFO. When FIFOs are not implemented or not enabled, this bit is used to write parity error detection information to the RBR. **Reset Value:** 0x0
7:0	RFWD	W	Receive FIFO Write Data. These bits are only valid when FIFO access mode is enabled (FAR[0] is set to 1). When FIFOs are implemented and enabled, the data that is written to the RFWD is pushed into the receive FIFO. Each consecutive write pushes the new data to the next write location in the receive FIFO. When FIFOs are not implemented or not enabled, the data that is written to the RFWD is pushed into the RBR. **Reset Value:** 0x0

6.2.20 USR

- **Name:** UART Status Register
- **Size:** 32 bits
- **Address Offset:** 0x7C
- **Read/write access:** read-only



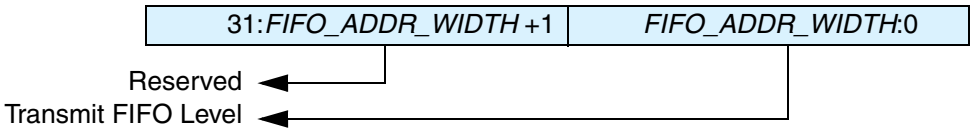
Bits	Name	R/W	Description
31:5	Reserved and read as 0		
4	RFF	R	Receive FIFO Full. This bit is only valid when FIFO_STAT = YES. This is used to indicate that the receive FIFO is completely full. ■ 0 – Receive FIFO not full ■ 1 – Receive FIFO Full This bit is cleared when the RX FIFO is no longer full. **Reset Value:** 0x0
3	RFNE	R	Receive FIFO Not Empty. This bit is only valid when FIFO_STAT = YES. This is used to indicate that the receive FIFO contains one or more entries. ■ 0 – Receive FIFO is empty ■ 1 – Receive FIFO is not empty This bit is cleared when the RX FIFO is empty. **Reset Value:** 0x0
2	TFE	R	Transmit FIFO Empty. This bit is only valid when FIFO_STAT = YES. This is used to indicate that the transmit FIFO is completely empty. ■ 0 – Transmit FIFO is not empty ■ 1 – Transmit FIFO is empty This bit is cleared when the TX FIFO is no longer empty. **Reset Value:** 0x1

Bits	Name	R/W	Description
1	TFNF	R	Transmit FIFO Not Full. This bit is only valid when FIFO_STAT = YES. This is used to indicate that the transmit FIFO in not full. ■ 0 – Transmit FIFO is full ■ 1 – Transmit FIFO is not full This bit is cleared when the TX FIFO is full. **Reset Value:** 0x1
0	BUSY	R	UART Busy. This bit is valid only when UART_16550_COMPATIBLE = NO and indicates that a serial transfer is in progress; when cleared, indicates that the DW_apb_uart is idle or inactive. ■ 0 – DW_apb_uart is idle or inactive ■ 1 – DW_apb_uart is busy (actively transferring data) This bit will be set to 1 (busy) under any of the following conditions: 1. Transmission in progress on serial interface 2. Transmit data present in THR, when FIFO access mode is not being used (FAR = 0) and the baud divisor is non-zero ({DLH,DLL} does not equal 0) when the divisor latch access bit is 0 (LCR.DLAB = 0) 3. Reception in progress on the interface 4. Receive data present in RBR, when FIFO access mode is not being used (FAR = 0) **NOTE:** It is possible for the UART Busy bit to be cleared even though a new character may have been sent from another device. That is, if the DW_apb_uart has no data in THR and RBR and there is no transmission in progress and a start bit of a new character has just reached the DW_apb_uart. This is due to the fact that a valid start is not seen until the middle of the bit period and this duration is dependent on the baud divisor that has been programmed. If a second system clock has been implemented (CLOCK_MODE = Enabled), the assertion of this bit is also delayed by several cycles of the slower clock. **Reset Value:** 0x0

6.2.21 TFL

- **Name:** Transmit FIFO Level
- **Size:** *FIFO_ADDR_WIDTH* + 1
- **Address Offset:** 0x80
- **Read/write access:** read-only

This register is valid only when the DW_apb_uart is configured to have additional FIFO status registers implemented (FIFO_STAT = YES). If status registers are not implemented, this register does not exist and reading from this register address returns 0.

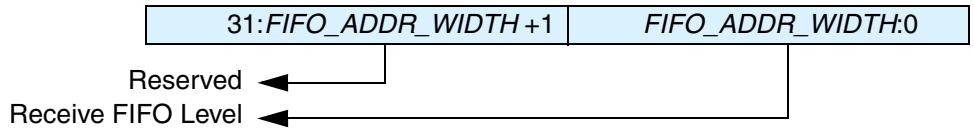


Bits	Name	R/W	Description
31:*FIFO_ADDR_WIDTH* + 1	Reserved and read as 0		
FIFO_ADDR_WIDTH:0	Transmit FIFO Level	R	Transmit FIFO Level. This indicates the number of data entries in the transmit FIFO. **Reset Value:** 0x0

6.2.22 RFL

- **Name:** Receive FIFO Level
- **Size:** *FIFO_ADDR_WIDTH* + 1
- **Address Offset:** 0x84
- **Read/write access:** read-only

This register is valid only when the DW_apb_uart is configured to have additional FIFO status registers implemented (FIFO_STAT = YES). If status registers are not implemented, this register does not exist and reading from this register address returns 0.



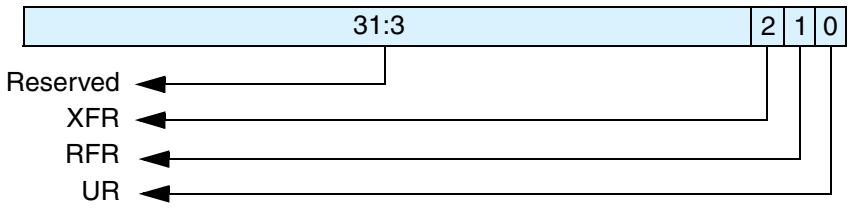
Bits	Name	R/W	Description
31:*FIFO_ADDR_WIDTH* + 1	Reserved and read as 0		
FIFO_ADDR_WIDTH:0	Receive FIFO Level	R	Receive FIFO Level. This indicates the number of data entries in the receive FIFO. **Reset Value:** 0x0

6.2.23 SRR

- **Name:** Software Reset Register
- **Size:** 32 bits
- **Address Offset:** 0x88
- **Read/write access:** write-only

This register is valid only when the DW_apb_uart is configured to have additional shadow registers implemented (SHADOW = YES). If shadow registers are not implemented, this register does not exist and reading from this register address returns 0.

For more information on the amount of time that serial clock modules need in order to see new register values and reset their respective state machines, refer to the "Clock Support" subsection.



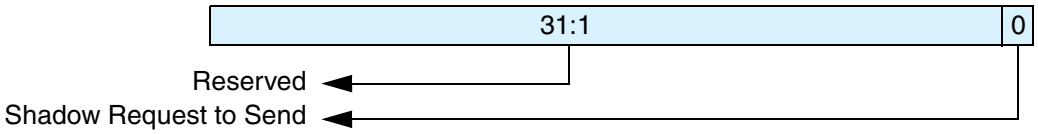
Bits	Name	R/W	Description
31:3	Reserved and read as 0		
2	XFR	W	XMIT FIFO Reset. This is a shadow register for the XMIT FIFO Reset bit (FCR[2]). This can be used to remove the burden on software having to store previously written FCR values (which are pretty static) just to reset the transmit FIFO. This resets the control portion of the transmit FIFO and treats the FIFO as empty. This also de-asserts the DMA TX request and single signals when additional DMA handshaking signals are selected (DMA_EXTRA = YES). Note that this bit is 'self-clearing'. It is not necessary to clear this bit. **Reset Value:** 0x0 **Dependencies:** Writes have no effect when FIFO_MODE = None.
1	RFR	W	RCVR FIFO Reset. This is a shadow register for the RCVR FIFO Reset bit (FCR[1]). This can be used to remove the burden on software having to store previously written FCR values (which are pretty static) just to reset the receive FIFO This resets the control portion of the receive FIFO and treats the FIFO as empty. This also de-asserts the DMA RX request and single signals when additional DMA handshaking signals are selected (DMA_EXTRA = YES). Note that this bit is 'self-clearing'. It is not necessary to clear this bit. **Reset Value:** 0x0 **Dependencies:** Writes have no effect when FIFO_MODE = None.

Bits	Name	R/W	Description
0	UR	W	UART Reset. This asynchronously resets the DW_apb_uart and synchronously removes the reset assertion. For a two clock implementation both pclk and sclk domains are reset. **Reset Value:** 0x0

6.2.24 SRTS

- **Name:** Shadow Request to Send
- **Size:** 32 bits
- **Address Offset:** 0x8C
- **Read/write access:** read/write

This register is valid only when the DW_apb_uart is configured to have additional shadow registers implemented (SHADOW = YES). If shadow registers are not implemented, this register does not exist and reading from this register address returns 0.

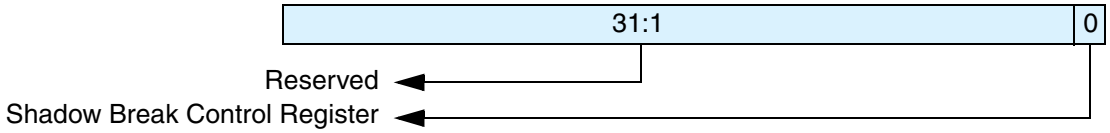


Bits	Name	R/W	Description
31:1	Reserved and read as 0		
0	Shadow Request to Send	R/W	Shadow Request to Send. This is a shadow register for the RTS bit (MCR[1]), this can be used to remove the burden of having to performing a read-modify-write on the MCR. This is used to directly control the Request to Send (rts_n) output. The Request To Send (rts_n) output is used to inform the modem or data set that the DW_apb_uart is ready to exchange data. When Auto RTS Flow Control is not enabled (MCR[5] = 0), the rts_n signal is set low by programming MCR[1] (RTS) to a high. In Auto Flow Control, AFCE_MODE = Enabled and active (MCR[5] = 1) and FIFOs enable (FCR[0] = 1), the rts_n output is controlled in the same way, but is also gated with the receiver FIFO threshold trigger (rts_n is inactive high when above the threshold) only when RTC Flow Trigger is disabled; otherwise it is gated by the receiver FIFO almost-full trigger, where “almost full” refers to two available slots in the FIFO (rts_n is inactive high when above the threshold). Note that in Loopback mode (MCR[4] = 1), the rts_n output is held inactive-high while the value of this location is internally looped back to an input. **Reset Value:** 0x0

6.2.25 SBCR

- **Name:** Shadow Break Control Register
- **Size:** 32 bits
- **Address Offset:** 0x90
- **Read/write access:** read/write

This register is valid only when the DW_apb_uart is configured to have additional shadow registers implemented (SHADOW = YES). If shadow registers are not implemented, this register does not exist and reading from this register address returns 0.



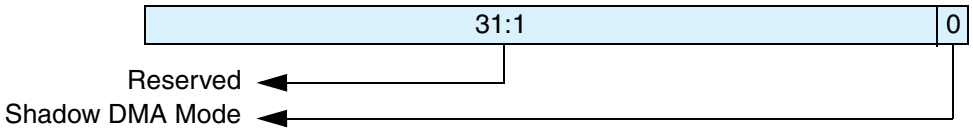
Bits	Name	R/W	Description
31:1	Reserved and read as 0		
0	Shadow Break Control Register	R/W	Shadow Break Control Bit. This is a shadow register for the Break bit (LCR[6]), this can be used to remove the burden of having to performing a read modify write on the LCR. This is used to cause a break condition to be transmitted to the receiving device. If set to 1, the serial output is forced to the spacing (logic 0) state. When not in Loopback Mode, as determined by MCR[4], the sout line is forced low until the Break bit is cleared. If SIR_MODE = Enabled and active (MCR[6] = 1) the sir_out_n line is continuously pulsed. When in Loopback Mode, the break condition is internally looped back to the receiver. **Reset Value:** 0x0

6.2.26 SDMAM

- **Name:** Shadow DMA Mode
- **Size:** 32 bits
- **Address Offset:** 0x94
- **Read/write access:** read/write

This register is valid only when the DW_apb_uart is configured to have additional FIFO registers implemented (FIFO_MODE != None) and additional shadow registers implemented (SHADOW = YES). If

these registers are not implemented, this register does not exist and reading from this register address returns 0.



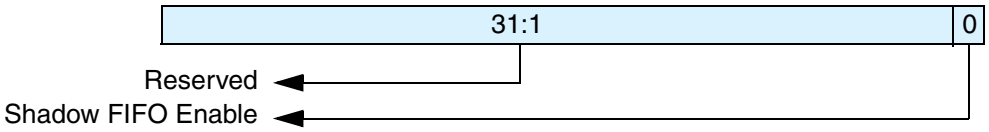
Bits	Name	R/W	Description
31:1	Reserved and read as 0		
0	Shadow DMA Mode	R/W	Shadow DMA Mode. This is a shadow register for the DMA mode bit (FCR[3]). This can be used to remove the burden of having to store the previously written value to the FCR in memory and having to mask this value so that only the DMA Mode bit gets updated. This determines the DMA signalling mode used for the dma_tx_req_n and dma_rx_req_n output signals when additional DMA handshaking signals are not selected (DMA_EXTRA = NO). ■ 0 – mode 0 ■ 1 – mode 1 **Reset Value:** 0x0

6.2.27 SFE

- **Name:** Shadow FIFO Enable
- **Size:** 32 bits
- **Address Offset:** 0x98
- **Read/write access:** read/write

This register is valid only when the DW_apb_uart is configured to have additional FIFO registers implemented (FIFO_MODE != None) and additional shadow registers implemented (SHADOW = YES). If

these registers are not implemented, this register does not exist and reading from this register address returns 0.



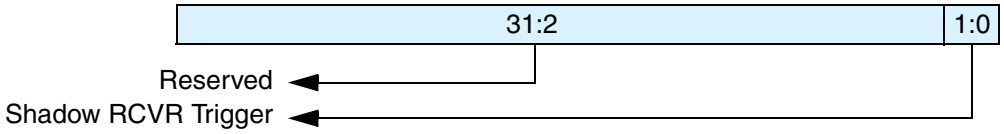
Bits	Name	R/W	Description
31:1	Reserved and read as 0		
0	Shadow FIFO Enable	R/W	Shadow FIFO Enable. This is a shadow register for the FIFO enable bit (FCR[0]). This can be used to remove the burden of having to store the previously written value to the FCR in memory and having to mask this value so that only the FIFO enable bit gets updated.This enables/disables the transmit (XMIT) and receive (RCVR) FIFOs. If this bit is set to 0 (disabled) after being enabled then both the XMIT and RCVR controller portion of FIFOs are reset. **Reset Value:** 0x0

6.2.28 SRT

- **Name:** Shadow RCVR Trigger
- **Size:** 32 bits
- **Address Offset:** 0x9C
- **Read/write access:** read/write

This register is valid only when the DW_apb_uart is configured to have additional FIFO registers implemented (FIFO_MODE != None) and additional shadow registers implemented (SHADOW = YES). If

these registers are not implemented, this register does not exist and reading from this register address returns 0.



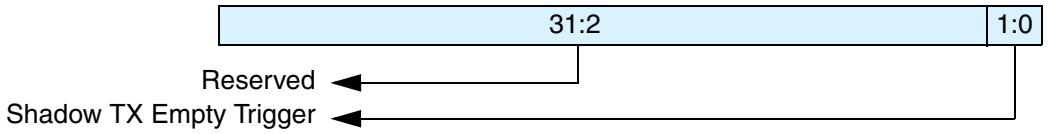
Bits	Name	R/W	Description
31:2	Reserved and read as 0		
1:0	Shadow RCVR Trigger	R/W	Shadow RCVR Trigger. This is a shadow register for the RCVR trigger bits (FCR[7:6]). This can be used to remove the burden of having to store the previously written value to the FCR in memory and having to mask this value so that only the RCVR trigger bit gets updated. This is used to select the trigger level in the receiver FIFO at which the Received Data Available Interrupt is generated. It also determines when the dma_rx_req_n signal is asserted when DMA Mode (FCR[3]) = 1. The following trigger levels are supported: ■ 00 – 1 character in the FIFO ■ 01 – FIFO ¼ full ■ 10 – FIFO ½ full ■ 11 – FIFO 2 less than full **Reset Value:** 0x0

6.2.29 STET

- **Name:** Shadow TX Empty Trigger
- **Size:** 32 bits
- **Address Offset:** 0xA0
- **Read/write access:** read/write

This register is valid only when the DW_apb_uart is configured to have FIFOs implemented (FIFO_MODE != NONE) and THRE interrupt support implemented (THRE_MODE_USER = Enabled) and additional shadow registers implemented (SHADOW = YES). If FIFOs are not implemented or THRE

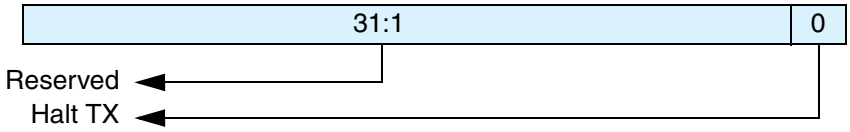
interrupt support is not implemented or shadow registers are not implemented, this register does not exist and reading from this register address returns 0.



Bits	Name	R/W	Description
31:2	Reserved and read as 0		
1:0	Shadow TX Empty Trigger	R/W	Shadow TX Empty Trigger. This is a shadow register for the TX empty trigger bits (FCR[5:4]). This can be used to remove the burden of having to store the previously written value to the FCR in memory and having to mask this value so that only the TX empty trigger bit gets updated. This is used to select the empty threshold level at which the THRE Interrupts are generated when the mode is active. The following trigger levels are supported: ■ 00 – FIFO empty ■ 01 – 2 characters in the FIFO ■ 10 – FIFO ¼ full ■ 11 – FIFO ½ full **Reset Value:** 0x0 **Dependencies:** Writes have no effect when THRE_MODE_USER = Disabled.

6.2.30 HTX

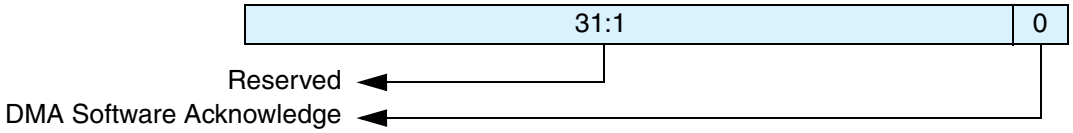
- **Name:** Halt TX
- **Size:** 32 bits
- **Address Offset:** 0xA4
- **Read/write access:** read/write



Bits	Name	R/W	Description
31:1	Reserved and read as 0		
0	Halt TX	R/W	This register is use to halt transmissions for testing, so that the transmit FIFO can be filled by the master when FIFOs are implemented and enabled. ■ 0 – Halt TX disabled ■ 1 – Halt TX enabled Note, if FIFOs are implemented and not enabled, the setting of the halt TX register has no effect on operation. **Reset Value:** 0x0 **Dependencies:** Writes have no effect when FIFO_MODE = None.

6.2.31 DMASA

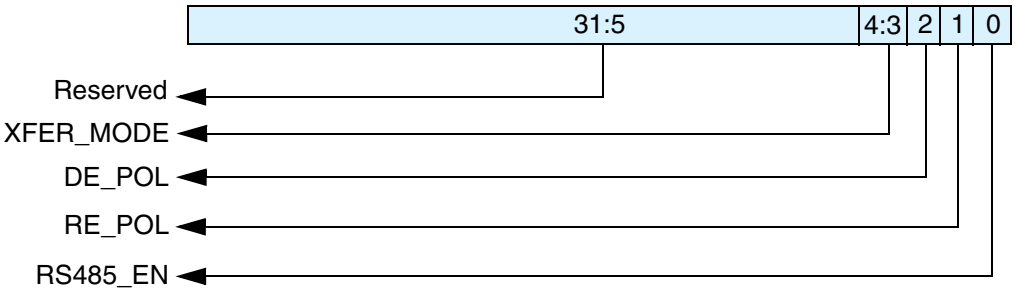
- **Name:** DMA Software Acknowledge
- **Size:** 32 bits
- **Address Offset:** 0xA8
- **Read/write access:** write



Bits	Name	R/W	Description
31:1	Reserved and read as 0		
0	DMA Software Acknowledge	W	This register is use to perform a DMA software acknowledge if a transfer needs to be terminated due to an error condition. For example, if the DMA disables the channel, then the DW_apb_uart should clear its request. This causes the TX request, TX single, RX request and RX single signals to de-assert. Note that this bit is 'self-clearing'. It is not necessary to clear this bit. **Reset Value:** 0x0 **Dependencies:** Writes have no effect when DMA_EXTRA = No.

6.2.32 TCR

- **Name:** Transceiver Control Register (TCR)
- **Size:** 32 bits
- **Address Offset:** 0xAC
- **Read/write access:** read/write



This register is used to enable or disable RS485 mode and also control the polarity values for Driven enable (de) and Receiver Enable (re) signals.

This register is only valid when the DW_apb_uart is configured to have RS485 interface implemented (UART_RS485_INTERFACE_EN = ENABLED). If RS485 interface is not implemented, this register does not exist and reading from this register address returns zero.

Bits	Name	R/W	Description
31:5	Reserved and read as 0		
4:3	XFER_MODE	R/W	Transfer Mode ■ 0 : In this mode, transmit and receive can happen simultaneously. You can enable DE_EN and RE_EN at any point of time. Turn around timing as programmed in the TAT register is not applicable in this mode. ■ 1 : In this mode, DE and RE are mutually exclusive. The hardware considers the turnaround timings that are programmed in the TAT register while switching from RE to DE or from DE to RE. Ensure that either DE or RE is expected to be enabled while programming. For transmission, hardware waits if it is in the midst of receiving any transfer, before it starts transmitting. ■ 2 : In this mode, DE and RE are mutually exclusive. Once DE_EN or RE_EN is programed, 're' is enabled by default and DW_apb_uart controller will be ready to receive. If the user programs the TX FIFO with data, then DW_apb_uart, after ensuring no receive is in progress, disables the 're' and enables the 'de' signal. Once the TX FIFO becomes empty, the 're' signal gets enabled and the 'de' signal will be disabled. In this mode of operation, the hardware considers the turnaround timings that are programmed in the TAT register while switching from RE to DE or from DE to RE. In this mode, 'de' and 're' signals are strictly complementary to each other. **Reset Value**: 0x0
2	DE_POL	R/W	Driver Enable Polarity ■ 1: DE signal is active high ■ 0: DE signal is active low **Reset Value**: UART_DE_POL
1	RE_POL	R/W	Receiver Enable Polarity ■ 1: RE signal is active high ■ 0: RE signal is active low **Reset Value**: UART_RE_POL

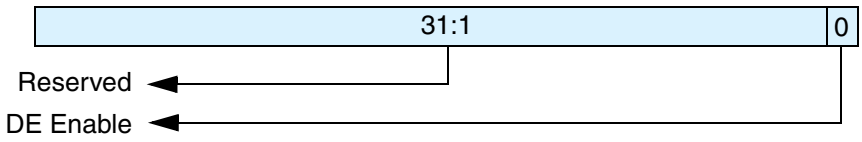
Bits	Name	R/W	Description
0	RS485_EN	R/W	RS485 Transfer Enable ■ 0 : In this mode, the transfers are still in the RS232 mode. All other fields in this register are reserved and registers DE_EN, RE_EN, DET and TAT are reserved. ■ 1: In this mode, the transfers will happen in RS485 mode. All other fields of this register are applicable. **Reset Value**: 0x0

6.2.33 DE_EN

- **Name:** Driver Output Enable Register (DE_EN)
- **Size:** 32 bits
- **Address Offset:** 0xB0
- **Read/write access:** read/write

The Driver Output Enable Register (DE_EN) is used to control the assertion and de-assertion of the DE signal.

This register is only valid when the DW_apb_uart is configured to have RS485 interface implemented (UART_RS485_INTERFACE_EN = ENABLED). If RS485 interface is not implemented, this register does not exist and reading from this register address will return zero.



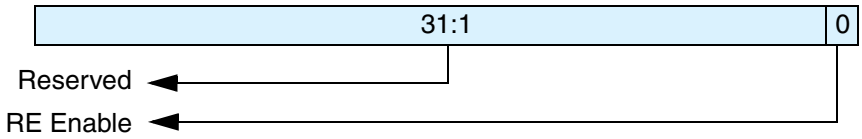
Bits	Name	R/W	Description
31:1	Reserved and read as 0		
0	DE Enable	R/W	DE Enable control The 'DE Enable' register bit is used to control assertion and de-assertion of 'de' signal. ■ 0: De-assert 'de' signal ■ 1: Assert 'de' signal **Reset Value**: 0x0

6.2.34 RE_EN

- **Name:** Receiver Output Enable Register (RE_EN)
- **Size:** 32 bits
- **Address Offset:** 0xB4
- **Read/write access:** read/write

The Receiver Output Enable Register (RE_EN) is used to control the assertion and de-assertion of the RE signal.

This register is only valid when the DW_apb_uart is configured to have RS485 interface implemented (UART_RS485_INTERFACE_EN = ENABLED). If the RS485 interface is not implemented, this register does not exist and reading from this register address will return zero.



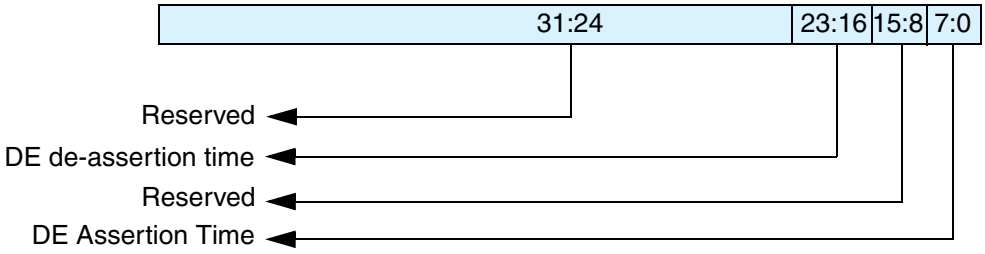
Bits	Name	R/W	Description
31:1	Reserved and read as 0		
0	RE Enable	R/W	RE Enable control The 'RE Enable' register bit is used to control assertion and de-assertion of 're' signal. ■ 0: De-assert 're' signal ■ 1: Assert 're' signal **Reset Value**: 0x0

6.2.35 DET

- **Name:** Driver Output Enable Timing Register (DET)
- **Size:** 32 bits
- **Address Offset:** 0xB8
- **Read/write access:** read/write

The Driver Output Enable Timing Register (DET) is used to control the DE assertion and de-assertion timings of 'de' signal.

This register is only valid when the DW_apb_uart is configured to have RS485 interface implemented (UART_RS485_INTERFACE = ENABLED). If RS485 interface is not implemented, this register does not exist and reading from this register address will return zero.



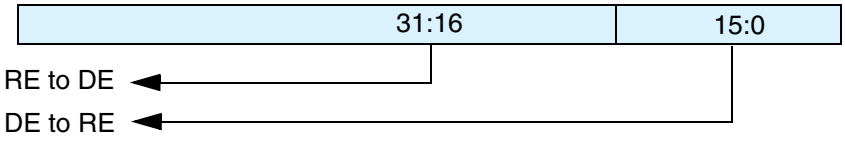
Bits	Name	R/W	Description
31:24	Reserved and read as 0		
23:16	DE de-assertion time	R/W	Driver enable de-assertion time. This field controls the amount of time (in terms of number of serial clock periods) between the end of stop bit on the serial output (sout) to the falling edge of Driver output enable signal. **Reset Value**: 0x0
15:8	Reserved and read as 0		
7:0	DE assertion time	R/W	Driver enable assertion time. This field controls the amount of time (in terms of number of serial clock periods) between the assertion of rising edge of Driver output enable signal to serial transmit enable. Any data in transmit buffer, will start on serial output (sout) after the transmit enable. **Reset Value**: 0x0

6.2.36 TAT

- **Name:** TurnAround Timing Register (TAT)
- **Size:** 32 bits
- **Address Offset:** 0xBC
- **Read/write access:** read/write

The TurnAround Timing Register (TAT) is used to hold the turnaround time between switching of 're' and 'de' signals.

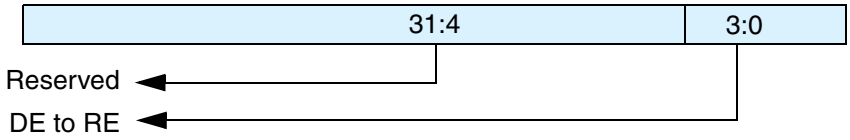
This register is only valid when the DW_apb_uart is configured to have the RS485 interface implemented (UART_RS485_INTERFACE_EN = ENABLED). If RS485 interface is not implemented, this register does not exist and reading from this register address will return zero.



Bits	Name	R/W	Description
31:16	RE to DE	R/W	Receiver Enable to Driver Enable TurnAround time. Turnaround time (in terms of serial clock) for RE de-assertion to DE assertion. **Note**: ■ If the DE assertion time in the DET register is 0, then the actual value is the programmed value + 3. ■ If the DE assertion time in the DET register is 1, then the actual value is the programmed value + 2. ■ If the DE assertion time in the DET register is greater than 1, then the actual value is the programmed value + 1. **Reset Value**: 0x0
15:0	DE to RE	R/W	Driver Enable to Receiver Enable TurnAround time. Turnaround time (in terms of serial clock) for DE de-assertion to RE assertion. **Note**: The actual time is the programmed value + 1. **Reset Value**: 0x0

6.2.37 DLF

- **Name:** Divisor Latch Fraction Register (DLF)
- **Size:** 32 bits
- **Address Offset:** 0xC0
- **Read/write access:** read/write



This register is only valid when the DW_apb_uart is configured to have Fractional Baud rate Divisor implemented (FRACTIONAL_BAUD_DIVISOR_EN = ENABLED). If Fractional Baud rate divisor is not implemented, this register does not exist and reading from this register address will return zero.

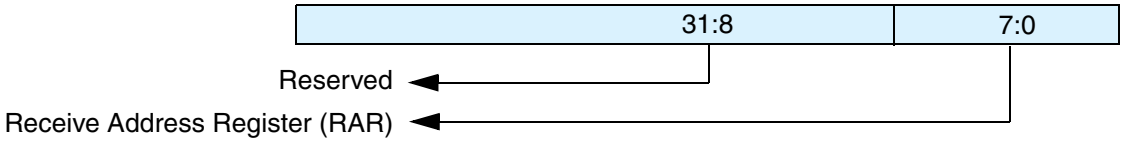
Bits	Name	R/W	Description
31:DLF_SIZE	Reserved and read as zero		
DLF_SIZE-1:0	DLF	R/W	Fractional part of divisor. The fractional value is added to integer value set by DLH, DLL. Fractional value is determined by (Divisor Fraction value)/(2^DLF_SIZE). Table 6-3 describes the DLF Values to be programmed for DLF_SIZE=4. **Reset Value**: 0x0

Table 6-3 Divisor Latch Fractional Values

DLF Value	Fraction	Fractional Value
0000	0/16	0.0000
0001	1/16	0.0625
0010	2/16	0.125
0011	3/16	0.1875
0100	4/16	0.25
0101	5/16	0.3125
0110	6/16	0.375
0111	7/16	0.4375
1000	8/16	0.5
1001	9/16	0.5625
1010	10/16	0.625
1011	11/16	0.6875
1100	12/16	0.75
1101	13/16	0.8125
1110	14/16	0.875
1111	15/16	0.9375

6.2.38 RAR

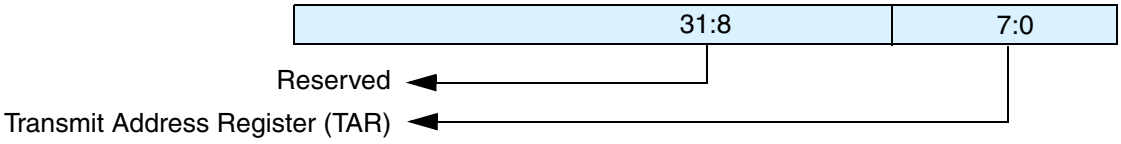
- **Name:** Receive Address Register (RAR)
- **Size:** 32 bits
- **Address Offset:** 0xC4
- **Read/write access:** read/write



Bits	Name	R/W	Description
31:8	Reserved and read as 0		
7:0	RAR	R/W	This is an address matching register during receive mode. If the 9-th bit is set in the incoming character then the remaining 8-bits will be checked against this register value. If the match happens then sub-sequent characters with 9-th bit set to 0 will be treated as data byte until the next address byte is received. **Note**: ■ This register is applicable only when 'ADDR_MATCH' (LCR_EXT[1]) and 'DLS_E' (LCR_EXT[0]) bits are set to 1. ■ If UART_16550_COMPATIBLE is configured to 0, then RAR should be programmed only when UART is not busy. ■ If UART_16550_COMPATIBLE is configured to 0, then RAR can be programmed at any point of the time. However, user must not change this register value when any receive is in progress. **Reset Value**: 0x0

6.2.39 TAR

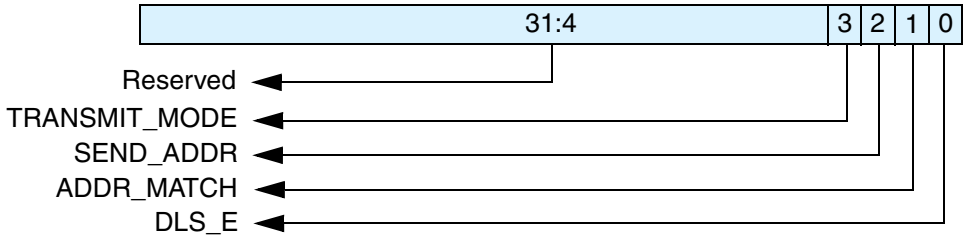
- **Name:** Transmit Address Register (TAR)
- **Size:** 32 bits
- **Address Offset:** 0xC8
- **Read/write access:** read/write



Bits	Name	R/W	Description
31:8	Reserved and read as 0		
7:0	TAR	R/W	This is an address matching register during transmit mode. If DLS_E (LCR_EXT[0]) bit is enabled, then DW_apb_uart sends the 9-bit character with 9-th bit set to 1 and remaining 8-bit address will be sent from this register provided 'SEND_ADDR' (LCR_EXT[2]) bit is set to 1. **NOTE**: ■ This register is used only to send the address. The normal data should be sent by programming THR register. ■ Once the address is started to send on the DW_apb_uart serial lane, then 'SEND_ADDR' bit will be auto-cleared by the hardware. **Reset Value**: 0x0

6.2.40 LCR_EXT

- **Name:** Line Extended Control Register (LCR_EXT)
- **Size:** 32 bits
- **Address Offset:** 0xCC
- **Read/write access:** read/write



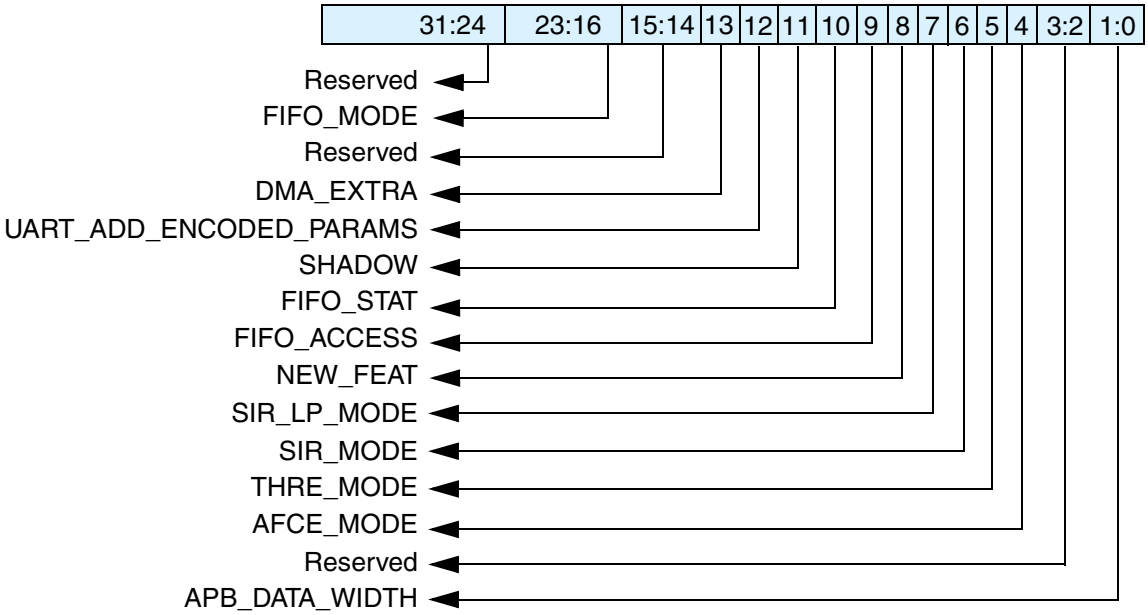
Bits	Name	R/W	Description
31:4	Reserved and read as 0		
3	TRANSMIT_MODE	R/W	Transmit mode control bit. This bit is used to control the type of transmit mode during 9-bit data transfers. ■ 1 : In this mode of operation, Transmit Holding Register (THR) and Shadow Transmit Holding Register (STHR) are 9-bit wide. You must ensure that the THR/STHR register is written correctly for address/data. Address: 9^{th} bit is set to 1, Data: 9^{th} bit is set to 0. **NOTE**: Transmit address register (TAR) is not applicable in this mode of operation. ■ 0 : In this mode of operation, Transmit Holding Register (THR) and Shadow Transmit Holding register (STHR) are 8-bit wide. The user needs to program the address into Transmit Address Register (TAR) and data into the THR/STHR register. SEND_ADDR bit is used as a control knob to indicate the DW_apb_uart on when to send the address. **Reset Value**: 0x0
2	SEND_ADDR	R/W	Send address control bit. This bit is used as a control knob for the user to determine when to send the address during transmit mode. ■ 1 - 9-bit character will be transmitted with 9-th bit set to 1 and the remaining 8-bits will match to what is being programmed in "Transmit Address Register". ■ 0 - 9-bit character will be transmitted with 9-th bit set to 0 and the remaining 8-bits will be taken from the TxFIFO which is programmed through 8-bit wide THR/STHR register. **NOTE**: 1. This bit is auto-cleared by the hardware, after sending out the address character. User is not expected to program this bit to 0. 2. This field is applicable only when DLS_E bit is set to 1 and TRANSMIT_MODE is set to 0. **Reset Value**: 0x0
1	ADDR_MATCH	R/W	Address Match Mode. This bit is used to enable the address match feature during receive. ■ 1 - Address match mode; DW_apb_uart will wait until the incoming character with 9-th bit set to 1. And, further checks to see if the address matches with what is programmed in "Receive Address Match Register". If match is found, then sub-sequent characters will be treated as valid data and DW_apb_uart starts receiving data. ■ 0 - Normal mode; DW_apb_uart will start to receive the data and 9-bit character will be formed and written into the receive RxFIFO. User is responsible to read the data and differentiate b/n address and data. **NOTE**: This field is applicable only when DLS_E is set to 1.

Bits	Name	R/W	Description
0	DLS_E	R/W	Extension for DLS. This bit is used to enable 9-bit data for transmit and receive transfers. ■ 1 = 9 bits per character ■ 0 = Number of data bits selected by DLS **Reset Value**: 0x0

6.2.41 CPR

- **Name:** Component Parameter Register
- **Size:** 32 bits
- **Address Offset:** 0xF4
- **Read/write access:** read-only

This register is valid only when UART_ADD_ENCODED_PARAMS = 1. If the UART_ADD_ENCODED_PARAMS parameter is not set, this register does not exist and reading from this register address returns 0.



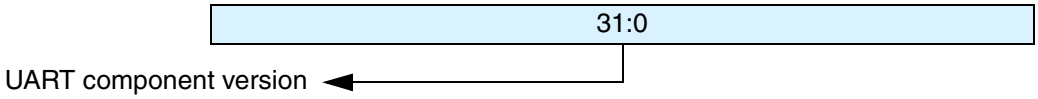
Bits	Name	R/W	Description
31:24	Reserved and read as 0		
23:16	FIFO_MODE	R	0x00 = 0 0x01 = 16 0x02 = 32 to 0x80 = 2048 0x81- 0xff = reserved
15:14	Reserved and read as 0		
13	DMA_EXTRA	R	0 – FALSE 1 – TRUE
12	UART_ADD_ENCODED_PARAMS	R	0 – FALSE 1 – TRUE

Bits	Name	R/W	Description
11	SHADOW	R	0 – FALSE 1 – TRUE
10	FIFO_STAT	R	0 – FALSE 1 – TRUE
9	FIFO_ACCESS	R	0 – FALSE 1 – TRUE
8	ADDITIONAL_FEAT	R	0 – FALSE 1 – TRUE
7	SIR_LP_MODE	R	0 – FALSE 1 – TRUE
6	SIR_MODE	R	0 – FALSE 1 – TRUE
5	THRE_MODE	R	0 – FALSE 1 – TRUE
4	AFCE_MODE	R	0 – FALSE 1 – TRUE
3:2	Reserved and read as 0		
1:0	APB_DATA_WIDTH	R	00 – 8 bits 01 – 16 bits 10 – 32 bits 11 – reserved

6.2.42 UCV

- **Name:** UART Component Version
- **Size:** 32 bits
- **Address Offset:** 0xF8
- **Read/write access:** read-only

This register is valid only when the DW_apb_uart is configured to have additional features implemented (ADDITIONAL_FEATURES = YES). If additional features are not implemented, this register does not exist and reading from this register address returns 0.

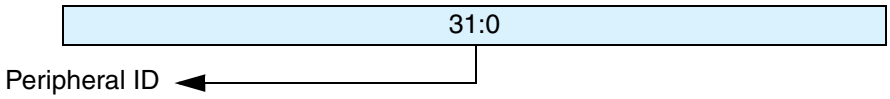


Bits	Name	R/W	Description
31:0	UART Component Version	R	ASCII value for each number in the version, followed by *. For example 32_30_31_2A represents the version 2.01* **Reset Value:** See the releases table in the AMBA 2 release notes.

6.2.43 CTR

- **Name:** Component Type Register
- **Size:** 32 bits
- **Address Offset:** 0xFC
- **Read/write access:** read-only

This register is valid only when the DW_apb_uart is configured to have additional features implemented (ADDITIONAL_FEATURES = YES). If additional features are not implemented, this register does not exist and reading from this register address returns 0.



Bits	Name	R/W	Description
31:0	Peripheral ID	R	This register contains the peripherals identification code. **Reset Value:** 0x44570110

7 Internal Parameter Descriptions

Provides a description of the internal parameters that might be indirectly referenced in expressions in the Signals, Parameters, or Registers chapters. These parameters are not visible in the coreConsultant GUI and most of them are derived automatically from visible parameters. **You must not set any of these parameters directly.**

Some expressions might refer to TCL functions or procedures (sometimes identified as **function_of**) that coreConsultant uses to make calculations. The exact formula used by these TCL functions is not provided in this chapter. However, when you configure the core in coreConsultant, all TCL functions and parameters are evaluated completely; and the resulting values are displayed where appropriate in the coreConsultant GUI reports.

Table 7-1 Internal Parameters

Parameter Name	Equals To
FIFO_ADDR_WIDTH	{[function_of: FIFO_MODE]}
MEM_SELECT	(((FIFO_MODE ! = 0) && (FIFO_MODE <= 256)) ? MEM_SELECT_USER : 0)
RX_RAM_DATA_WIDTH	RXFIFO_RW
TX_RAM_DATA_WIDTH	TXFIFO_RW
UART_ADDR_SLICE_LHS	8
blank	intentionally left blank

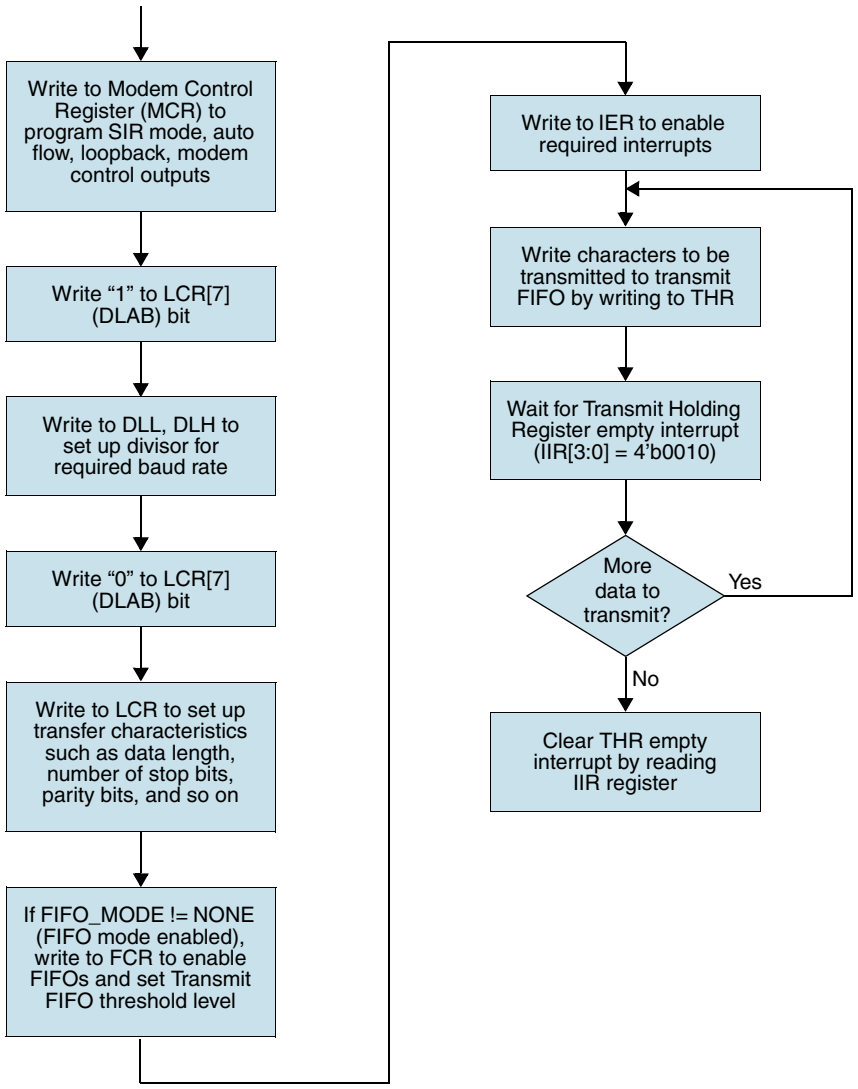
8 Programming the DW_apb_uart

The following topics provide information necessary to program the DW_apb_uart.

8.1 Programing Examples

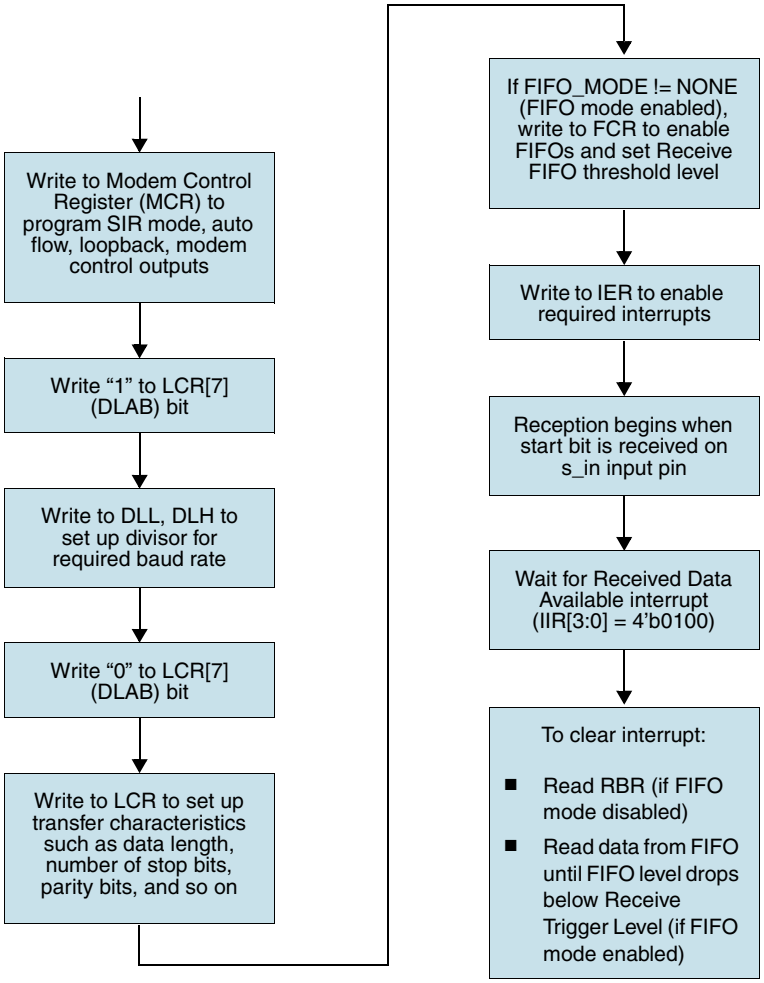
The flow diagram in Figure 8-1 on page 178 shows the programming sequence for setting up the DW_apb_uart for transmission.

Figure 8-1 Flowchart for DW_apb_uart Transmit Programming Example



The flow diagram in Figure 8-2 shows the programming sequence for setting up the DW_apb_uart for reception.

Figure 8-2 Flowchart for DW_apb_uart Receive Programming Example



8.2 Programming Flow in RS485 Mode

8.2.1 Full Duplex Mode (XFER_MODE=0)

- Program the TCR register to set the XFER_MODE(0), DE_POL (polarity of 'de'signal) and RE_POL (polarity of 're' signals).
- Program the DE assertion and de-assertion timing in the DET register.
- Program RE_EN and DE_EN register to assert 're' signal and 'de' signal, respectively.
- Perform the data transmission and reception.
- Program RE_EN to de-assert 're' signal.
- 'The de' signal gets de-asserted based on TxFIFO empty. Program DE_EN to '0' if you do not want to transmit further.

8.2.2 Software-Enabled Half Duplex Mode (XFER_MODE=1)

- Program the TCR register to set the XFER_MODE (1), DE_POL (polarity of 'de' signal) and RE_POL (polarity of 're' signals).
- Program the DE assertion and de-assertion timing in the DET register.
- Program the turnaround times in TAT register.
- Program RE_EN and DE_EN registers to assert 're' signal and 'de' signal, respectively.
- Perform the data transmission/receive.
- Program the RE_EN register to de-assert 're' signal.
- Program the DE_EN register to '0', before programming RE_EN to '1'.
  - 'de' signal de-assertion is based on TxFIFO empty condition and will be taken care by the Hardware. You need to program DE_EN to '0' only in the situation where you want to change the mode.

8.2.3 Hardware enabled Half Duplex mode (XFER_MODE=2)

- Program the TCR register to set the XFER_MODE (2), DE_POL (polarity of 'de' signal) and RE_POL (polarity of 're' signals).
- Program the DE assertion and de-assertion timing in the DET register.
- Program the turnaround times in TAT register.
- Program RE_EN and DE_EN register to enable the transmit and receive paths.
- Perform the Data transmission / receive.
- Once the 'RE_EN' and 'DE_EN' is programmed to '1', then by default 're' signal will be asserted and 'de' will be de-asserted. When the software pushes the data into the TX FIFO and if there is no ongoing receive transfer, then the 're' signal will get de-asserted and then the 'de' signal gets asserted until the TX FIFO has data to be transmitted.
- RE_EN and DE_EN will still serve as the software overrides to decide when to shutdown transmit and receive paths.

Note

- Irrespective of other configurations (parameters enabled), RS485 mode is not applicable when loopback mode is enabled. However, external signal rs485_en is still asserted when TCR[0] (RS485 EN) bit is asserted even in loopback mode.
- When Clock Gating is enabled (CLK_GATE_EN=1), the clock gate enable signal(s) uart_lp_req_pclk for single clock implementations or uart_lp_req_pclk and uart_lp_req_sclk for two clock implementations is used to indicate the following:
  - Transmit and receive pipelines are clear (no data).
  - No activity has occurred.
  - Modem control input signals have not changed in more than one character time—the time taken to TX/RX a character—so that clocks can be gated.

  A character is made up of:

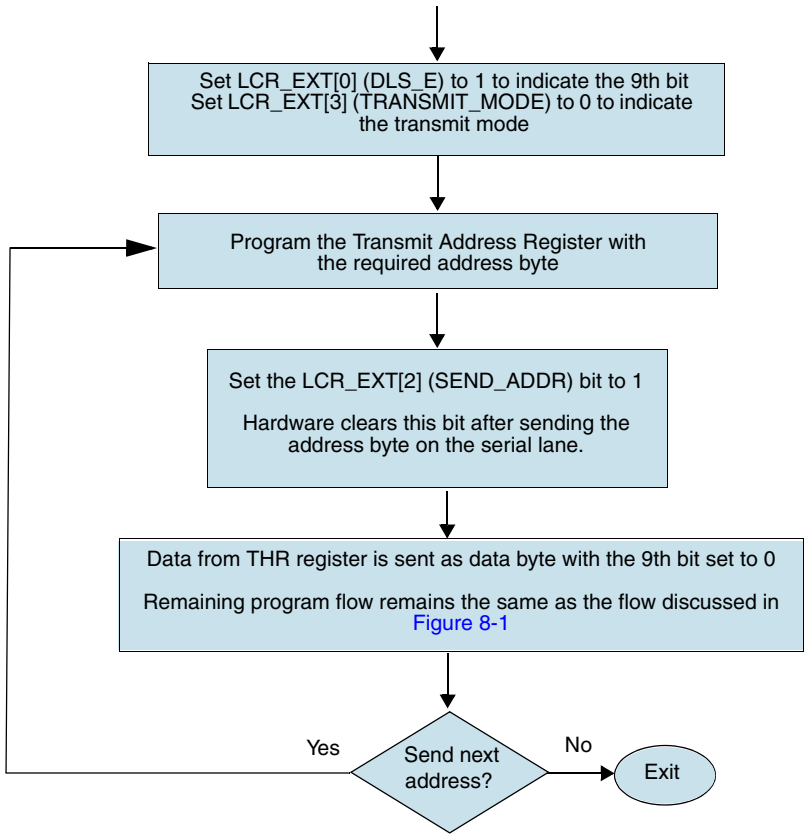
  start_bit + data_bits + parity (optional) + stop_bit(s))

  If UART_RS485_INTERFCAE_EN=1 and rs485_en (TCR[0]) =1, then DW_apb_uart will also ensure that counters related to DET/TAT are also taken care of before entering into low power mode.

8.3 Programming Flow in 9-bit Data Mode

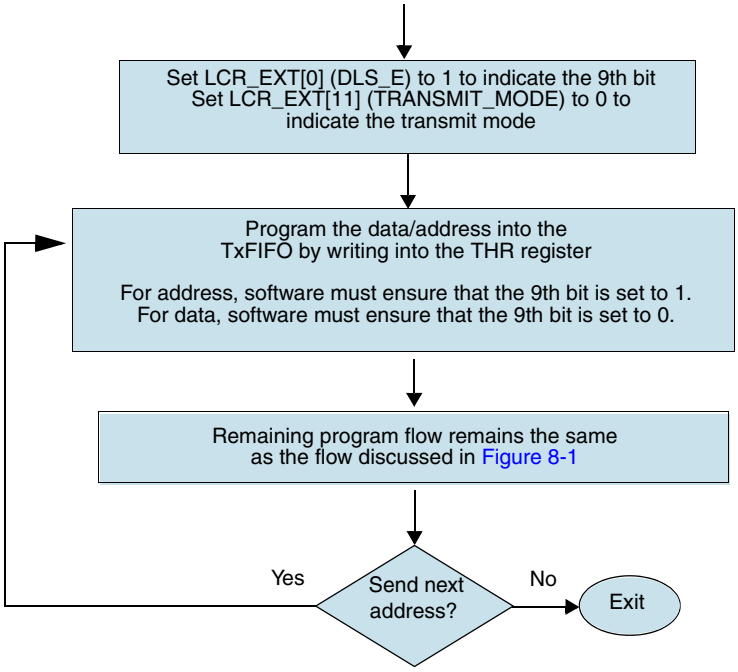
8.3.1 Transmit Mode 0

Figure 8-3 Auto Address Transmit Mode



8.3.2 Transmit Mode 1

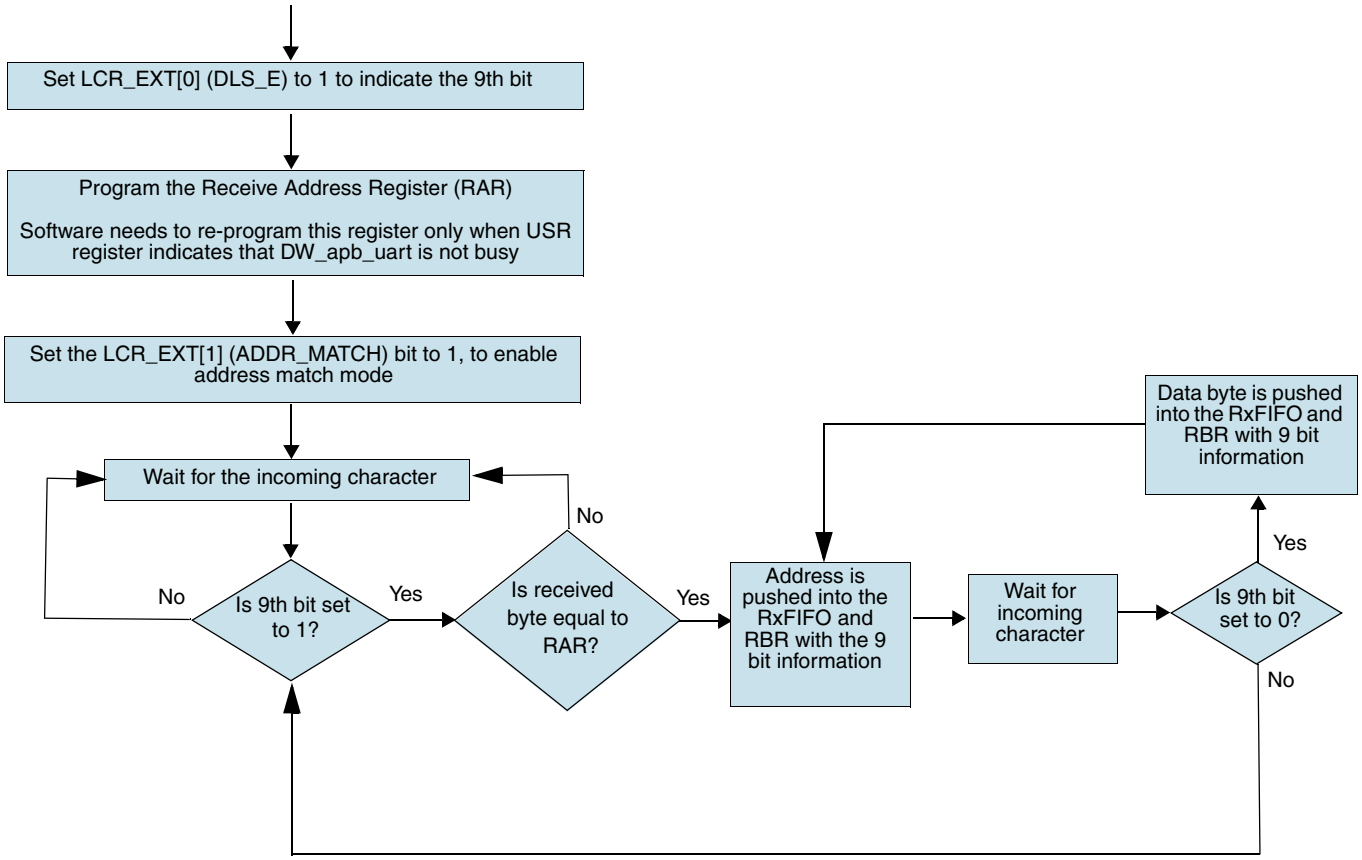
Figure 8-4 Normal Transmit Mode Programming Flow



8.3.3 Hardware Address Match Receive mode

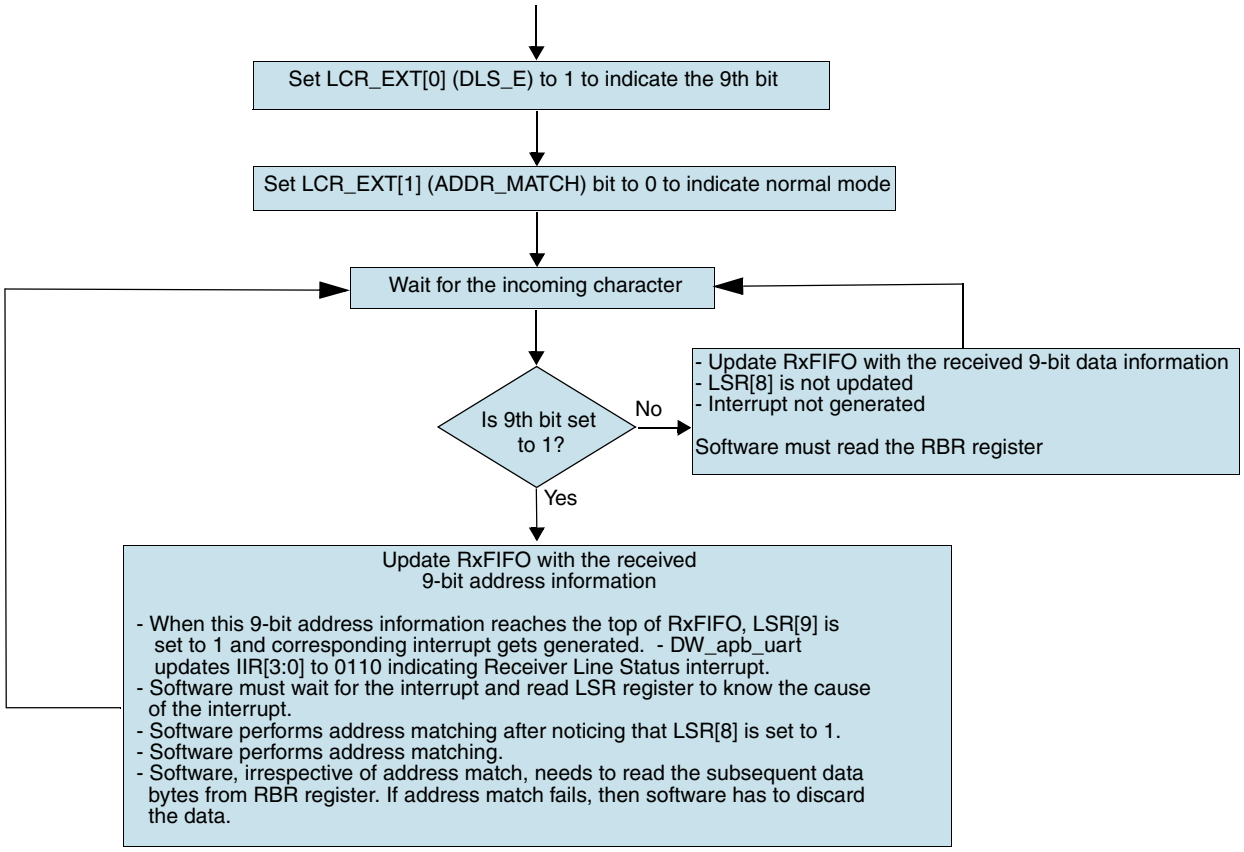
For setting up DW_apb_uart in hardware address match receive mode, follow the programming sequence mentioned in Figure 8-2 and additional steps as mentioned in Figure 8-5.

Figure 8-5 Hardware Address Match Receive Programming Flow



8.3.4 Software Address Match Receive mode

Figure 8-6 Software Address Match Receive Programming Mode



Note In the FIFO mode, if UART_9BIT_DATA_EN=1, the character timeout also considers the 9^{th} bit.

8.4 Programming Flow for Fractional Baud Rate

The programming flow for fractional baud rate is the same as explained in "Programing Examples" on page 177 except that you must configure the DLF register before programming the DLH and DLL registers if the FRACTIONAL_BAUD_DIVISOR_EN parameter is enabled.

8.5 Software Drivers

The family of DesignWare Synthesizable Components includes a Driver Kit for the DW_apb_uart component. This low-level driver allows you to program a DW_apb_uart component and integrate your code into a larger software system. The Driver Kit provides the following benefits to IP designers:

- Proven method of access to DW_apb_uart minimizing usage errors
- Rapid software development with minimum overhead
- Detailed knowledge of DW_apb_uart register bit fields not required
- Easy integration of DW_apb_uart into existing software system
- Programming at register level eliminated

You must purchase a source code license (DWC-APB-Periph-Source) to use the DW_apb_uart Driver Kit. However, you can access some Driver Kit files and documentation in $DESIGNWARE_HOME/drivers/DW_apb_uart/latest. For more information about the Driver Kit, refer to the *DW_apb_uart Driver Kit User Guide*. For more information about purchasing the source code license and obtaining a download of the Driver Kit, contact Synopsys at designware@synopsys.com for details.

9 Verification

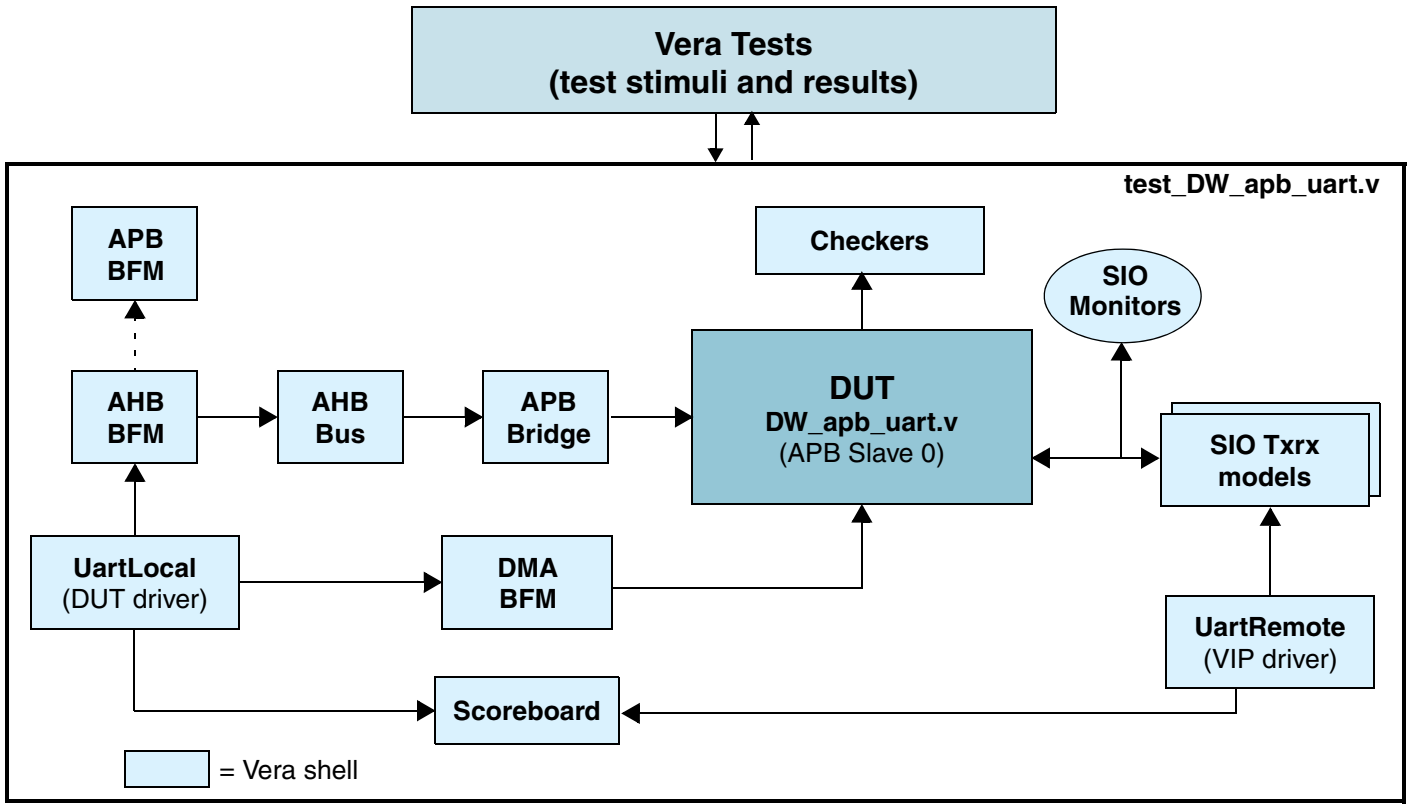
This chapter provides an overview of the testbench and tests available for DW_apb_uart verification. (Also see "Verification Environment Overview" on page 22). Once the DW_apb_uart has been configured and the verification environment set up, simulations can be automatically ran.

Note The DW_apb_uart verification testbench is built with DesignWare Verification IP (VIP). Please make sure you have the supported version of the VIP components for this release, otherwise, you may experience some tool compatibility problems. For more information about supported tools in this release, refer to the following web page:
www.synopsys.com/products/designware/docs/doc/amba/latest/dw_amba_install.pdf

9.1 Overview of DW_apb_uart Testbench

As illustrated in Figure 9-1, the DW_apb_uart Verilog testbench includes an instantiation of the design under test (DUT), AHB and APB bus models, and a Vera shell.

Figure 9-1 DW_apb_uart Testbench



The DW_apb_uart testbench consists of the following:

- Vera Test – Responsible for enumerating the test conditions under which the DUT (UART) is verified. These conditions steer the simulations in various aspects, such as the register settings of the UART, the transfer direction (UART to SIO_TxRx, SIO_TxRx to UART, loopback) and length (number of characters serially exchanged), number of iterations for a single test scenario, simulation controls, and so on. All this information is randomly created and encapsulated in several classes with associated Vera randomization and constraint constructs. This information is also relayed to the other Vera components.
- Testbench API – Takes in the randomized test conditions and uses the relevant portions for appropriate directing of the simulation controls, such as the number of iterations executed. It is also responsible for ensuring that all test monitors are alerted and set up for the indicated test type, as well as relaying information (in the form of class objects) to the two drivers (UartLocalClass, UartRemoteClass) in order to execute the desired simulation behavior to effect; for example, transfers to and from the DUT.
- DUT Driver, or UartLocalClass – Responsible for translating the information provided by the Testbench API into the desired simulation behaviors. This Vera component ensures that corresponding command and/or sequence of commands are issued to the AHB BFM to effect the

desired register settings, transferring of data, toggling of the modem interface signals, loopback mode, interrupts, and so on in the DUT(UART). Since the information directing the required simulations are shielded by UartLocalClass away from AHB BFM, revised versions of the latter Vera component can be easily accommodated by updating UartLocalClass.

- VIP Driver, or UartRemoteClass – Performs a similar role to that of UartLocalClass, translating the information provided by Testbench API into corresponding SIO_TxRx BFM commands in order to effect the desired simulation behaviors. Note that controls complementary to that of the UartLocalClass are performed in the UartRemoteClass, such that if the DUT performs transmits, then the SIO_TxRx BFM attempts reception(s). UartRemoteClass also serves to shield the rest of the verification environment from revised versions of this VIP component.
- AHB BFM – VIP harness BFM required to imitate as an AHB master. All actual register accesses (reads and writes) required by a current test are performed using AHB BFM commands. Existing class definitions for this BFM are re-used.
- DMA BFM – Exercises the DMA interface of the DUT/UARTv3.0. It behaves as another AHB master, issuing commands to perform reads and writes from/to the UART. These activities are coordinated within the UartLocalClass.
- Checkers – Examine the behavior of the DUT through the DUT signal interfaces, and evaluate the outcome of the prescribed tests targeted at the DUT. The verification tests determine the degree to which the DUT is verified, and is therefore linked to one (or more) test monitors in the test environment. These Checkers operate independently of the main flow in the test code. This form of messaging uses two classes, TestmonAlertClass and TestmonExecuteClass.
- SIOMonitor – Serial monitor VIP from the SIO VIP package. When appropriately parameterized, the SIO_Mon examines the serial bit patterns exchanged between the DUT and the SIO_TxRx.
- SIOTxRx BFM – Vera model of a UART capable of serial data exchanges with any other UART.
- APB Slave BFM – Used to ensure that violations in the APB accesses are appropriately captured and logged.
- Scoreboard – Tracks the data that are exchanged between the UART and the SIOTxrx models. This allows verification of the actual contents transmitted and/or received on either side in either direction.

10 Integration Considerations

After you have configured, tested, and synthesized your component with the coreTools flow, you can integrate the component into your own design environment.

10.1 Accessing Top-level Constraints

To get SDC constraints out of coreConsultant, you need to first complete the synthesis activity and then use the "write_sdc" command to write out the results:

1. This cC command sets synthesis to write out scripts only, without running DC:

   ```
   set_activity_parameter Synthesize ScriptsOnly 1
   ```

2. This cC command autocompletes the activity:

   ```
   autocomplete_activity Synthesize
   ```

3. Finally, this cC command writes out SDC constraints:

   ```
   write_sdc <filename>
   ```

10.2 Coherency

Coherency is where bits within a register are logically connected. For instance, part of a register is read at time 1 and another part is read at time 2. Being coherent means that the part read at time 2 is at the same value it was when the register was read at time 1. The unread part is stored into a shadow register and this is read at time 2. When there is no coherency, no shadow registers are involved.

A bus master may need to be able to read the contents of a register, regardless of the data bus width, and be guaranteed of the coherency of the value read. A bus master may need to be able to write a register coherently regardless of the data bus width and use that register only when it has been fully programmed. This may need to be the case regardless of the relationship between the clocks.

Coherency enables a value to be read that is an accurate reflection of the state of the counter, independent of the data bus width, the counter width, and even the relationship between the clocks. Additionally, a value written in one domain is transferred to another domain in a seamless and coherent fashion.

Throughout this appendix the following terms are used:

- **Writing**. A bus master programs a configuration register. An example is programming the load value of a counter into a register.
- **Transferring**. The programmed register is in a different clock domain to where it is used, therefore, it needs to be transferred to the other clock domain.
- **Loading**. Once the programmed register is transferred into the correct clock domain, it needs to be loaded or used to perform its function. For example, once the load value is transferred into the counter domain, it gets loaded into the counter.

10.2.1 Writing Coherently

Writing coherently means that all the bits of a register can be written at the same time. A peripheral may have programmable registers that are wider than the width of the connected APB data bus, which prevents all the bits being programmed at the same time unless additional coherency circuitry is provided.

The programmable register could be the load value for a counter that may exist in a different clock domain. Not only does the value to be programmed need to be coherent, it also needs to be transferred to a different clock domain and then loaded into the counter. Depending on the function of the programmable register, a qualifier may need to be generated with the data so that it knows when the new value is currently transferred and when it should be loaded into the counter.

Depending on the system and on the register being programmed, there may be no need for any special coherency circuitry. One example that requires coherency circuitry is a 32-bit timer within an 8-bit APB system. The value is entirely programmed only after four 8-bit wide write transfers. It is safe to transfer or use the register when the last byte is currently written. An example where no coherency is required is a 16-bit wide timer within a 16-bit APB system. The value is entirely programmed after a single 16-bit wide write transfer.

Coherency circuitry enables the value to be loaded into the counter only when fully programmed and crossed over clock domains if the peripheral clock is not synchronous to the processor clock. While the load register is being programmed, the counter has access to the previous load value in case it needs to reload the counter.

Coherency circuitry is only added in cores where it is needed. The coherency circuitry incorporates an upper byte method that requires users to program the load register in LSB to MSB order when the peripheral width is smaller than the register width. When the upper byte is programmed, the value can be transferred and loaded into the load register. When the lower bytes are being programmed, they need to be stored in shadow registers so that the previous load register is available to the counter if it needs to reload. When the upper byte is programmed, the contents of the shadow registers and the upper byte are loaded into the load register.

The upper byte is the top byte of a register. A register can be transferred and loaded into the counter only when it has been fully programmed. A new value is available to the counter once this upper byte is written into the register. The following table shows the relationship between the register width and the peripheral bus width for the generation of the correct upper byte. The numbers in the table represent bytes, Byte 0 is the LSB and Byte 3 is the MSB. NCR means that no coherency circuitry is required, as the entire register is written with one access.

Table 10-1 Upper Byte Generation

	Upper Byte Bus Width		
Load Register Width	8	16	32
1 - 8	NCR	NCR	NCR
9 - 16	1	NCR	NCR
17 - 24	2	2	NCR
25 - 32	3	2 (or 3)	NCR

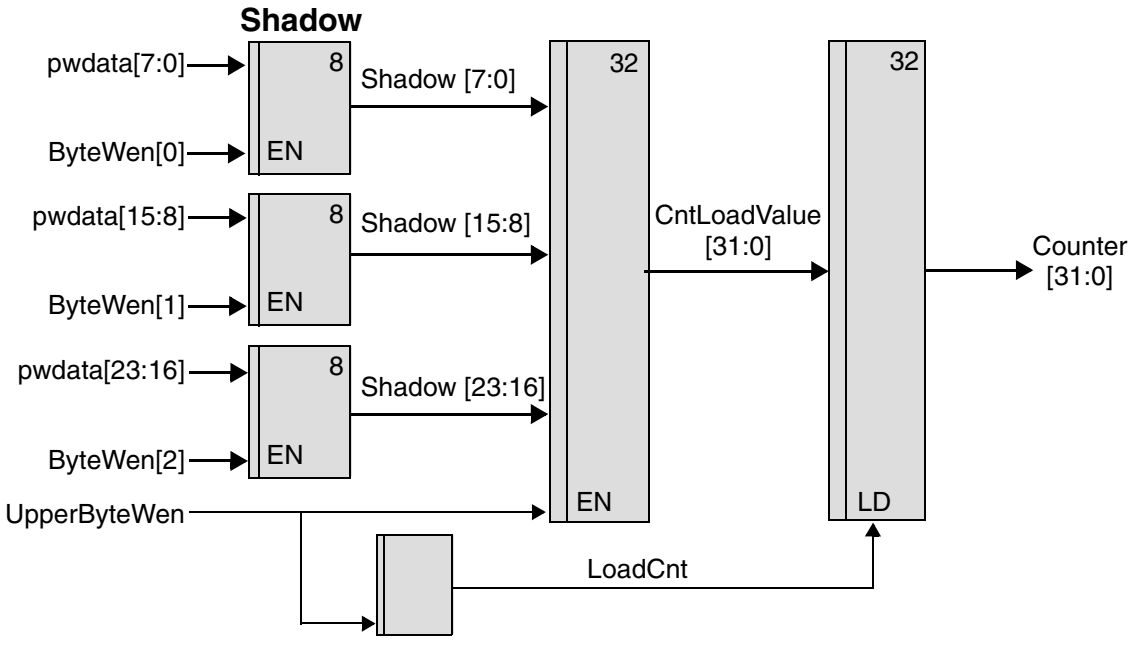
There are three relationship cases to be considered for the processor and peripheral clocks:

- Identical
- Synchronous (phase coherent but of an integer fraction)
- Asynchronous

10.2.1.1 Identical Clocks

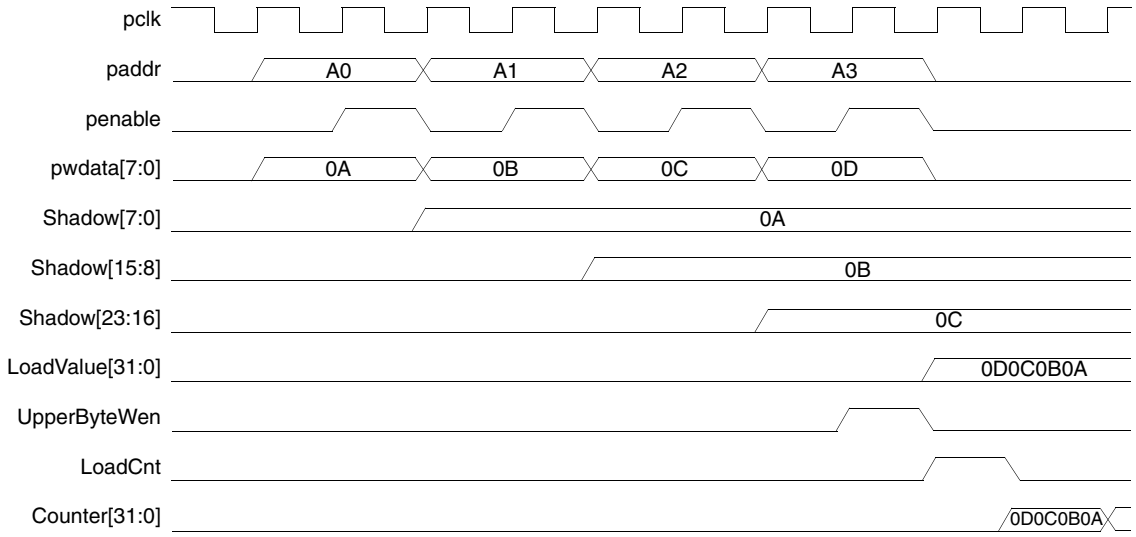
The following figure illustrates an RTL diagram for the circuitry required to implement the coherent write transaction when the APB bus clock and peripheral clocks are identical.

Figure 10-1 Coherent Loading – Identical Synchronous Clocks



The following figure shows a 32-bit register that is written over an 8-bit data bus, as well as the shadow registers being loaded and then loaded into the counter when fully programmed. The LoadCnt signal lasts for one cycle and is used to load the counter with CntLoadValue.

Figure 10-2 Coherent Loading – Identical Synchronous Clocks



Each of the bytes that make up the load register are stored into shadow registers until the final byte is written. The shadow register is up to three bytes wide. The contents of the shadow registers and the final byte are transferred into the CntLoadValue register when the final byte is written. The counter uses this register to load/initialize itself. If the counter is operating in a periodic mode, it reloads from this register each time the count expires.

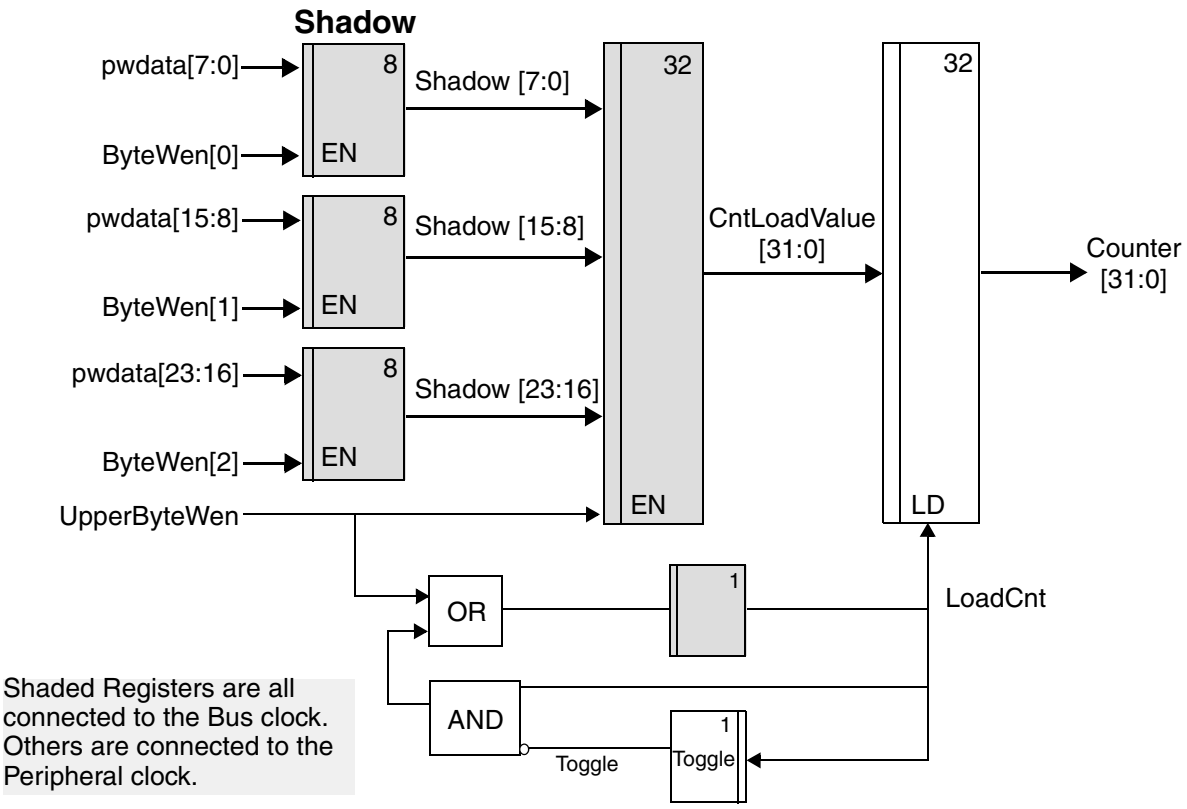
By using the shadow registers, the CntLoadValue is kept stable until it can be changed in one cycle. This allows the counter to be loaded in one access and the state of the counter is not affected by the latency in programming it. When there is a new value to be loaded into the counter initially, this is signaled by LoadCnt = 1. After the upper byte is written, the LoadCnt goes to zero.

10.2.1.2 Synchronous Clocks

When the clocks are synchronous but do not have identical periods, the circuitry needs to be extended so that the LoadCnt signal is kept high until a rising edge of the counter clock occurs. This extension is necessary so that the value can be loaded, using LoadCnt, into the counter on the first counter clock edge. At the rising edge of the counter clock if LoadCnt is high, then a register clocked with the counter clock toggles, otherwise it keeps its current value. A circuit detecting the toggling is used to clear the original LoadCnt by looking for edge changes. The value is loaded into the counter when a toggle has been detected. Once it is loaded, the counter should be free to increment or decrement by normal rules.

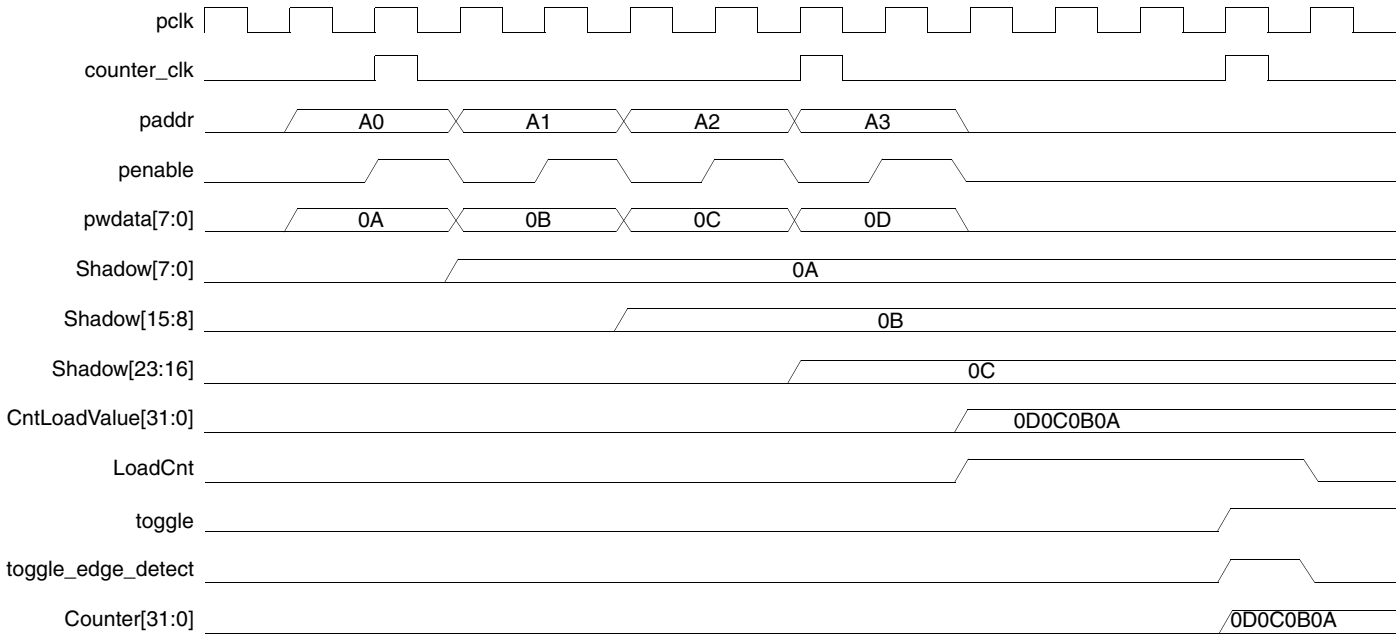
The following figure shows an RTL diagram for the circuitry required to implement the coherent write when the bus and peripheral clocks are synchronous.

Figure 10-3 Coherent Loading – Synchronous Clocks



The following figure shows a 32-bit register being written over an 8-bit data bus, as well as the shadow registers being loaded and then loaded into the counter when fully programmed. The LoadCnt signal is extended until a change in the toggle is detected and is used to load the counter.

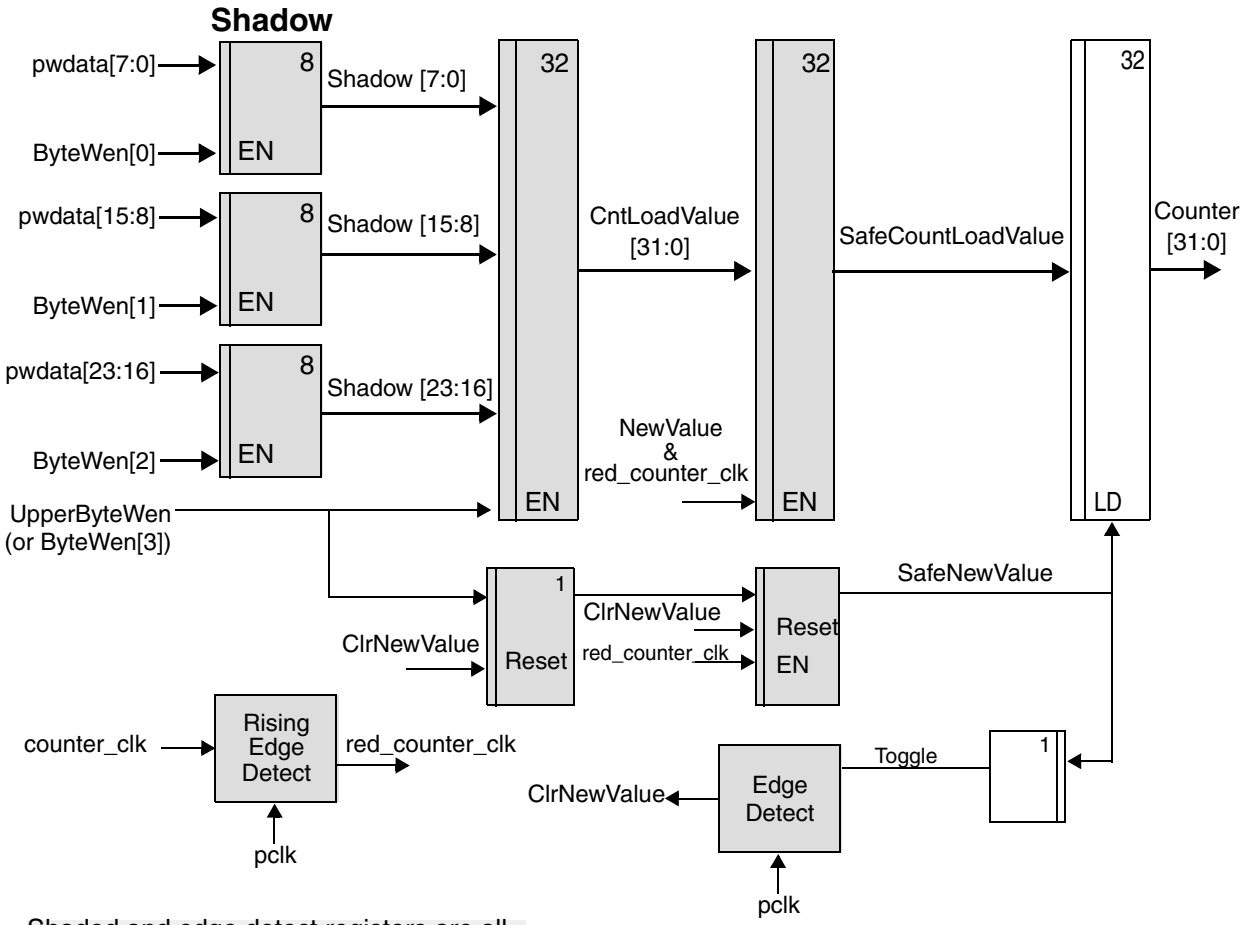
Figure 10-4 Coherent Loading – Synchronous Clocks



10.2.1.3 Asynchronous Clocks

When the clocks are asynchronous, the processor clock needs to be three-times the speed of the peripheral clock for the re-timing to operate correctly. The high pulse time of the peripheral clock needs to be greater than the period of the processor clock. The following figure shows an RTL diagram for the circuitry required to implement the coherent write when the bus and peripheral clocks are asynchronous.

Figure 10-5 Coherent Loading – Asynchronous Clocks



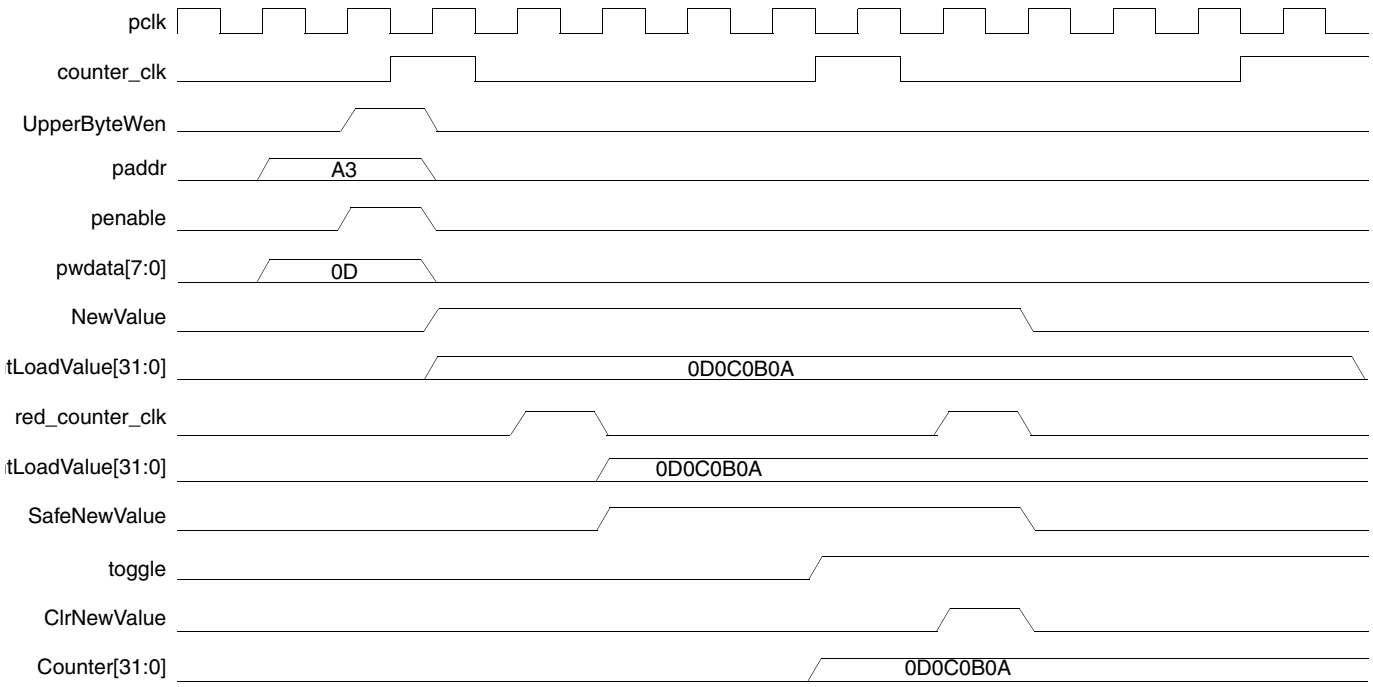
When the clocks are asynchronous, you need to transfer the contents of the register from one clock domain to another. It is not desirable to transfer the entire register through meta-stability registers, as coherency is not guaranteed with this method. The circuitry needed requires the processor clock to be used to re-time the peripheral clock. Upon a rising edge of the re-timed clock, the new value signal, NewValue, is transferred into a safe new value signal, SafeNewValue, which happens after the edge of the peripheral clock has occurred.

Every time there is a rising edge of the peripheral clock detected, the CntLoadValue is transferred into a SafeCntLoadValue. This value is used to transfer the load value across the clock domains. The SafeCntLoadValue only changes a number of bus clock cycles after the peripheral clock edge changes. A

counter running on the peripheral clock is able to use this value safely. It could be up to two peripheral clock periods before the value is loaded into the counter. Along with this loaded value, there also is a single bit transferred that is used to qualify the loading of the value into the counter.

The timing diagram depicted in the following figure does not show the shadow registers being loaded. This is identical to the loading for the other clock modes.

Figure 10-6 Coherent Loading – Asynchronous Clocks



The NewValue signal is extended until a change in the toggle is detected and is used to update the safe value. The SafeNewValue is used to load the counter at the rising edge of the peripheral clock. Each time a new value is written the toggle bit is flipped and the edge detection of the toggle is used to remove both the NewValue and the SafeNewValue.

10.2.2 Reading Coherently

For writing to registers, an upper-byte concept is proposed for solving coherency issues. For read transactions, a lower-byte concept is required. The following table provides the relationship between the register width and the bus width for the generation of the correct lower byte.

Table 10-2 Lower Byte Generation

	Lower Byte Bus Width		
Counter Register Width	8	16	32
1 - 8	NCR	NCR	NCR
9 - 16	0	NCR	NCR

Table 10-2 Lower Byte Generation

	Lower Byte Bus Width		
17 - 24	0	0	NCR
25 - 32	0	0	NCR

Depending on the bus width and the register width, there may be no need to save the upper bits because the entire register is read in one access, in which case there is no problem with coherency. When the lower byte is read, the remaining upper bytes within the counter register are transferred into a holding register. The holding register is the source for the remaining upper bytes. Users must read LSB to MSB for this solution to operate correctly. NCR means that no coherency circuitry is required, as the entire register is read with one access.

There are two cases regarding the relationship between the processor and peripheral clocks to be considered as follows:

- Identical and/or synchronous
- Asynchronous

10.2.2.1 Synchronous Clocks

When the clocks are identical and/or synchronous, the remaining unread bits (if any) need to be saved into a holding register once a read is started. The first read byte must be the lower byte provided in the previous table, which causes the other bits to be moved into the holding register, SafeCntVal, provided that the register cannot be read in one access. The upper bytes of the register are read from the holding register rather than the actual register so that the value read is coherent. This is illustrated in the following figure and in the timing diagram after it.

Figure 10-7 Coherent Registering – Synchronous Clocks

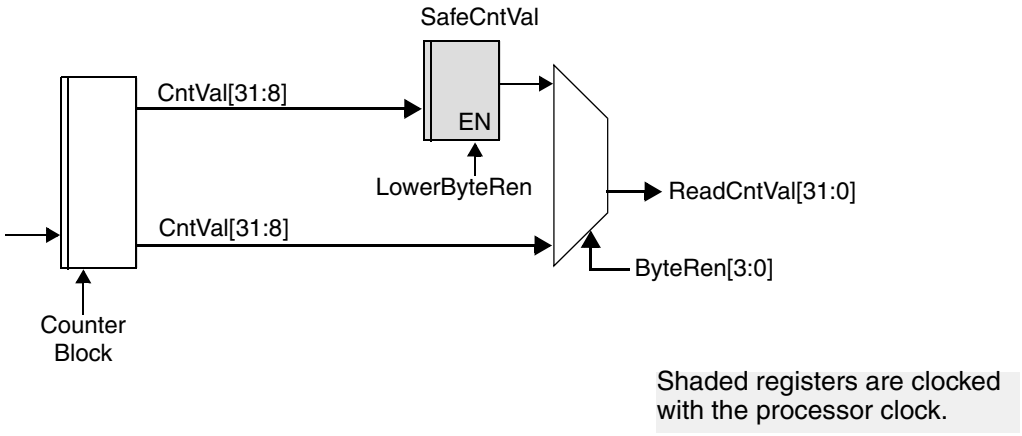
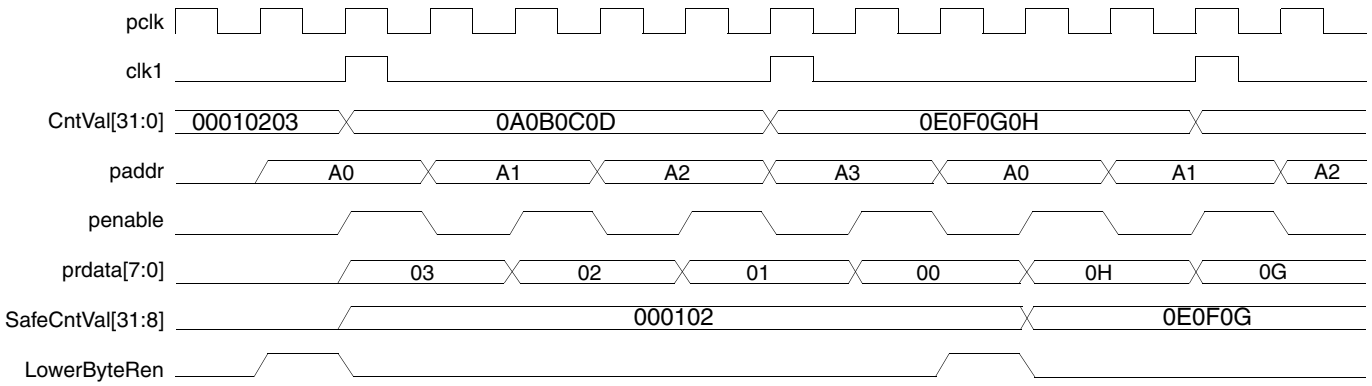


Figure 10-8 Coherent Registering – Synchronous Clocks



10.2.2.2 Asynchronous Clocks

When the clocks are asynchronous, the processor clock needs to be three times the speed of the peripheral clock for the re-timing to operate correctly. The high pulse time of the peripheral clock needs to be greater than the period of the processor clock.

To safely transfer a counter value from the counter clock domain to the bus clock domain, the counter clock signal should be transferred to the bus clock domain. When the rising edge detect of this re-timed counter clock signal is detected, it is safe to use the counter value to update a shadow register that holds the current value of the counter.

While reading the counter contents it may take multiple APB transfers to read the value.

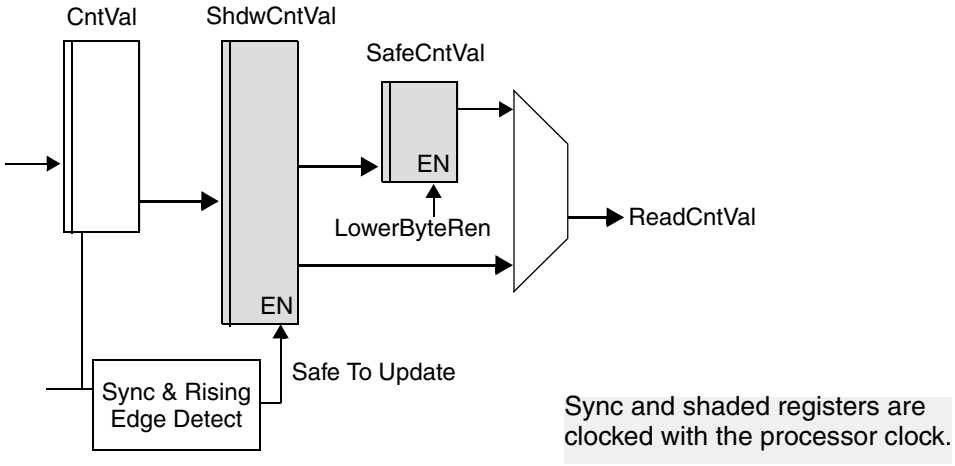
Note You must read LSB to MSB when the bus width is narrower than the counter width.

Once a read transaction has started, the value of the upper register bits need to be stored into a shadow register so that they can be read with subsequent read accesses. Storing these upper bits preserves the coherency of the value that is being read. When the processor reads the current value it actually reads the contents of the shadow register instead of the actual counter value. The holding register is read when the bus width is narrower than the counter width. When the LSB is read, the value comes from the shadow register; when the remaining bytes are read they come from the holding register. If the data bus width is wide enough to read the counter in one access, then the holding registers do not exist.

The counter clock is registered and successively pipelined to sense a rising edge on the counter clock. Having detected the rising edge, the value from the counter is known to be stable and can be transferred into the shadow register. The coherency of the counter value is maintained before it is transferred, because the value is stable.

The following figure illustrates the synchronization of the counter clock and the update of the shadow register.

Figure 10-9 Coherency and Shadow Registering – Asynchronous Clocks



10.3 Performance

This section discusses performance and the hardware configuration parameters that affect the performance of the DW_apb_uart.

10.3.1 Area

This section provides information to help you configure area for your configuration.

The following table includes synthesis results that have been generated using the TSMC 65nm and 28 nm technology libraries.

Table 10-3 Synthesis Results Using TSMC 65nm and 28 nm Technology Libraries

Configuration	Operating Frequency	Gate Count TSMC 65 nm	 TSMC 28nm
Default Configuration	pclk: 200MHz	2924 gates	2920 gates
Maximum Configuration-1: AFCE_MODE=1 APB_DATA_WIDTH=8 CLK_GATE_EN=1 CLOCK_MODE=2 DEBUG=1 DMA_EXTRA=1 FIFO_ACCESS=1 FIFO_MODE=2048 FIFO_STAT=1 SHADOW=1 SIR_MODE=1 THRE_MODE_USER=1	pclk: 200MHz sclk: 25MHz	6861 gates	6686 gates
Maximum Configuration-2: AFCE_MODE=1 BAUD_CLK=0 CLOCK_MODE=2 DEBUG=1 DMA_EXTRA=1 FRACTIONAL_BAUD_DIVISOR_EN=1 MEM_SELECT_USER=1 THRE_MODE_USER=1 UART_16550_COMPATIBLE=1 UART_9BIT_DATA_EN=1 UART_RS485_INTERFACE_EN=1 APB_DATA_WIDTH=32	pclk: 200MHz sclk: 25MHz	10803 gates	10729 gates

10.3.2 Power Consumption

The following table provides information about the power consumption of the DW_apb_uart using the TSMC 65nm technology library and how it affects performance.

Table 10-4 Power Consumption of DW_apb_uart Using TSMC 65nm Technology Library

Configuration	Operating Frequency	TSMC 65 nm		TSMC 28 nm	
		Static Power Consumption	Dynamic Power Consumption	Static Power Consumption	Dynamic Power Consumption
Default Configuration	pclk: 200MHz	0.182µW	1.07µW	9.25µW	98.016µW
Maximum Configuration-1: AFCE_MODE=1 APB_DATA_WIDTH=8 CLK_GATE_EN=1 CLOCK_MODE=2 DEBUG=1 DMA_EXTRA=1 FIFO_ACCESS=1 FIFO_MODE=2048 FIFO_STAT=1 SHADOW=1 SIR_MODE=1 THRE_MODE_USER=1	pclk: 200MHz sclk: 25MHz	1.829µW	0.182µW	23.2µW	1.413µW
Maximum Configuration-2: AFCE_MODE=1 BAUD_CLK=0 CLOCK_MODE=2 DEBUG=1 DMA_EXTRA=1 FRACTIONAL_BAUD_DIVISOR_EN=1 MEM_SELECT_USER=1 THRE_MODE_USER=1 UART_16550_COMPATIBLE=1 UART_9BIT_DATA_EN=1 UART_RS485_INTERFACE_EN=1 APB_DATA_WIDTH=32	pclk: 200MHz sclk: 25MHz	0.735µW	3.672µW	0.378µW	2.906µW

The following table provides information about the power consumption of the DW_apb_uart using the TSMC 28nm technology library and how it affects performance.

Table 10-5 Power Consumption of DW_apb_uart Using TSMC 28nm Technology Library

Configuration	Operating Frequency	Static Power Consumption	Dynamic Power Consumption
Default Configuration	166.67 MHz	312.4062 µW	241.3092 µW
Minimum Configuration: FIFO_MODE=16	166.67 MHz	312.4062 µW	241.3092 µW
Maximum Configuration: APB_DATA_WIDTH=8 FIFO_MODE=2048 CLOCK_MODE=2 AFCE_MODE=1 THRE_MODE_USER=1 SIR_MODE=1 CLK_GATE_EN=1 FIFO_ACCESS=1 DMA_EXTRA=1 DEBUG=1	pclk : 166.67 MHz sclk : 25 MHz	713.1808 µW	450.5119 µW

A
Synchronizer Methods

This appendix describes the synchronizer methods (blocks of synchronizer functionality) that are used in the DW_apb_uart to cross clock boundaries.

This appendix contains the following sections:

The DesignWare Building Blocks (DWBB) contains several synchronizer components with functionality similar to methods documented in this appendix. For more information about the DWBB synchronizer components go to:
www.synopsys.com/products/designware/docs/doc/dwf/datasheets/interface_cdc_overview.pdf

A.1 Synchronizers Used in DW_apb_uart

Each of the synchronizers and synchronizer sub-modules are comprised of verified DesignWare Basic Core (BCM) RTL designs. The BCM synchronizer designs are identified by the synchronizer type. The corresponding RTL files comprising the BCM synchronizers used in the DW_apb_i2c are listed and cross referenced to the synchronizer type in Table A-1. Note that certain BCM modules are contained in other BCM modules, as they are used in a building block fashion.

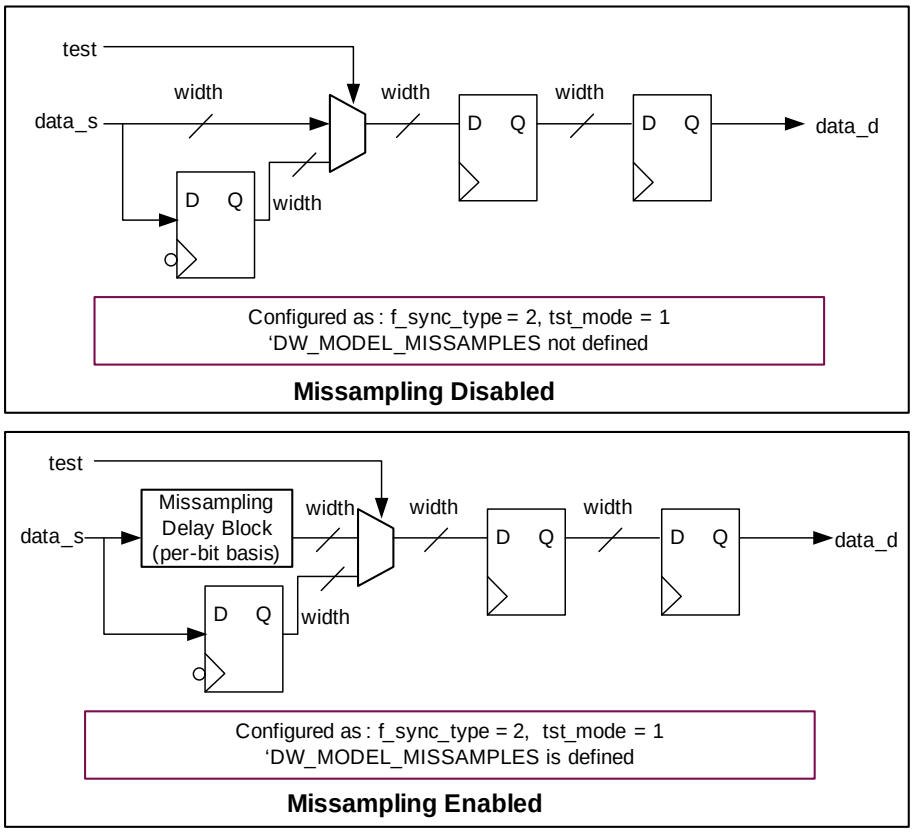
Table A-1 Synchronizers used in DW_apb_uart

Synchronizer module file	Sub module file	Synchronizer Type and Number
DW_apb_uart_bcm21.v		Synchronizer 1: Simple Multiple Register Synchronizer
DW_apb_uart_bcm41.v	DW_apb_uart_bcm21.v	Synchronizer 2: Simple Multiple Register Synchronizer with Configurable Polarity Reset
DW_apb_uart_bcm25.v	DW_apb_uart_bcm21.v DW_apb_uart_bcm23.v	Synchronizer 3: Simple Multiple register Synchronizer with Acknowledge

A.2 Synchronizer 1: Simple Double Register Synchronizer

This is a single clock data bus synchronizer for synchronizing data that crosses asynchronous clock boundaries. This synchronization scheme is always present for synchronizing the Modem control signals. If pclk and sclk are asynchronous (CLOCK_MODE =Enabled) then DW_apb_uart_bcm21 is instantiated inside the core for synchronization. This uses two stage synchronization process () both using positive edge of clock.

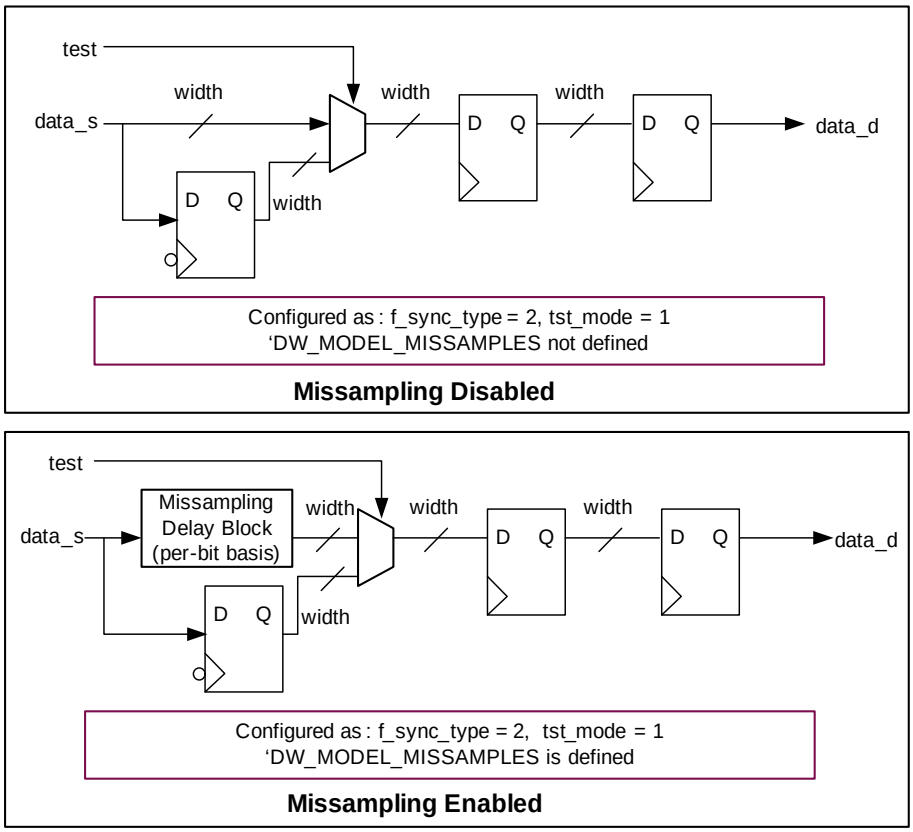
Figure A-1 Block diagram of Synchronizer 1 with two stage synchronization (both positive edge)



A.3 Synchronizer 2: Simple Double Register Synchronizer with Configurable Polarity Reset

This is a single clock data bus synchronizer for synchronizing data that crosses asynchronous clock boundaries with configurable polarity reset. The synchronization scheme (DW_apb_uart_bcm41.v) is always present in core for synchronization of sin input signal. This DW_apb_uart_bcm41 synchronizer is similar to the DW_apb_uart_bcm21 synchronizer and the polarity of the output of this synchronizer can be configured. Figure A-2 shows the block diagram of Synchronizer 2.

Figure A-2 Block diagram of Synchronizer 2 with two stage synchronization (both positive edge)



A.4 Synchronizer 3: Simple Double register Synchronizer with Acknowledge

This synchronizer (DW_apb_uart_bcm25.v) passes data values from the source domain to the destination domain through a hand-shake protocol which includes an acknowledge back to the source domain. This module uses clock-domain-crossing techniques to safely transfer pulses between logic operating on different clocks and acknowledge guaranteeing the pulse has arrived in the destination domain. This synchronizer is present based on the core configuration. If pclk and sclk are synchronous (CLOCK_MODE=Enabled), then DW_apb_uart_bcm25.v is used to synchronize the data signals from one domain to the other domain.

This synchronizer uses the same DW_apb_uart_bcm21.v module to synchronize the data from source clock domain to destination clock domain.

B
Application Notes

This appendix provides application notes for DW_apb_uart.

Q. The DesignWare DW_apb_uart Databook states that the timeout detection hardware block is optional, but I do not see any configuration parameter available for this feature. How do I set this option?

A. If FIFO_MODE!=NONE or CLK_GATE_EN=1, you will get this timeout detector block instantiated at the top level. If CLK_GATE_EN=1, clock gating circuitry is also included in the Timeout Detector block.

Q. If I have a DesignWare license, can I use the DW_apb_uart driver kit?

A. No, you cannot use the DW_apb_uart driver kit with a DesignWare license. The DW_apb_uart driver kit requires the DWC-APB-Advanced-Source license.

Q. I am using the DW_apb_uart driver example. The driver includes a header file called inttypes.h but it is not supplied with the example driver files. What should I do?

A. The inttypes.h, stdarg.h and stddef.h are all standard C header files that come with the ARM C compiler and are present in $ARMHOME/common/include directory.

To find out how to obtain and configure an ARM-946 CPU model, refer to the DW_apb_uart *Driver Kit Databook* at:

http://www.synopsys.com/products/designware/docs/drivers/DW_apb_uart/latest/doc/dw_apb_uart_driver.pdf

Q. What is the difference between bits PTIME (bit 7) and ETBEI (bit 1) of the IER register?

A. PTIME is used to enable the Programmable THRE Interrupt Mode, when DW_apb_uart is configured to support this mode (THRE_MODE_USER = Enabled). The PTIME bit field is writable only when the Programmable THRE Interrupt Mode Enable configuration parameter (THRE_MODE_USER) is set to True. It is always readable. This is used to enable or disable the generation of THRE Interrupt.

ETBEI is used to enable the interrupt regardless of the setting of the Programmable THRE Interrupt Mode. The interrupt provides the following information depending on whether the Programmable THRE Interrupt Mode is enabled or disabled. The interrupt indicates either the transmitter holding register empty (THRE_MODE_USER is disabled) or the transmit FIFO at or below threshold (THRE_MODE_USER is enabled).

Thus, DW_apb_uart has been configured to have Programmable THRE Interrupt Mode, the PTIME bit is used to switch between the two modes of operation.

Q. *When I read the Component Parameter register (CPR), it always returns a value of 0. Why does this occur?*

A. The CPR is only valid when DW_apb_uart is configured to have the Component Parameter register implemented (UART_ADD_ENCODED_PARAMS is set to Yes). If the Component Parameter register is not implemented, this register does not exist and reading from this register address returns 0.

Q. *Why do we have IIR[3:0]=0x7, an additional busy detect interrupt in comparison to the 16550 National specification?*

A. Busy functionality helps to safe guard against errors if the LCR, DLL, and/or DLH registers are changed during a transaction even though they should only be set during initialization (as stated in the National specification for DLL/DLH, section 8.3 p.16).

Q. *Why are there two resets in DW_apb_uart?*

A. When operating in asynchronous serial clock mode, dedicated reset signals for the different clock domains are required. All the logic operating on pclk is reset by presetn, while the logic operating on sclk is reset by s_rst_n.

Note

- The presetn and s_rst_n signals must be synchronous to the pclk and sclk clock signals, respectively.
- For correct operation, the logic on both clock domains should be reset simultaneously; resetting only one clock domain results in undetermined behavior. When de-asserting the reset signals, s_rst_n should be de-asserted first.

The software reset (when this feature is enabled) resets both pclk and sclk logic; although this signal is generated in the pclk domain, internal synchronization ensures it can be safely used in the sclk domain without the risk of metastability.

When operating in synchronous serial clock mode all logic is reset by the presetn signal.

Q. *Is it possible to do burst (back-to-back) FIFO reads/writes? For example, can the write and enable lines be held for two consecutive clocks to do back-to-back transfers?*

A. DW_apb_uart does accommodate burst FIFO reads and writes using the SRBR (Shadow Receive Buffer Register) and STHR (Shadow Transmit Holding Register).

Q. *What activity occurs on the sout and sir_out_n signals in UART and IR mode?*

A. The serial data out signal sout is driven high if DW_apb_uart is in loopback mode or serial infrared mode. Otherwise, it is assigned to the current bit of the character that is being transmitted.

Q. *Is there a way to find out whether DW_apb_uart is operating in either normal serial data mode or in Infrared mode?*

A. The only way to find out whether DW_apb_uart is operating in either normal serial data mode or in Infrared mode is to check the Modem Control Register (MCR) bit 6. If MCR[6]=0, IrDA SIR Mode disabled. If MCR[6]=1, IrDA SIR Mode is enabled. This bit is writable only when SIR_MODE is enabled.

Q. *Is DW_apb_uart completely compliant with the 16750 specification from Texas Instruments?*

A. No, DW_apb_uart is not completely compliant with the 16750 TI specification. DW_apb_uart does not support features such as sleep mode (has an enable bit in the IER) and low-power mode (has an enable bit in the IER), which seem to be for a uart/clock oscillator control within their chip.

Q. *What is the safest way to hold DW_apb_uart in reset or non-response state when it is being initialized so that no characters during this time period are received/transmitted?*

A. When you are in the initialization stage, there are two ways to ensure that no characters during this time period are received/transmitted:

a. Set DLL and DLH to 0; configure the control registers for transfer, for instance, set LCR register (data size, stop bits, and so on); set the MCR register; set the FCR register to enable FIFOs; and set IER register to enable interrupts. Once this is completed, write to the divisor registers DLL and DLH to set the bit rate and then write the data to be transmitted into the THR register.

b. There is a loopback mode (MCR[4]) that provides a local loopback feature. When this bit is set to logic 1, the transmitter Serial Output (SOUT) is set to a logic 1 state, the receiver Serial Input (SIN) is disconnected, and the output of the Transmitter Shift Register is "looped back" into the Receiver Shift Register input. So DW_apb_uart does not receive anything because the sin signal is disconnected.

   Put DW_apb_uart in loopback mode; setup the control registers for transfer (for instance, write the divisor registers to set the bit rate); set LCR register (data size, stop bits, and so on); set MCR register; set FCR register to enable FIFOs; and set IER register to enable interrupts. Once this is completed, write 0 in MCR[4] (no loopback mode) and then write the data to be transmitted into the THR register.

Q. *After initialization of DW_apb_uart, if we want to program the baud rate, how can we make sure we do not receive/send any characters during this configuration time?*

A. After initialization of DW_apb_uart, once TX/RX has started, if you want to reprogram the baud rate, make sure that the serial transfer has been completed. The safest way to accomplish this is to poll USR[0] (Busy) bit, and when DW_apb_uart is not busy, change the register values.

Q. *What is the functionality of the SIR_MODE and SIR_LP_MODE configuration parameters?*

A. IrDA 1.0 SIR mode specifies a maximum baud rate of 115.2 Kbaud. But if you want to operate using the low-power pulse duration (SIR_LP_MODE=1) of 1.63us, you must run at 115.2K baud. DW_apb_uart automatically handles this by being configured with asynchronous clock support.

If SIR_MODE is set to 1, you can have a baud rate anywhere from 9.6K to 115.2K with 3/16 nominal pulse width support. However, if SIR_LP_MODE is set to 1, you must run at 115.2K.

If you set CLOCK_MODE to 2, SIR_MODE to 1, and run at 115.2K, you are getting functionality for SIR_LP_MODE set to 1. If you set a baud rate higher or lower then 115.2K in SIR_LP_MODE, it still generates 3/16 pulse width but will not work properly because it violates the requirement of 1.63us for low-power IrDA SIR mode.

C
Glossary

active command queue	Command queue from which a model is currently taking commands; see also command queue.
activity	A set of functions in coreConsultant that step you through configuration, verification, and synthesis of a selected core.
application design	Overall chip-level design into which a subsystem or subsystems are integrated.
BFM	Bus-Functional Model — A simulation model used for early hardware debug. A BFM simulates the bus cycles of a device and models device pins, as well as certain on-chip functions. See also Full-Functional Model.
big-endian	Data format in which most significant byte comes first; normal order of bytes in a word.
blocked command stream	A command stream that is blocked due to a blocking command issued to that stream; see also command stream, blocking command, and non-blocking command.
blocking command	A command that prevents a testbench from advancing to next testbench statement until this command executes in model. Blocking commands typically return data to the testbench from the model.
command channel	Manages command streams. Models with multiple command channels execute command streams independently of each other to provide full-duplex mode function.
command stream	The communication channel between the testbench and the model.
component	A generic term that can refer to any synthesizable IP or verification IP in the DesignWare Library. In the context of synthesizable IP, this is a configurable block that can be instantiated as a single entity (VHDL) or module (Verilog) in a design.
configuration	The act of specifying parameters for a core prior to synthesis; can also be used in the context of VIP.
configuration intent	Range of values allowed for each parameter associated with a reusable core.

core	Any configurable block of synthesizable IP that can be instantiated as a single entity (VHDL) or module (Verilog) in a design. Core is the preferred term for a big piece of IIP. Anything that requires coreConsultant for configuration, as well as anything in the DesignWare Cores library, is a core.
core developer	Person or company who creates or packages a reusable core. All the cores in the DesignWare Library are developed by Synopsys.
core integrator	Person who uses coreConsultant or coreAssembler to incorporate reusable cores into a system-level design.
coreAssembler	Synopsys product that enables automatic connection of a group of cores into a subsystem. Generates RTL and gate-level views of the entire subsystem.
coreConsultant	A Synopsys product that lets you configure a core and generate the design views and synthesis views you need to integrate the core into your design. Can also synthesize the core and run the unit-level testbench supplied with the core.
coreKit	An unconfigured core and associated files, including the core itself, a specified synthesis methodology, interfaces definitions, and optional items such as verification environment files and core-specific documentation.
cycle command	A command that executes and causes HDL simulation time to advance.
decoder	Software or hardware subsystem that translates from and “encoded” format back to standard format.
design context	Aspects of a component or subsystem target environment that affect the synthesis of the component or subsystem.
design creation	The process of capturing a design as parameterized RTL.
Design View	A simulation model for a core generated by coreConsultant.
DesignWare cores	A specific collection of synthesizable cores that are licensed individually. For more information, refer to www.synopsys.com/designware.
DesignWare Library	A collection of synthesizable IP and verification IP components that is authorized by a single DesignWare license. Products include SmartModels, VMT model suites, DesignWare Memory Models, Building Block IP, and the DesignWare Synthesizable Components.
dual role device	Device having the capabilities of function and host (limited).
endian	Ordering of bytes in a multi-byte word; see also little-endian and big-endian.
Full-Functional Mode	A simulation model that describes the complete range of device behavior, including code execution. See also BFM.
GPIO	General Purpose Input Output.
GTECH	A generic technology view used for RTL simulation of encrypted source code by non-Synopsys simulators.
hard IP	Non-synthesizable implementation IP.

HDL	Hardware Description Language – examples include Verilog and VHDL.
IIP	Implementation Intellectual Property — A generic term for synthesizable HDL and non-synthesizable “hard” IP in all of its forms (coreKit, component, core, MacroCell, and so on).
implementation view	The RTL for a core. You can simulate, synthesize, and implement this view of a core in a real chip.
instantiate	The act of placing a core or model into a design.
interface	Set of ports and parameters that defines a connection point to a component.
IP	Intellectual property — A term that encompasses simulation models and synthesizable blocks of HDL code.
little-endian	Data format in which the least-significant byte comes first.
MacroCell	Bigger IP blocks (6811, 8051, memory controller) available in the DesignWare Library and delivered with coreConsultant.
master	Device or model that initiates and controls another device or peripheral.
model	A Verification IP component or a Design View of a core.
monitor	A device or model that gathers performance statistics of a system.
non-blocking command	A testbench command that advances to the next testbench statement without waiting for the command to complete.
peripheral	Generally refers to a small core that has a bus connection, specifically an APB interface.
RTL	Register Transfer Level. A higher level of abstraction that implies a certain gate-level structure. Synthesis of RTL code yields a gate-level design.
SDRAM	Synchronous Dynamic Random Access Memory; high-speed DRAM adds a separate clock signal to control signals.
SDRAM controller	A memory controller with specific connections for SDRAMs.
slave	Device or model that is controlled by and responds to a master.
SoC	System on a chip.
soft IP	Any implementation IP that is configurable. Generally referred to as synthesizable IP.
static controller	Memory controller with specific connections for Static memories such as asynchronous SRAMs, Flash memory, and ROMs.
subsystem	In relation to coreAssembler, highest level of RTL that is automatically generated.
synthesis intent	Attributes that a core developer applies to a top-level design, ports, and core.

synthesizable IP	A type of Implementation IP that can be mapped to a target technology through synthesis. Sometimes referred to as Soft IP.
technology-independent	Design that allows the technology (that is, the library that implements the gate and via widths for gates) to be specified later during synthesis.
Testsuite Regression Environment (TRE)	A collection of files for stand-alone verification of the configured component. The files, tests, and functionality vary from component to component.
VIP	Verification Intellectual Property — A generic term for a simulation model in any form, including a Design View.
workspace	A network location that contains a personal copy of a component or subsystem. After you configure the component or subsystem (using coreConsultant or coreAssembler), the workspace contains the configured component/subsystem and generated views needed for integration of the component/subsystem at the top level.
wrap, wrapper	Code, usually VHDL or Verilog, that surrounds a design or model, allowing easier interfacing. Usually requires an extra, sometimes automated, step to create the wrapper.
zero-cycle command	A command that executes without HDL simulation time advancing.

Index