SYNOPSYS®

DesignWare DW_apb_uart DataBook Universal Asynchronous Receiver/Transmitter Peripheral

Release 1.01a

February 19, 2002

Preface

About This Manual

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^{TM} Specification, Revision 2.0* from ARM. Readers are assumed to be familiar with this specification.

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 coreKit, and synthesis information for the coreKit.

Related Documents

To see a complete listing of documentation related to the DW_apb_uart within the SoC Platform, refer to *Guide to SoC Platform Documentation*.

Manual Overview

This manual contains the following chapters and appendixes:

Preface	Describes the manual and lists the typographical conventions and symbols used in it.
Chapter 1 "DW_apb_uart Description"	Provides a functional description, pin and parameter descriptions, and a memory map.
Chapter 2 "Verification of DW_apb_uart"	Provides an overview of the component testbench and a description of the tests that are run to verify the coreKit.
Chapter 3 "Synthesis of DW_apb_uart"	Provides synthesis information for the coreKit.

Appendix A "APB Slave Interface"	Provides general information on interfacing to an APB slave device.

Typographical and Symbol Conventions

The following conventions are used throughout this document:

Table 1: Documentation Conventions

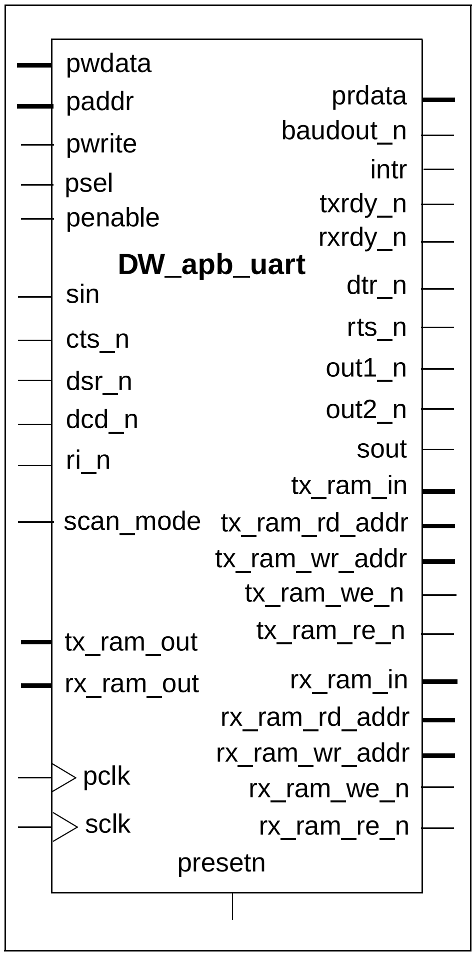
Convention	Description and Example
%	Represents the UNIX prompt.
Bold	User input (text entered by the user). `% cd $LMC_HOME/hdl`
`Monospace`	System-generated text (prompts, messages, files, reports). `No Mismatches: 66 Vectors processed: 66 Possible"`
Italic or *`Italic`*	Variables for which you supply a specific value. As a command line example: `% setenv LMC_HOME prod_dir` In body text: In the previous example, *prod_dir* is the directory where your product must be installed.
| (Vertical rule)	Choice among alternatives, as in the following syntax example: `-effort_level low | medium | high`
[ ] (Square brackets)	Enclose optional parameters: `pin1 [pin2 ... pinN]` In this example, you must enter at least one pin name (*pin1*), but others are optional ([*pin2* … *pinN*]).
TopMenu > **SubMenu**	Pulldown menu paths, such as: **File** > **Save As …**

Contents

1 DW_apb_uart Description

Overview

- Functionality based on the industry standard 16550
- AMBA 2.0 compliant APB Interface with synthesis selectable prdata & pwdata bus widths (8, 16, 32)
- Synthesis selectable transmit and receive FIFO depths (0, 16, 32,...,2048),
- Synthesis selectable internal DesignWare D-flip-flop based RAM (DW_ram_r_w_s_dff)
- Synthesis selectable synchronous or asynchronous external Read Port RAM interface when external RAMs are selected
- Synthesis selectable asynchronous serial clock support (pclk or pclk and sclk)
- Synthesis selectable ‘lockup latch’ insertion before clock boundary crossing, in two-clock implementations, for test purposes
- Programmable FIFO disabling
- External memory read enable signals for RAM wake-up when external RAMs are selected



- Support for any serial data baud rate, subject to the serial clock frequency, as follows: baud rate = (serial clock frequency) / (16 * divisor)
- Modem and status lines are independently controlled

Installation

For instructions about installing a coreKit, refer to the *coreConsultant User Guide*, the topic titled "Installing a coreKit"

DW_apb_uart Description

The DW_apb_uart is a parameterizable, synthesizable, and programmable Universal Asynchronous Receiver/Transmitter (UART), modeled after the industry-standard 16550. However, the register address space has been relocated to 32-bit data boundaries for APB bus implementation.

The DW_apb_uart is used to serially receive and transmit data to a peripheral, modem, or data set. It contains registers to control the character length, baud rate, parity generation/checking, and interrupt generation. It is also equipped with FIFO controls to buffer the transmit and receive data, and to enable/disable the FIFOs. It has external DMA signals *(*txrdy_n and rxrdy_n), to signal when data is ready to be read or when the transmit FIFO is empty.

In addition, a new host of features is supported, including an AMBA compliant APB Interface and support for a separate serial data clock. Support for separate CPU bus interface and serial interface clocks solves problems surrounding CPU data synchronization in relationship to required serial baud clock requirements. All data crossing between the two clock domains is guaranteed via full handshaking and level syncing synchronization. Also, the capability to insert lock-up latches in the data path, just prior to clock crossing, has been added for test purposes.

The ability to select between external customer supplied FIFO RAMs or internal DesignWare D-flip-flop based RAMs (DW_ram_r_w_s_dff) is now available. When external RAM support is chosen, the type of memory, synchronous or asynchronous read port, can also be selected. Asynchronous read port memory provides read data during the clock cycle that has the address and read signals active, for sampling on the next rising clock edge. Synchronous read port memory registers the data at the current address out and is not available until the next clock cycle. A generalized functional diagram of the DW_apb_uart appears in Figure 1 on page 9.

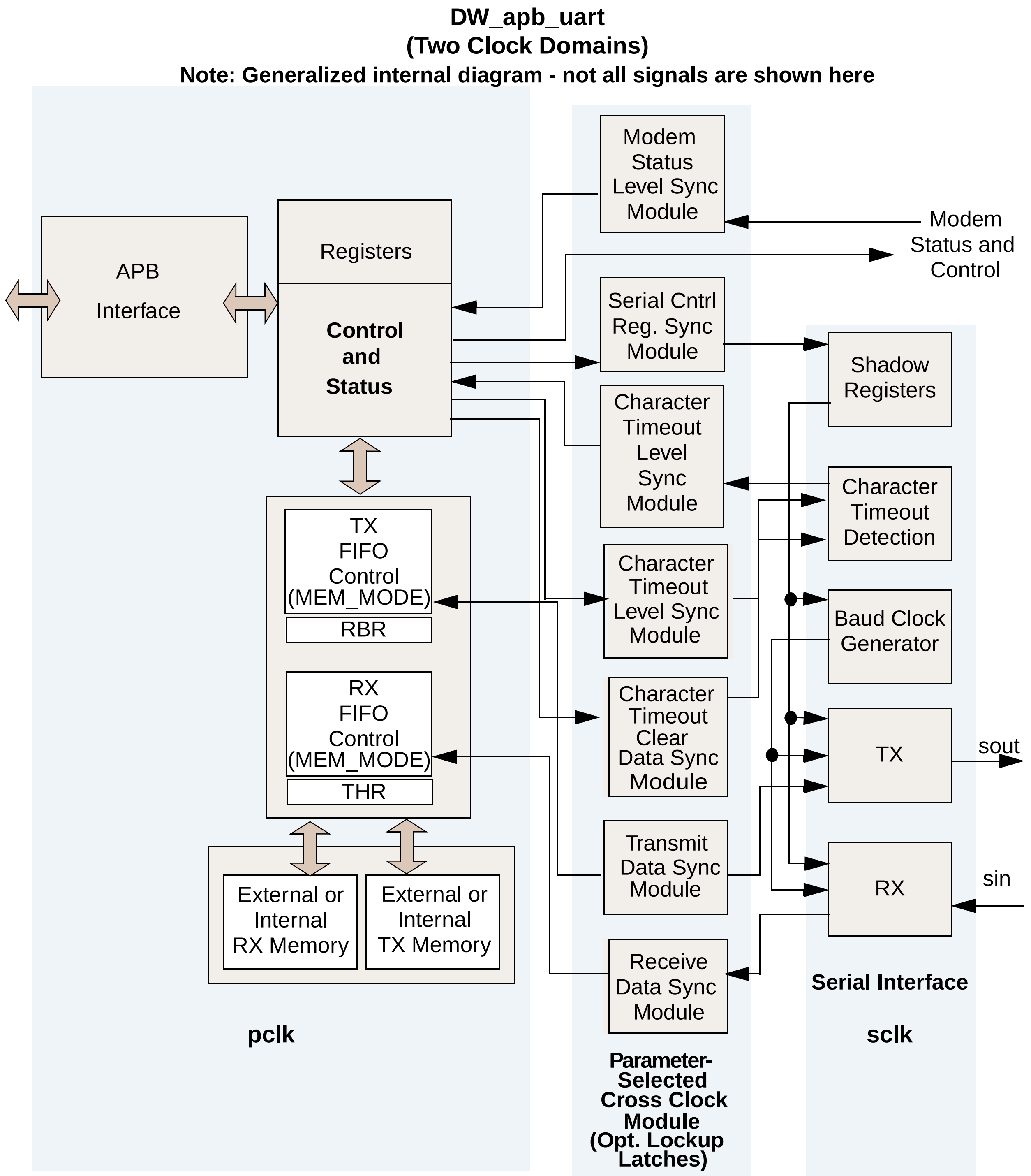

DW_apb_uart
(Two Clock Domains)
Note: Generalized internal diagram - not all signals are shown here
APB Interface
Registers
Control and Status
TX FIFO Control (MEM_MODE)
RBR
RX FIFO Control (MEM_MODE)
THR
External or Internal RX Memory
External or Internal TX Memory
pclk
Modem Status Level Sync Module
Serial Cntrl Reg. Sync Module
Character Timeout Level Sync Module
Character Timeout Level Sync Module
Character Timeout Clear Data Sync Module
Transmit Data Sync Module
Receive Data Sync Module
Parameter-Selected Cross Clock Module (Opt. Lockup Latches)
Modem Status and Control
Shadow Registers
Character Timeout Detection
Baud Clock Generator
TX
RX
sout
sin
Serial Interface
sclk

Figure 1: DW_apb_uart Functional Diagram

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

APB Interface - Standard AMBA 2.0 compliant APB interface.

Registers, Control and Status - Primary control and status registers exist in this module, as well as the main UART functionality. Serial data control registers are stored here and used for control and status generation. In addition, copies of serial control data is also forwarded on to the Serial clock domain. See "Programming the DW_apb_uart" on page 27 for complete register and control details. This module is also responsible for Interrupt generation based on transmitter and receiver status, as well as which interrupts are enabled. SeeTable 2 on page 13 for details.

RX and TX FIFO Controllers - These special FIFO controllers implement specially designed logic to access data elements before the top of the FIFO. This guarantees correct status and data at all times.

External/Internal Memory - Customer supplied memory can be implemented and accessed via external memory ports, when external RAMs are selected. As well, internal DesignWare memory can be instantiated within the design. This selection is controlled via the 'Fifo Memory' synthesis parameter. When internal memories (or no FIFOs) are chosen, unused external design ports are stripped out. In addition, when internal memory is chosen, the only type supported is an internal DesignWare D-flip-flop based RAM (DW_ram_r_w_s_dff), with an asynchronous read port.

Cross Clock Module - This module is responsible for synchronization of all registers and data across the two system clock boundaries, when a two clock design is implemented. If two clocks are chosen, the appropriate Data Sync and Level Sync modules are created to handle synchronization. If only one clock is implemented, all synchronization logic is absent and signals are simply passed through this module.

Note that a full synchronization handshake takes place on all signals that are 'data synchronized'. All signals that are 'level synchronized' are simply passed through three 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 the serial clock (sclk) to being no faster than 4× the pclk clock. A serial clock faster than four times the pclk signal does not leave enough time for a complete incoming character to be received and pushed into the RX FIFO. However, in most cases, the pclk signal will be faster than the serial clock and this should never be an issue. There will also be 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 additional time should be no more than (2 * baud divisor) 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. In systems where only one clock is implemented, there are no additional latencies. See Figure 2 for internal details of the Data Sync component used in the Cross Clock Module. Note that the Data Sync also contains Level Sync Modules, which can also be seen in Figure 2.

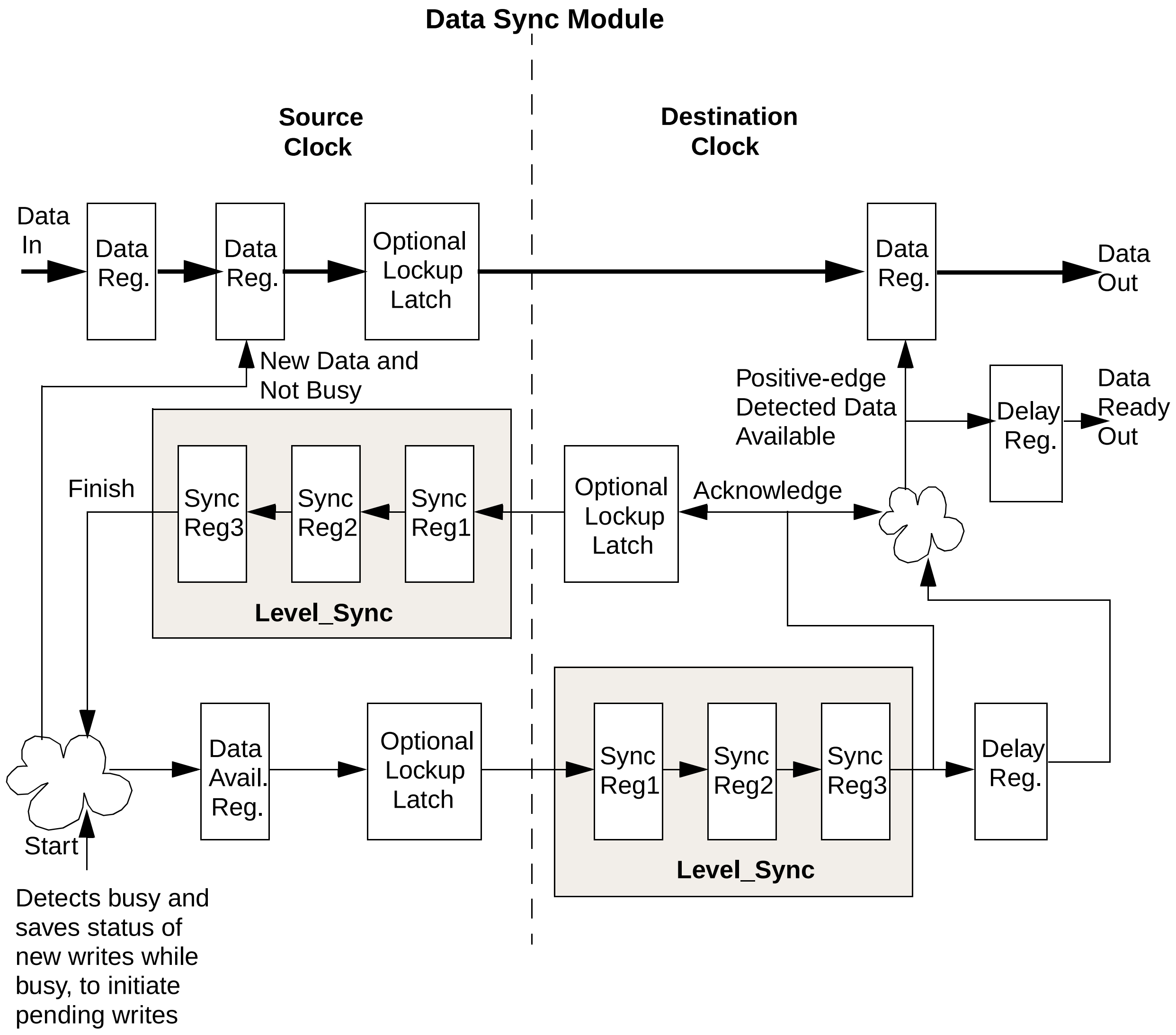


Figure 2: DW_apb_uart Data Sync Module

Shadow Registers - All serial data control registers are also forwarded across the clock boundary to the Serial Clock domain for use by the Baud Clock Generator, Character Timeout, Receiver and Transmitter modules. These registers have to be synchronized across clock boundaries for two clock systems, thus the reason for additional latencies incurred in two clock systems. See the description of the “Cross Clock Module” on page 10.

Character Timeout Detection - This module simply uses information from both clock domains to set and clear a timeout counter. This counter is then used to generate Character Timeout Interrupts when enabled. See Table 2 on page 13 for details.

Baud Clock Generator - This module uses the values in the Divisor Latch registers, sent across the Cross Clock Module, to generate the divide by 16 Baud Clock. See the Divisor Latch High and Low Register in“Programming the DW_apb_uart” on page 27 for details.

TX (Transmitter) - The transmitter converts parallel data that has been programmed into the Transmitter Holding Register into a serial data stream. This serial data stream is built according to conditions specified in the Line Control Register. See “Programming the DW_apb_uart” on page 27. The serial data then exits the design on the sout port. If FIFOs are enabled, the parallel data will be sourced from the TX FIFO. If two clocks are implemented, the data is also synchronized across the Cross Clock Synchronization module. In two clock implementations, an additional register with cross clock synchronization exists in the sclk domain. This register facilitates back-to-back transmit data with correct stop bit durations.

RX (Receiver) - The receiver converts serial data that has been sent to the DW_apb_uart on the sin port and converts it to a parallel data character, based on Line Control Register settings. See “Programming the DW_apb_uart” on page 27. Once a complete character is received, it is then sent to the RX FIFO, if FIFOs are enabled, or the Receive Buffer Register if FIFOs are disabled. If two clocks are implemented in the design, the data is also synchronized across the Cross Clock Synchronization module

Table 2: Interrupt Control Functions

FIFO Mode Only[a]	Interrupt Identification Register			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 or break interrupt	Reading the line status register
0	1	0	0	Second	Received data available	Receiver data available or read data FIFO trigger level reached	Reading the receiver buffer register or the FIFO drops below the trigger level
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
0	0	1	0	Third	Transmitter holding register empty	Transmitter holding register empty	Reading the IIR register (if source of interrupt) or writing into THR
0	0	0	0	Fourth	Modem status	Clear to send or data set ready or ring indicator or data center detect	Reading the Modem status register

a. Indicates an interrupt can only occur when the FIFOs are enabled and is used to distinguish a Character Timeout condition interrupt.

Interfacing to the DW_apb_uart

This section describes how to interface to the DW_apb_uart.

Pin Description

The following table provides a list and description of the DW_apb_uart pins:

Pin Name	Width	Direction	Function
ABP Interface			
pclk	1 bit	Input	APB clock (main interface and control clock)
presetn	1 bit	Input	An active-low, asynchronous system reset
psel	1 bit	Input	APB peripheral select
paddr	5 bits	Input	Address bus (Only uses paddr[4:0] bits)
pwrite	1 bit	Input	APB write control
penable	1 bit	Input	APB enable control that indicates the second cycle of the APB frame
prdata	*APB_DATA_WIDTH*	Input	APB read data bus (Internally, only prdata[7:0] are used. The remaining bits are padded with zeroes.)
pwdata	*APB_DATA_WIDTH*	Input	APB write data bus (Internally, only pwdata[7:0] are used)
UART Signals			
scan_mode	1 bit	Input	Active-high scan mode used to ensure that test automation tools can control all asynchronous flop signals.
sclk	1 bit	Input	Serial Interface Clock
cts_n	1 bit	Input	Clear To Send Modem Status input, active low
dsr_n	1 bit	Input	Data Set Ready Modem Status input, active low
dcd_n	1 bit	Input	Data Carrier Detect Modem Status input, active low
ri_n	1 bit	Input	Ring Indicator Modem Status input, active low

Pin Name	Width	Direction	Function
sin	1 bit	Input	Serial input, active high
dtr_n	1 bit	Output	Modem Control Data Terminal Ready output, active low
rts_n	1 bit	Output	Modem Control Request To Send output, active low
out2_n	1 bit	Output	Modem Control Programmable output 2 output, active low
out1_n	1 bit	Output	Modem Control Programmable output 1 output, active low
txrdy_n	1 bit	Output	Transmit buffer ready output, active low
rxrdy_n	1 bit	Output	Receive buffer ready output, active low
intr	1 bit	Output	Interrupt, active high
sout	1 bit	Output	Serial output, active high
baudout_n	1 bit	Output	Transmit clock output, active low
FIFO Interface			
tx_ram_out	8 bits	Input	Data to the transmit FIFO RAM
tx_ram_in	8 bits	Output	Data from the transmit FIFO RAM
tx_ram_rd_addr	$\log_2$(*FIFO_MODE*)	Output	Read address pointer for the transmit FIFO RAM
tx_ram_wr_addr	$\log_2$(*FIFO_MODE*)	Output	Write address pointer for the transmit FIFO RAM
tx_ram_we_n	1 bit	Output	Write enable for the transmit FIFO RAM, active low
tx_ram_re_n	1 bit	Output	Read enable for transmit FIFO RAM wake-up
rx_ram_out	10 bits	Input	Data to the receive FIFO RAM
rx_ram_in	10 bits	Output	Data from the receive FIFO RAM
rx_ram_rd_addr	$\log_2$(*FIFO_MODE*)	Output	Read address pointer for the receive FIFO RAM
rx_ram_wr_addr	$\log_2$(*FIFO_MODE*)	Output	Write address pointer for the receive FIFO RAM

Pin Name	Width	Direction	Function
rx_ram_we_n	1 bit	Output	Write enable for the receive FIFO RAM, active low
rx_ram_re_n	1 bit	Output	Read enable for receive FIFO RAM wake-up

Parameter Description

The following list identifies the configurable parameters supported by the DW_apb_uart:

Table 3: Top-Level Parameters

coreConsultant Field Name	Parameter Definition
APB Data Bus Width (bits)	**Parameter Name:** APB_DATA_WIDTH **Values:** 8, 16, 32 **Default Value:** 32 **Dependencies:** None **Description:** APB prdata & pwdata bus widths
Fifo Depth	**Parameter Name:** FIFO_MODE **Values:** NONE, 16, 32, ..., 2KB (2048) **Default Value:** 16 **Dependencies:** None **Description**: FIFO depth in bytes. NONE means no FIFOs.
Fifo Memory	**Parameter Name:** MEM_SELECT **Values:** External (user-supplied memory) Internal (DesignWare memory instantiation) **Default Value:** External **Dependencies:** Only changeable to Internal if (Fifo Depth != NONE) and (Fifo Depth <= 256) **Description**: Select memory.

Table 3: Top-Level Parameters (Continued)

coreConsultant Field Name	Parameter Definition
Memory Read Port Type	**Parameter Name:** MEM_MODE **Values:** Async (Asynchronous read port mode) Sync (Synchronous read port mode) **Default Value:** Async **Dependencies:** Only changeable to Sync if (Fifo Memory == External) and (Fifo Depth != NONE) **Description**: Synchronous or Asynchronous read port RAM support. When internal memory is selected (Fifo Memory == Internal), internal DesignWare D-flip-flop based RAM (DW_ram_r_w_s_dff) is created.
Asynchronous Serial Clock Support	**Parameter Name:** CLOCK_MODE **Values:** Disabled (One clock) Enabled (Two clocks) **Default Value:** Disabled (OneClk) **Dependencies:** None **Description**: When Enabled, two system clocks are implemented (sclk and pclk), otherwise pclk drives the entire design.
Test Mode Lock-up Latches	**Parameter Name:** LATCH_MODE **Values:** Exclude (No lock-up latches) Include (Lock-up latches inserted) **Default Value:** Exclude (NoLatches) **Dependencies:** Changeable to Include only when (Asynchronous Serial Clock Support == Enabled) **Description**: Clock domain signal crossing lock-up latches inserted in design. These latches bypassed when scan_mode test pin inactive.

DW_apb_uart Application Example

Figure 3 on page 18 illustrates the use of a DW_apb_uart in a typical APB Interface application. It contains a DW_apb_uart instantiation with dual ported RAMs for the TX_FIFO and RX_FIFO. Note that the entire diagram represents the DW_apb_uart with the Fifo Memory synthesis parameter set for ‘Internal’ memory support, while the dashed box represents the DW_apb_uart synthesized with ‘External’ memory selected. When ‘Internal’ memory is selected, DesignWare memories are automatically instantiated within the design.

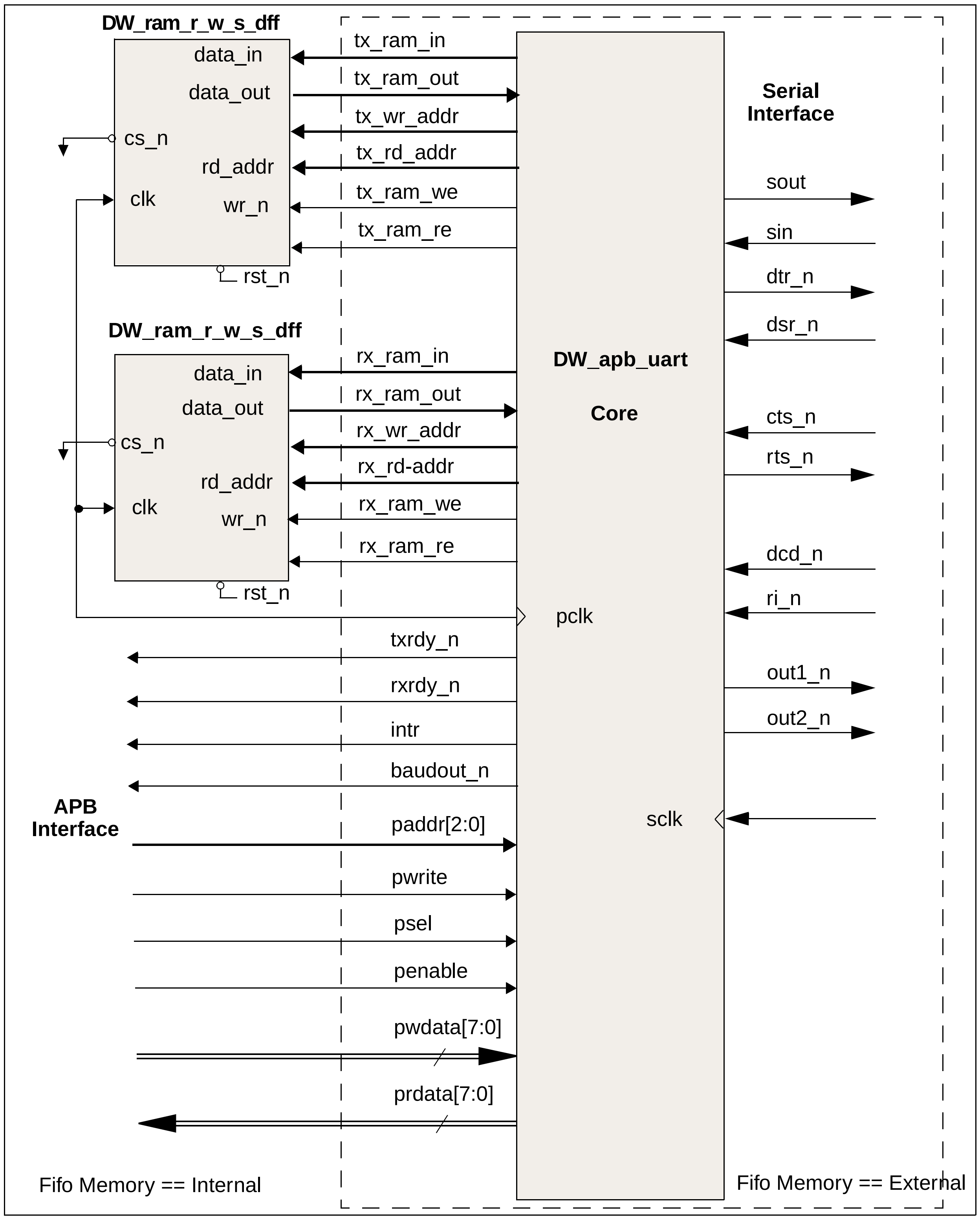


Figure 3: DW_apb_uart Application Example

Microprocessor Interface

scan_mode

The scan_mode pin enables the lock-up latches (when Test Mode Lock-up Latches = Enabled), that allow scan testing across clock domain boundaries. For scannable designs, the scan_mode pin is held active (HIGH) during testing. For normal operation, it is held inactive (LOW). When the scan_mode pin is inactive, the lock-up latches are completely bypassed, as opposed to being held open.

DW_apb_uart AMBA APB Interface

APB interface read and write operations are described as follows and shown in Figure 4:

- WRITE operation - on the rising edge of pclk, data on pwdata is written to the register at address paddr when pwrite = 1, psel = 1 and penable = 1.
- READ operation - the contents of the register at address paddr is registered out on the prdata port when pwrite = 0, psel = 1 and penable = 0. The data is then available for sampling the next clock cycle. Once the output data is valid, internal DW_uart_apb status bits are updated on the next rising edge of the pclk.

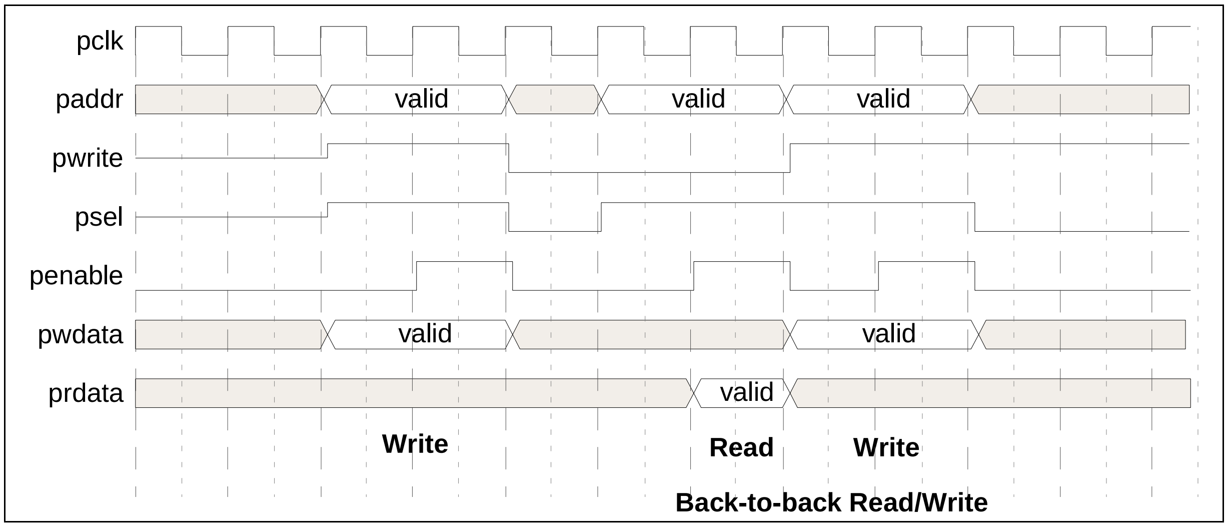


Figure 4: DW_apb_uart Register Interface Timing

The following list provides a detailed description of the APB interface signals:

paddr[4:0]

Address paddr is used to select internal DW_uart_apb registers to be written to or read from. Note that all DW_apb_uart registers are 8 bits. However, these registers are all located on 32-bit data boundaries and can only be accessed one at a time. When invalid registers are addressed, the DW_apb_uart does not respond and data is on prdata is invalid.

pwrite

The pwrite signal is used to select between register writes and reads. When the pwrite signal is (HIGH), in conjunction with psel and penable also being active (HIGH), data on pwdata is written to the register at address paddr. When the pwrite signal is (LOW), in conjunction with psel and penable also being active (HIGH), data is read from the UART register selected by address paddr.

psel

The psel signal selects the APB peripheral for a data transaction and must remain active (HIGH) through an entire write or read operation. Each APB operation consists of two APB clock (pclk) cycles. The first clock cycle of the psel signal being active while the penable is still inactive, is called the SETUP state.

penable

The second clock cycle of an APB data transaction is called the ENABLE state. During this time, the penable signal becomes active (HIGH) and in the case of a ‘write’, the data is written to the internal registers on the next positive edge of the cpu_clk. In the case of a data read, read data is valid on prdata during this cycle. The psel signal must remain active (HIGH) throughout the transaction.

pwdata[APB_DATA_WIDTH-1:0]

The pwdata bus provides the data path for control bits and data from the CPU to the input side of the UART. The data is written to registers as described above. Only pwdata[7:0] is used internally and all other bits are ignored.

prdata[APB_DATA_WIDTH-1:0]

The prdata bus provides status bits and data from the output side of the UART to the CPU. Data is read from registers as described above. Only prdata[7:0] contains active data and all other bits are padded with logic ‘0’s.

baudout_n

The baudout_n signal is the 16 × clock signal from the transmitter section of the UART. A timing example is shown in Figure 5.

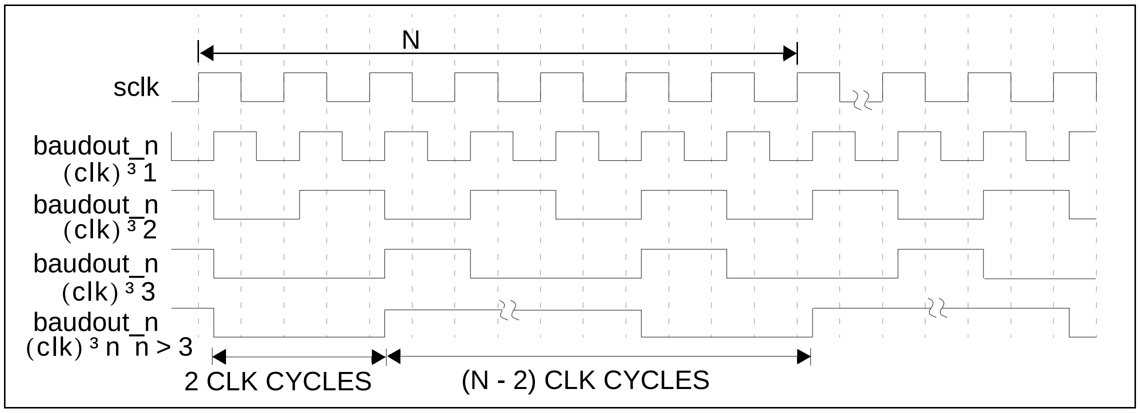


Figure 5: Baudout Timing Waveform

DMA Signalling with txrdy_n and rxrdy_n

DMA signalling is available through the trxrdy_n and rxrdy_n outputs. There are two DMA modes: mode 0 and mode 1, controllable via bit 3 of the FIFO Control Register (only DMA mode 0 is available when the FIFOs are disabled).

DMA mode 0 supports single DMA data transfers at a time. In mode 0, the txrdy_n signal goes active low when the Transmitter Holding Register is empty in non-FIFO mode or when the transmitter FIFO is empty in FIFO mode. It goes inactive when a single character has been written into the Transmitter Holding Register or transmitter FIFO. The rxrdy_n signal goes active low when there is a single character available in the Receiver FIFO or the Receive Buffer Register and it goes inactive when the Receive Buffer Register and Receiver FIFO are empty, depending on FIFO mode.

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 txrdy_n signal is asserted when the Transmit FIFO is empty and is de-asserted when it is completely full. The rxrdy_n signal is asserted when the Receiver FIFO trigger level is reached, or a character timeout has occurred, and is de-asserted when the receiver FIFO becomes empty.

intr

The intr output signal goes active high whenever an enabled and active interrupt has occurred. The following interrupts are enabled via the Interrupt Enable Register (IER):

- Receive Error
- Received Data Available
- Character Time-out (FIFO mode only)
- Transmitter Holding Register empty
- Modem Status

An interrupt must be enabled and the condition active to drive intr HIGH. The intr signal is driven LOW when rst is applied. Details of interrupt operation are contained in Table 2 on page 13.

Serial Interface

sin

The sin signal is the serial data input from the modem, data set, or peripheral device.

sout

The sout signal is the serial data output to the modem, data set, or peripheral device. The sout signal is set HIGH when the reset (preset) is applied. In Loopback mode, sout is held at a steady HIGH.

cts_n

The Clear To Send (cts_n) modem status input, when active low, indicates that the modem or data set is ready to exchange data. The cts_n signal can be checked by the CPU by reading MSR bit 4 (CTS), which is the compliment of cts_n. Bit 0 of the MSR (DCTS) indicates whether cts_n has changed state since the previous reading of the MSR. Whenever the DCTS bit of the MSR changes state, an interrupt is generated if the Modem Status Interrupt is enabled. In Loopback mode, this input signal is disconnected.

dsr_n

The Data Set Ready (dsr_n) modem status input, when active low, indicates that the modem or data set is ready to establish communications with the UART. The dsr_n signal can be checked by the CPU by reading MSR bit 5 (DSR), which is the compliment of dsr_n. Bit 1 of the MSR (DDSR) indicates whether dsr_n changed state since the previous reading of the MSR. Whenever the DDSR bit of the MSR changes state, an interrupt is generated if the Modem Status Interrupt is enabled. In Loopback mode, this input signal is disconnected.

dcd_n

The Data Carrier Detect (dcd_n) modem status input, when active low, indicates that the carrier has been detected by the modem or data set. The dcd_n signal can be checked by the CPU by reading MSR bit 7 (DCD), which is the compliment of dcd_n. Bit 3 of the MSR (DDCD) indicates whether dcd_n changed state since the previous reading of the MSR. Whenever the DDCD bit of the MSR changes state, an interrupt is generated if the Modem Status Interrupt is enabled. In Loopback mode, this input signal is disconnected.

ri_n

The Ring In (ri_n) modem status input, when active low, indicates that a telephone ringing signal has been received by the modem or data set. The ri_n signal can be checked by the CPU by reading MSR bit 6 (RI), which is the compliment of ri_n. Bit 2 of the MSR (TERI) indicates whether ri_n has changed from LOW to HIGH since the previous reading of the MSR. Whenever the TERI bit of the MSR changes from LOW to HIGH, an interrupt is generated if the Modem Status Interrupt is enabled. In Loopback mode, this input signal is disconnected.

dtr_n

The Data Terminal Ready (dtr_n) modem status output, when active low, informs the modem or data set that the UART is ready to establish communications. The dtr_n signal is set LOW by programming MCR bit 0 (DTR) to a HIGH. The dtr_n signal is driven inactive HIGH when the reset (preset) is applied to the UART. In Loopback mode, this output is held inactive HIGH.

rts_n

The Request To Send (rts_n) modem status output, when active low, informs the modem or data set that the UART is ready to send data. The rts_n signal is set LOW by programming MCR bit 1 (RTS) to a HIGH. The rts_n signal is driven inactive HIGH when the reset (preset) is applied. In Loopback mode, this output is held inactive HIGH.

out1_n

The out1_n signal is a user-designated modem status output that is set active LOW by programming MCR bit 2 (OUT1) to a HIGH. The out1_n signal is driven inactive HIGH when the reset (preset) is applied. In Loopback mode, this output is held inactive HIGH.

out2_n

The out2_n signal is a user-designated modem status output that is set active LOW by programming MCR bit 3 (OUT1) to a HIGH. The out2_n signal is driven inactive HIGH when the reset (preset) is applied. In Loopback mode, this output is held inactive HIGH.

FIFO Interface

The FIFO Interface only exists in the design when (Fifo Depth != NONE) and (Fifo Memory == External). Otherwise, all ports are removed

The tx_ram_in signal is an 8-bit output data bus from the DW_apb_uart core to the transmit FIFO. The Transmit FIFO temporarily stores output data before it is sent back to the UART along the tx_ram_out bus for subsequent transmission out the serial sout line.

The tx_ram_out signal is an 8-bit input data bus from the transmit FIFO. This signal carries the data to be transmitted out the serial sout line.

The tx_ram_rd_addr signal is the read address pointer for the transmit FIFO RAM.

The tx_ram_wr_addr signal is the write address pointer for the transmit FIFO RAM.

The tx_ram_we_n signal is the write enable, active low, for the transmit FIFO. Data on tx_ram_in is written to the transmit FIFO when this signal is LOW.

The tx_ram_re_n signal is simply available to ‘wake up’ the transmitter FIFO memory on a FIFO read, for low power savings.

The rx_ram_in signal is a 10-bit output data bus from the DW_apb_uart core to the receive FIFO. The receive FIFO temporarily stores data received from a peripheral device until the CPU is ready to read the data. The data in the receive FIFO is sent to the UART via the rx_ram_out data bus.

The rx_ram_out signal is a 10-bit input data bus from the receive FIFO to the DW_apb_uart core. The rx_ram_out signal carries the data that is to be read by the CPU.

The rx_ram_rd_addr signal is the read address pointer for the receive FIFO RAM.

The rx_ram_wr_addr signal is the write address pointer for the receive FIFO RAM.

The rx_ram_we_n signal is the write enable, active low, for the receive FIFO RAM. Data on rx_ram_in is written to the receive FIFO when this signal is LOW.

The rx_ram_re_n signal is simply available to ‘wake up’ the receiver FIFO memory on a FIFO read, for low power savings.

Reset

When presetn is active (LOW), the UART outputs and registers are set according to the values in Table 4.

Table 4: UART Reset Status

Register/Signal	Reset State
Receive Buffer Register	0000 0000
Transmit Holding Register	0000 0000
Divisor Latch Low	0000 0000
Divisor Latch High	0000 0000
Interrupt Enable Register	0000 0000
Interrupt Identification Register	0000 0001
FIFO Control	0000 0000
Line Control Register	0000 0000
Modem Control Register	0000 0000
Line Status Register	0110 0000
Modem Status Register	0000 0000
Scratch Register	0000 0000
sout	High
intr (Receiver Errors)	Low
intr (Receiver Data Ready)	Low
intr (THRE)	Low
intr (Modem Status Changes)	Low
out1	High
out2	High
rts_n	High
dtr_n	High
Receiver FIFO	Cleared
Transmit FIFO	Cleared

Making the DW_16550 Scannable

The DW_apb_uart can be made scannable by via DFT Compiler. Use the `set_test_hold 1 scan_mode` command before `insert_scan`.

When performing check_test using DFT Compiler, setting scan_mode to ‘1’ allows the DFT Compiler to reach each flip-flop in the design for test purposes. When tied low, synthesis optimizes the design, and does not connect the scan_mode pin.

Note

For scannable designs, scan_mode should be active (HIGH) only when scan enable is active.

Programming the DW_apb_uart

This section describes the programable features of the DW_apb_uart.

Register Address Map and Description Summary

The following table summarizes the register address map for the DW_apb_uart:

DLAB[a]	Address	Access	Name	Abbreviation
0	0x00	read-only	Receive Buffer Register	RBR
0		write-only	Transmit Holding Register	THR
1		read/write	Divisor Latch (Low)	DLL
0	0x04	read/write	Interrupt Enable Register	IER
1		read/write	Divisor Latch (High)	DLH
X	0x08	read-only	Interrupt Identity Register	IIR
X		write-only	FIFO Control Register	FCR
X	0x0C	read/write	Line Control Register	LCR
X	0x10	read/write	Modem Control Register	MCR
X	0x14	read-only	Line Status Register	LSR
X	0x18	read-only	Modem Status Register	MSR
X	0x1C	read/write	Scratch Register	SCR

a. DLAB is bit 7 of the Line Control Register (LCR). It enables reading and writing of the Divisor Latch Registers (DLL and DLH) to set the baud rate of the UART. See the description for "Line Control Register (LCR)" on page 30.

Register Addressing

The DW_apb_uart has 12 internal registers that are accessed via the 5-bit address bus. The register addressing is described in "Register Address Map and Description Summary" on page 27.

The Divisor Latch Address Bit (DLAB) is the MSB of the Line Control Register (LCR). This bit must be set in order to address the Divisor Latch High (DLH) and Divisor Latch Low (DLL) registers.

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.

Register Bit Map Description

The following paragraphs describe the data fields of the DW_apb_uart registers. A register bit map is shown in Table 5 on page 33. Details of interrupt operation are contained in Table 2 on page 13.

Receive Buffer Register (RBR)

The RBR is a read-only register that contains the data byte received on the serial input port (sin). The data in this register is valid only if the Data Ready (DR) bit in the Line status Register (LSR) is set. In the non-FIFO mode (Fifo Depth == NONE) or when the FIFOs are programmed OFF, the data in the RBR must be read before the next data arrives, otherwise it will be overwritten, resulting in an overrun error. In the FIFO mode (Fifo Depth != NONE) and FIFOs programmed ON, 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 will be preserved but any incoming data will be lost. An overrun error will also occur.

Transmit Holding Register (THR)

The THR is a write-only register that contains data to be transmitted on the serial output port (sout). Data can be written to the THR any time that the THR Empty (THRE) bit of the Line Status Register (LSR) is set.

If FIFOs are not enabled 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 FIFOs are enabled and THRE is set, up to 16 characters of data may be written to the THR before the FIFO is full. Any attempt to write data when the FIFO is full results in the write data being lost.

Divisor Latch High and Low (DLH, DLL)

The DLH (Divisor Latch High) register in conjunction with DLL (Divisor Latch Low) register forms a 16-bit, read/write, Divisor Latch register that contains the baud rate divisor for the UART. It is accessed by first setting the DLAB bit (bit 7) in the Line

Control Register (LCR). The output baud rate is equal to the serial clock (pclk if one clock design, sclk if Asynchronous Serial Clock Support == Enabled) frequency divided by sixteen times the value of the baud rate divisor, as follows:

```
baud rate = (serial clock freq) / (16 * divisor)
```

Interrupt Enable Register (IER)

The IER is a read/write register that contains four bits that enable the generation of interrupts. These four bits are the Enable Received Data Available Interrupt (ERBFI), the Enable Transmitter Holding Register Empty Interrupt (ETBEI), the Enable Receiver Line Status Interrupt (ELSI), and the Enable Modem Status Interrupt (EDSSI). See Table 2 on page 13 for details of interrupt operation.

Interrupt Identity Register (IIR)

The Interrupt Identity Register is a read-only register that identifies the source of an interrupt. The upper two bits of the register are FIFO-enabled bits. These bits will be "00" if the FIFOs are disabled, and "11" if they are enabled. The lower four bits identify the highest priority pending interrupt. For details see Table 2 on page 13 and Table 5 on page 33.

FIFO Control Register (FCR)

The FIFO control register is a write-only register. If (Fifo Depth == NONE), this register has no effect. Otherwise, this register controls the read and write data FIFO operation and the mode of operation for the DMA signals txrdy_n and rxrdy_n.

Setting bit 0 of the FCR enables the transmit and receive FIFOs.

Writing a 1 value to bit 1 of the FCR resets and flushes data in the receive FIFO. Note that this bit is 'self-clearing'and it is not necessary to clear this bit.

Writing a 1 value to bit 2 of the FCR resets and flushes data in the transmit FIFO. Note that this bit is 'self-clearing'and it is not necessary to clear this bit.

The FIFOs are also reset anytime bit 0 of the FCR changes value.

If the FIFO mode is enabled (Fifo Depth != 1 and bit 0 of the FCR is set to 1) bits 3, 6, and 7 are active. Bit 3 determines the DMA signalling mode for txrdy_n and rxrdy_n output signals; see the description in "DMA Signalling with txrdy_n and rxrdy_n" on page 21 for details. As described in Table 5 on page 33, bit 6 and bit 7 set the trigger level in the receiver FIFO for both the rxrdy_n signal and the Enable Received Data Available Interrupt (ERBFI).

Line Control Register (LCR)

The Line Control Register controls the format of the data that is transmitted and received by the DW_apb_uart.

LCR bits 0 and 1 control the number of bits per character. Details are shown in Table 5 on page 33.

LCR bit 2 controls the number of stop bits transmitted. If bit 2 is a logic ‘0’, one stop bit is transmitted in the serial data. If bit 2 is a logic ‘1’ and the data bits are set to 5, one and a half stop bits are generated. Otherwise, two stop bits are generated and transmitted in the serial data out

LCR bit 3 is the Parity Enable bit. Parity is enabled when this bit is set.

LCR bit 4 is the Parity Select bit. If parity is enabled, bit 4 selects between even and odd parity. If bit 4 is a logic ‘1’, an even number of logic ‘1’s is transmitted or checked. If bit 4 is a logic ‘0’, an odd number of logic ‘1’s is transmitted or checked.

LCR bit 5 is the Stick Parity bit. The DW_apb_uart does not support the “stick parity” feature so bit 5 of the LCR has no affect.

LCR bit 6 is the Break Control bit. Setting the Break bit sends a break signal by holding the sout line low (when not in Loopback Mode, as determined by Modem Control Register bit 4), until the Break bit is cleared. When in Loopback Mode, the break condition is internally looped back to the receiver

LCR bit 7 is the Divisor Latch Address bit. Setting this bit enables reading and writing of the Divisor Latch register (DLL and DLH) to set the baud rate of the UART. This bit must be cleared after initial baud rate setup in order to access other registers.

Modem Control Register (MCR)

The lower four bits of the MCR directly manipulate the outputs of the DW_apb_uart. Details are shown in Table 5 on page 33. The DTR (bit 0), RTS (bit 1), OUT1 (bit 2), and OUT2 (bit 3) bits are inverted and then drive the corresponding UART outputs, dtr_n, rts_n, out1_n and out2_n.

MCR 4 is the LOOPBACK bit. When set, 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. This feature is used for diagnostic purposes. Also, in loopback mode, the modem control inputs (dsr_n, cts_n, ri_n, dcd_n) are disconnected and the four modem control outputs (dtr_n, rts_n, out1_n, out1_n) are looped back to the inputs, internally.

Line Status Register (LSR)

The Line Status Register contains status of the receiver and transmitter data transfers. This status can be read by the programmer at anytime. Details are shown in Table 5 on page 33.

LSR bit 0 is the Data Ready (DR) bit. When set, this bit indicates the receiver contains at least one character in the RBR or the receiver FIFO. This bit is cleared when the RBR is read in the non-FIFO mode, or when the receiver FIFO is empty, in the FIFO mode.

LSR bit 1 is the overrun error (OE) bit. When set, this bit indicates an overrun error has occurred because 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.

LSR bit 2 is the Parity Error (PE) bit. This bit is set whenever there is a parity error in the receiver if the Parity Enable (PEN) bit in the LCR 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.

LSR bit 3 is the Framing Error (FE) bit. This bit is set whenever there is 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.

The OE, PE and FE bits are reset when a read of the LSR is performed.

LSR bit 4 is the Break Interrupt (BI) bit. This bit 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*. A break condition on sin causes one and only one character, consisting of all zeros, to be received by the UART. In the 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 the non FIFO mode, the BI indication occurs immediately and persists until the LSR is read.

LSR 5 is the Transmitter Holding Register Empty (THRE) bit. When set, this bit indicates the DW_apb_uart can accept a new character for transmission. This bit is set whenever data is transferred from the THR to the transmitter shift register and no new data has been written to the THR. This also causes a THRE Interrupt to occur, if the THRE Interrupt is enabled.

LSR bit 6 is the Transmitter Empty (TEMT) bit. In the FIFO mode, this bit is set whenever the Transmitter Shift Register and the FIFO are both empty. In the non-FIFO mode, this bit is set whenever the Transmitter Holding Register and the Transmitter Shift Register are both empty.

LSR bit 7 is the Error in Receiver FIFO (FERR) bit. This bit is only active when FIFOs are enabled. It is set when there is at least one parity error, framing error, or break indication in the 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.

Modem Status Register (MSR)

The Modem Status Register contains the current status of the modem control input lines and if they changed. Details are shown in Table 5 on page 33.

DCTS (bit 0), DDSR (bit 1) and DDCD (bit 3) bits record whether the modem control lines (cts_n, dsr_n and dcd_n) have changed since the last time the CPU read the MSR. TERI (bit 2) indicates ri_n has changed from an active low, to an inactive high state since the last time the MSR was read. In Loopback Mode, DCTS reflects changes on MCR bit 1 (RTS), DDSR reflects changes on MCR bit 0 (DTR) and DDCD reflects changes on MCR bit 3 (Out2), while TERI reflects when MCR bit 2 (Out1) has changed state from a high to a low.

The CTS, DSR, RI and DCD Modem Status bits contain information on the current state of the modem control lines. CTS (bit 4) is the compliment of cts_n, DSR (bit 5) is the compliment of dsr_n, RI (bit 6) is the compliment of ri_n and DCD (bit 7) is the compliment of dcd_n. In Loopback Mode, CTS is the same as MCR bit 1 (RTS), DSR is the same as MCR bit 0 (DTR), RI is the same as MCR bit 2 (Out1) and DCD is the same as MCR bit 3 (Out2).

Scratchpad Register (SCR)

The SCR register is an 8-bit read/write register for programmers to use as a temporary storage space. It has no defined purpose in the DW_apb_uart.

Table 5: Register Bit Map

Reg	7	6	5	4	3	2	1	0
IER	0				EDSSI	ELSI	ETBEI	ERBFI
IIR	00 FIFO Disabled[a] 11 FIFO Enabled		0		Interrupt ID 0000 Modem Status Changed 0001 No interrupt pending 0010 THR empty 0100 Received Data available 0110 Receiver Status 1100 Character Time Out			
FCR	RCVR Trigger 00=1 char in FIFO 01=FIFO 1/4 full 10=FIFO 1/2 full 11=FIFO 2 less full		Reserved		DMA Mode	XMIT FIFO Reset	RCVR FIFO Reset	FIFO Enable
LCR	DLAB	Break	Stick Parity (not used)	EPS	PEN	STOP bits 0=1 bit 1=2 bits	CLS 00=5 bits 01=6 bits 10=7 bits 11=8 bits	
MCR	0			LoopBack	OUT2	OUT1	RTS	DTR
LSR	RX FIFO Error	TEMT[b]	THRE[b]	BI	FE	PE	OE	DR
MSR	DCD	RI	DSR	CTS	DDCD	TERI	DDSR	DCTS

a. Default in FIFO mode with FIFOs disabled upon reset.

b. Value upon reset equals 1 - All other bits are equal to 0 upon reset.

2 Verification of DW_apb_uart

This chapter provides an overview of the testbench available for DW_apb_uart verification. Once the DW_apb_uart has been configured and the verification environment set up, simulations can be automatically ran. DW_apb_uart verification is detailed in the following section:

- "Overview of DW_apb_uart Testbench"
- "Overview of VERA Tests" on page 38

Overview of DW_apb_uart Testbench

As illustrated in Figure 6 on page 36, the DW_apb_uart verification environment consists of a Verilog testbench that includes an instantiation of the design under test (DUT), a Verilog Serial BFM and a VERA shell. The Serial BFM generates serial data to, and captures serial data from the DUT. The Serial BFM is first programmed with serial configurations to match the DUT, via Vera tasks. Then, Vera tasks are used to send data to and capture data from the DUT.

The VERA shell consists of an APB Master bus functional model (BFM), an APB Monitor, test stimuli, BFM configuration, and test results. As mentioned above, Vera tasks in the shell are used to interact with the Serial BFM. In addition, the APB BFM is configured and used by the test bench to program DUT configuration registers and read received data and write transmit data to the DUT. The APB Monitor oversees APB activity between the Vera test stimuli and the DUT and reports any errors that occur. Note that no errors do occur during test bench operation and none have been emulated for DW_apb_uart verification.

The testbench stimuli tests all DW_apb_uart features, regardless of user configurations specified in the Specify Configuration task of coreConsultant, with one exception. Various tests require FIFOs to be enabled for verification. Tests that require the use of

FIFOs, will be unavailable for selection in the Verification setup in coreConcultant, if Fifo Depth has been set to 'NONE' in the configuration task. The testbench also tests that the component is AMBA compliant and includes a self-checking mechanism.

Finally, test results are generated and recorded as either 'Pass' or 'Fail' in log files. There is both a common log file containing test results for all tests ran, as well as individual logs detailing each test run. Waveform files are also generated in either VCD or VPD format, depending on the simulator chosen.

More detail on simulators and output format can be found in the following subsections:

- "Supported Simulators" on page 37
- "Simulation Output Files" on page 38

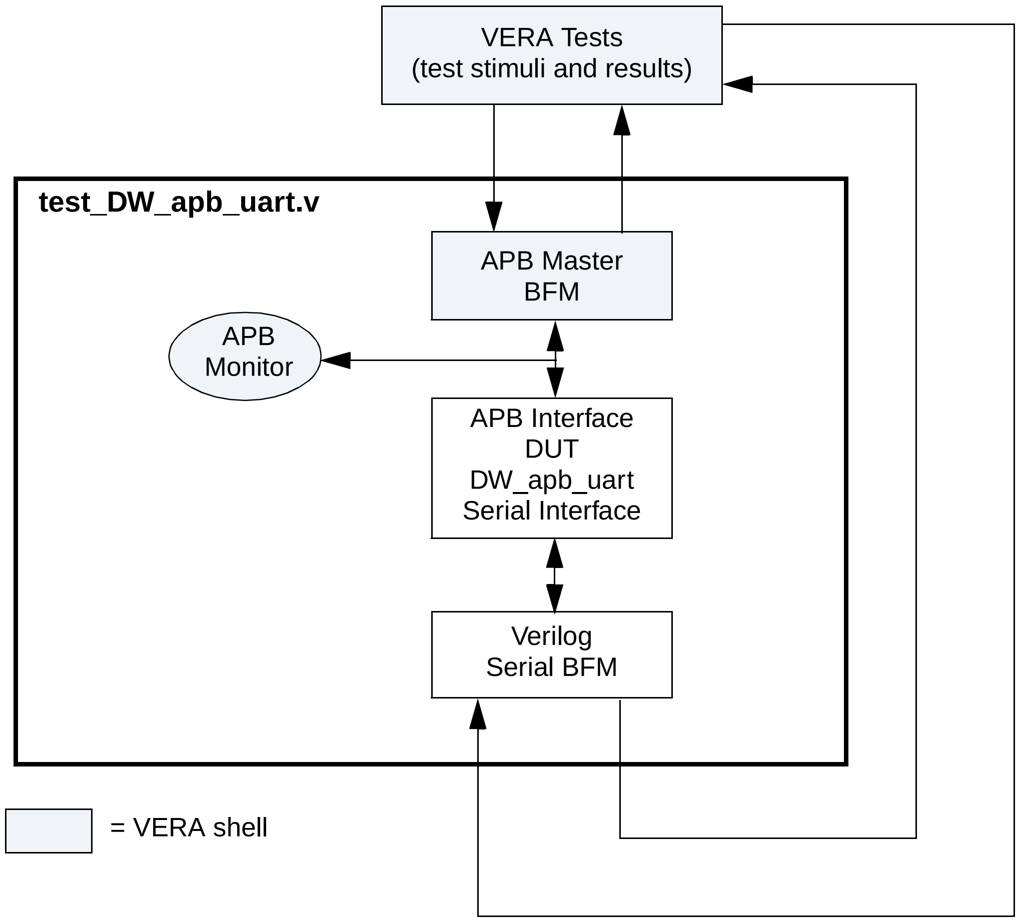


Figure 6: DW_apb_uart Testbench

Supported Simulators

The DW_apb_uart testbench supports the following simulators, which are configured via the Simulator Setup tab of the Verification activity in coreConsultant. For specific versions of these tools, refer to the DW_apb Release Note (dw_ahb_rn.pdf).

- Synopsys VCS
- Cadence Verilog-XL
- Cadence NC-Verilog
- ModelSim/VLOG

The simulation generates encrypted RTL source files, which can be read by Synopsys VCS. For the other simulators, technology independent GTECH source is generated.

Running Simulations from the Command Line

The DW_apb_uart does not have a simulation model. When running simulations with coreConsultant or coreAssembler, a simulation model is generated automatically, before simulations are run. However, to run simulations from a UNIX command line, a simulation model must be created manually, as follows:

Change your working directory to *workspace*/sim/qmap and then execute the following command:

```
run 'dc_shell -f mkapb.scr.danger'
```

Note that your SYNOPSYS variable and path must be set to a dc_shell install that supports the DW_apb_uart component. Please contact Synopsys Customer Support for further assistance.

Once the DW_apb_uart.v simulation model has been created, it can be used in a stand-alone test bench, or with the simulations that are packaged in the core kit.

To run the packaged tests, change the working directory to DW_apb_uart/sim and run the following:

```
runtest -test test_name
```

where *test_name* is the name of each test and the sub directory where each test is located. For example, to run the simple register write/read test, run the following:

```
runtest -test test_reg_wr_rd
```

The results of running tests via the command line are only available in each test directory, in the test.log file.

Simulation Output Files

The simulation.log file located in *workspace*/sim/ provides a pass/fail result for all tests ran in a particular simulation. A test.log file in *workspace*/sim/test_<name> provides a more detail on each specific test simulation, in addition to the pass/fail status. The waveforms are also written to this directory.

If the simulation results match expected results, the simulation completes successfully and the simulation status in the test.log file is PASSED. If the simulation results do not match expected results, the simulation terminates and the simulation status in the test.log file is FAILED.

Overview of VERA Tests

The DW_apb_uart peripheral incorporates numerous operational features. Many of these features, having related operational characteristics, are combined into one test to reduce simulation time. While some the tests in the following categories do have some overlap, they are organized as follows:

- “Basic Register Functionality” on page 39
- “Serial Transmission, reception and Functionality” on page 39
- “FIFO Functionality” on page 39
- “Interrupt and DMA Functionality” on page 40
- “Modem Status and Control” on page 41

A brief description of each test appears in the corresponding sections. A detailed description of each test, outlining specific transactions, appears in the ‘README’ file located in the workspace/sim/ directory.

Note

All tests use the APB Interface to program memory mapped registers dynamically during tests.

Basic Register Functionality

test_reg_wr_rd - This test verifies DW_apb_uart register values after a system reset, as well as performing various APB register writes and reads.

Serial Transmission, reception and Functionality

test_baudout_n - This test verifies the baud divisor/baudout_n clock over a range of values.

test_break_ctrl - This test verifies the break control feature (Line Control register bit 6) forces and holds sout ‘low’.

test_half_stop - This test simply allows visibility of a half stop bit for confirmation.

test_ser_rx - This test uses the Serial BFM to generate one serial data bitstream character to the uart and checks its valididty.

test_ser_tx - This test transmits one character from the uart, which is then captured by the Serial BFM and checked for correctness in the test bench.

t**est_extended_serial_io -** This test extensively exercises both the transmitter and receiver, in non Loopback Mode, with a Baud Divisor of 0x0001.

test_line_ctrl_loopback - This test exercises both the transmitter and receiver, in Loopback Mode, with a Baud Divisor of 0x0001.

FIFO Functionality

test_rxfifo_disabled - This test verifies the FIFOs cannot be enabled when FIFO Control register bit 0 is not set. This test also verifies a Receiver Overrun, since no RX FIFO is enabled.

test_rxfifo_error_seq - This test verifies correct Receiver Errors sequence properly through the RX FIFO. NOTE: This test is not ran when (Fifo Depth == NONE).

test_txfifo_disabled - This test verifies that the TX FIFO is not active when the FIFO Control Register is set for DMA Mode 1, but the enable bit (FIFO Control Register bit 0) is not set.

Interrupt and DMA Functionality

test_break_intr - This test verifies correct receiver break interrupt generation when a break condition (static ‘0’) is applied to sin.

test_char_timeout - This test verifies proper operation of the Character Timeout feature and interrupt. NOTE: This test is not ran when (Fifo Depth == NONE).

test_dma0_full_rxfifo - This test verifies Line Status DR and DMA RXRDY perform as expected when receiving data, both with FIFOs enabled and without, in DMA Mode 0. NOTE: This test is not ran when (Fifo Depth == NONE).

test_dma0_tx_intr - This test verifies proper TXRDY and Transmitter Empty Interrupt operation, both with FIFOs enabled and without, in DMA Mode 0. NOTE: This test is not ran when (Fifo Depth == NONE).

test_dma1_rx_thresh_intr - This test verifies that Line Status, RXRDY, Receive Data Available Interrupt and FIFOs perform as expected when the FIFOs are set for DMA Mode 1, and the RX FIFO is filled to all Threshold limits. NOTE: This test is not ran when (Fifo Depth == NONE).

test_dma1_tx_intr - This test verifies proper TXRDY and Transmitter Empty Interrupt operation as the TX FIFO is filled and emptied, in DMA Mode 1. NOTE: This test is not ran when (Fifo Depth == NONE).

test_framing_error_intr - This test verifies correct receiver Framing Error Interrupt generation.

test_intr_priority - This test ’minimally’ verifies correct interrupt priority handling by walking up and down interrupts in priority order.

test_overrun_error_intr - This test verifies correct Receiver Overrrun Error (OE) Interrupts. NOTE: This test is not ran when (Fifo Depth == NONE).

test_parity_error_intr - This test verifies correct Receiver Parity Interrupt generation.

test_rxdata_avail_intr - This test verifies correct Received Data Avaliable (RDA) Interrupts.

test_txreg_empty_intr - This test verifies correct Transmitter Holding Register Empty (THRE) interrupts with the FIFOs disabled.

Modem Status and Control

test_modem_control - This test verifies the Modem Control Outputs toggle correctly, based on the Modem Control Register bits.

test_modem_loopback - This test verifies the Modem Control Loopback feature by checking that sout remains inactive high, while a transmitted character is looped back to the receiver. Also verifies that sin is ignored internaly, during loopback mode.

test_modem_status - This test verifies correct Modem Status Register functionality, while the Modem Status input pins to the DW_apb_uart cycle through all possible states.

3 Synthesis of DW_apb_uart

Recommended Synthesis Strategy

The following are specified as the default synthesis strategy options for DesignWare AMBA Bus IP that can be specified in coreConsultant:

Timing constraints: pclk period = 47ns, sclk period = 135ns (default). The DW_apb_uart will not operate correctly when the sclk period is less than 1/4th the pclk period.

Compile effort: medium-effort (default)

Multiple clocks: The clocks are pclk and sclk.

Preserve Hierarchy: False

Multicycle paths: None

Timing exceptions:

set_false_path -from sclk -to pclk

set_false_path -from pclk -to sclk

Clock gating: Allowed.

Uniquify: True.

Test Ready: No (can be enabled)

False paths: There are only two false paths, one is from pclk to sclk and other is from sclk to pclk.

Overview of Synthesis Scripts

The top-level script that is executed in the Synthesis process is either initial.dcsh, incr1.dcsh, or incr2.dcsh depending on the Quality of Result (QoR) level you set in coreConsultant. These scripts are located in *workspace*/syn. If QoR low is selected, initial.dcsh will be created and executed; if QoR medium is selected, incr1.dcsh is created and executed; and if QoR high is selected, incr2.dcsh is created and executed.

This top-level script analyzes and elaborates all the RTL files and then sources the top-level constraint file, DesignWare AMBA Bus IP.cscr, which resides in *workspace*/syn/constrain/script. The top-level script (.dcsh) is encrypted, whereas all of the constraint files (*.cscr) are unencrypted.

Running Synthesis from Command Line

To run synthesis from the command line prompt for the files generated by coreConsultant, enter the following command:

```
% run.scr
```

This script resides in your workspace/syn directory according to what you specified as the QoR Effort in the **Synthesize Activity | Strategy Options** activity. The default choice is medium, which creates an incr1 directory. If you choose high, incr2 is created. So for instance, if you chose the default setting in coreConsultant, you would enter the command from *workspace*/syn/incr1. However, the *workspace*/syn/final directory is a pointer to the latest scripts generated.

A APB Slave Interface

Reading and Writing from an APB Slave

When writing to and reading from DesignWare APB slaves, you should consider the following:

- The size of the APB peripheral should always be set equal to the size of the APB data bus.
- The APB bus has no concept of a transfer size or a byte lane, unlike the DW_ahb.
- The APB slave subsystem is little endian; the DesignWare AMBA Bus IP performs the conversion from a big endian AHB to the little endian APB.
- All APB slave programming registers are aligned on 32-bit boundaries, irrespective of the APB bus size. Refer to Figure 7 on page 46.
- The maximum APB_DATA_WIDTH is 32 bits. Registers larger than this will occupy more than one location in the memory map.
- The DW_apb does not return any ERROR, SPLIT, or RETRY responses, always an OKAY response.
- For all bus widths:
  - In the case of a read, registers less than the full bus width will return zeros in the unused upper bits.
  - Writing to bit locations larger than the register width does not have an effect on anything. Only the pertinent bits are written to the register.
- The APB slaves do not need the full 32-bit address bus, paddr. The slaves include the lower bits even though they are not actually used in a 32- or 16-bit system.

The following sections (as illustrated in Figure 7) show the relationship between the register map and the read/write operations for the three possible APB_DATA_WIDTH values: 8-, 16-, and 32-bit APB buses.

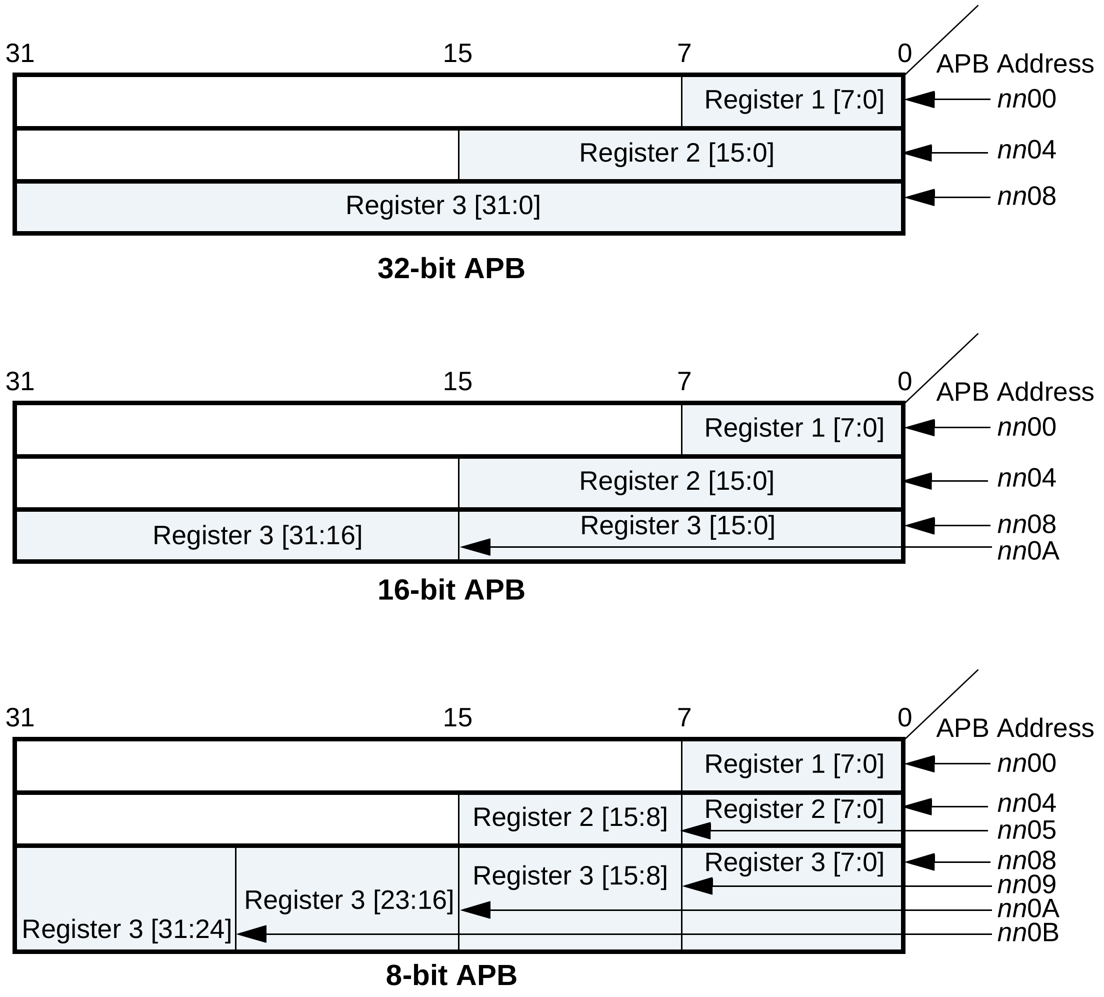


Figure 7: Read/write Locations for Different APB Bus Widths

32-bit Bus System

For 32-bit bus systems, all programming registers can be read or written with one operation as illustrated in Figure 7.

Because all registers are on 32-bit boundaries, paddr[1:0] is not actually needed in the 32-bit bus case. But these bits still exist in the configured code for usability purposes. If the user writes to an address location not on a 32-bit boundary, the bottom bits are ignored/not used.

16-bit Bus System

For 16-bit bus systems, two scenarios exist as illustrated in Figure 7 on page 46:

1. The register to be written to/read from is less than 16 bits

   In this case, the register can be read or written with one transaction. In the case of a read, registers less than 16 bits wide will return zeros in the un-used bits. Writing to bit locations larger than the register width will cause nothing to happen, i.e. only the pertinent bits are written to the register.

2. The register to be written to/read from is >16 and <= 32 bits

   In this case, two transactions are required to read or write the register. The first transaction should read/write the lower two bytes (half-word) and the second transaction the upper half-word.

Because the bus is reading a half-word at a time, paddr[0] is not actually needed in the 16-bit bus case. But these bits still exist in the configured code for connectivity purposes. If the user writes to an address location not on a 16-bit boundary, the bottom bits are ignored/not used.

8-bit Bus System

For 8-bit bus systems, three scenarios exist as illustrated in Figure 7 on page 46:

1. The register to be written to/read from is less than 8 bits

   In this case, the register can be read or written with one transaction. In the case of a read, registers less than 8 bits wide will return zeros in the un-used bits. Writing to bit locations larger than the register width will cause nothing to happen, i.e. only the pertinent bits are written to the register.

2. The register to be written to/read from is >8 and <=16 bits

   In this case, two transactions are required to read or write the register. The first transaction should read/write the lower byte and the second transaction the upper byte.

3. The register to be written to/read from is >16 and <=32 bits

   In this case, four transactions are required to read or write the register. The first transaction should read/write the lower byte and the second transaction the second byte, and so on.

Because the bus is reading a byte at a time, all lower bits of paddr are decoded in the 8-bit bus case.

Write Timing Operation

A timing diagram for an APB write for a DesignWare APB Interrupt Controller (DW_amba_ictl) register is shown in Figure 8 on page 48. Data, address, and control signals are aligned. The APB frame lasts for two cycles when psel is high.

A write can occur after the first phase with penable low, or after the second phase when penable is high. The second phase is preferred and is used in all DesignWare AMBA Bus IP slave components. The timing diagram is shown with the write occurring after the second phase. Whenever the address on paddr matches a corresponding address from the memory map and provided psel, pwrite, and penable are high, then the corresponding register write enable is generated.

A write from the AHB to the APB does not require the AHB system bus to stall until the transfer on the APB has completed. A write to the APB can be followed by a read from another AHB peripheral (not the DW_apb)

The timing example shown in Figure 8 on page 48 is a 33-bit register and a 32-bit APB data bus. To write this, 5 byte enables would be generated internally. The example shows writing to the first 32 bits with one write transaction.

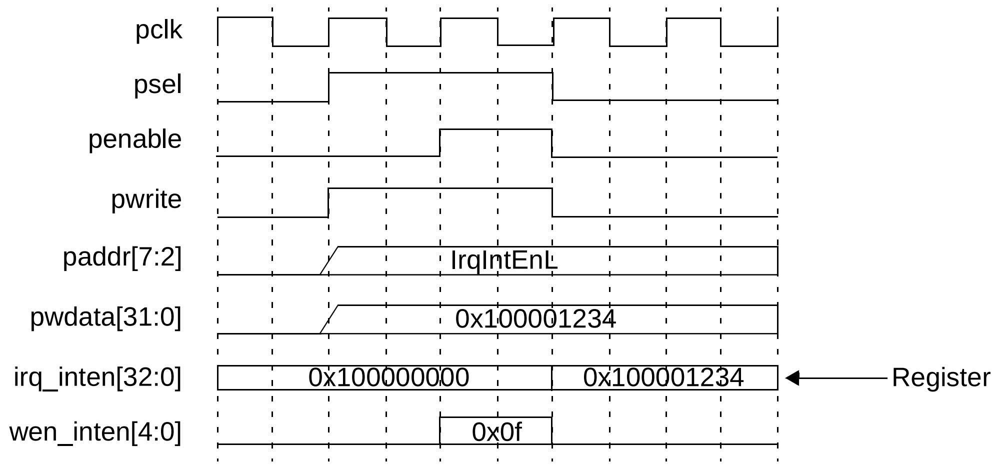


Figure 8: APB Write

Read Timing Operation

A timing diagram for a APB read for a DW_amba_ictl is shown in Figure 9 on page 49. There is no pipelining in the APB interface—data, address and control signals are aligned. The APB frame lasts for two cycles, when psel is high.

Whenever the address on paddr matches the corresponding address from the memory map, psel and pwrite are high, and penable is low, then the corresponding read enable is generated. The read data is registered within the peripheral before passing back to the master via the DW_apb and DW_ahb.

The qualification of the read back data with hready from the bridge is shown in the timing diagram, but this does not form part of the APB interface. The read happens in the first APB cycle and is passed straight back to the AHB master in the same cycles as it passes through the bridge. By returning the data immediately to the AHB bus, the bridge can release control of the AHB data bus faster. This is important for systems where the APB clock is slower than the AHB clock.

Once a read is started, it will be completed and the AHB bus is held until the data is returned from the slave

Note

If a read enable is not active, then the previously read data is maintained on the read back data bus.

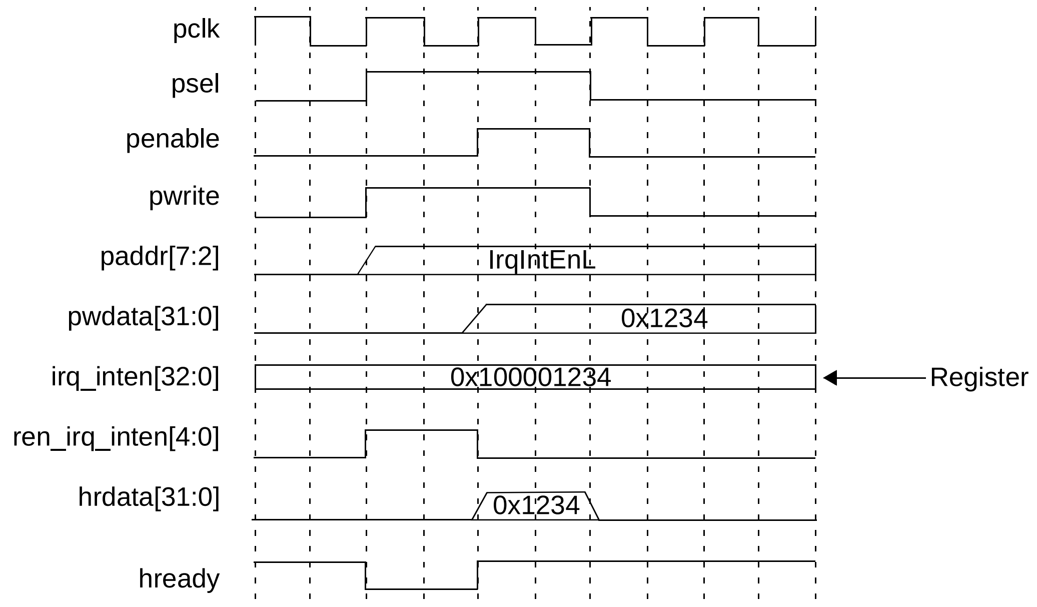


Figure 9: APB Read

Reading from unused locations

Reading from an unused location or unused bits in a particular register always returns zeros. Unlike an AHB slave interface, which would return an error, there is no error mechanism in the DW_apb.