



DesignWare DW_apb_gpio Databook

DW_apb_gpio – Product Code

Copyright Notice and Proprietary Information

© 2018 Synopsys, Inc. All rights reserved. This Synopsys software and all associated documentation are proprietary to Synopsys, Inc. and may only be used pursuant to the terms and conditions of a written license agreement with Synopsys, Inc. All other use, reproduction, modification, or distribution of the Synopsys software or the associated documentation is strictly prohibited.

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 <https://www.synopsys.com/company/legal/trademarks-brands.html>.

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

Third-Party Links

Any links to third-party websites included in this document are for your convenience only. Synopsys does not endorse and is not responsible for such websites and their practices, including privacy practices, availability, and content.

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

www.synopsys.com

Contents

| | |
|--|----|
| Revision History | 7 |
| Preface | 11 |
| Organization | 11 |
| Related Documentation | 11 |
| Web Resources | 12 |
| Customer Support | 12 |
| Product Code | 13 |
| Chapter 1 | |
| Product Overview | 15 |
| 1.1 DesignWare System Overview | 15 |
| 1.2 General Product Description | 17 |
| 1.2.1 DW_apb_gpio Block Diagram | 17 |
| 1.3 Features | 17 |
| 1.4 Standards Compliance | 18 |
| 1.5 Verification Environment Overview | 18 |
| 1.6 Licenses | 18 |
| 1.7 Where To Go From Here | 18 |
| Chapter 2 | |
| Functional Description | 19 |
| 2.1 Data and Control Flow | 19 |
| 2.1.1 Software Control Mode | 20 |
| 2.1.2 Hardware Control Mode | 21 |
| 2.1.3 Reading External Signals | 21 |
| 2.1.4 Synchronization of Data to System Clock | 22 |
| 2.2 Interrupts | 22 |
| 2.2.1 Debounce Operation | 24 |
| 2.2.2 Synchronization of Interrupt Signals to the System Clock | 26 |
| 2.2.3 Interrupt Edge Detection - Single Edge | 27 |
| 2.2.4 Interrupt Edge Detection - Both Edges | 30 |
| 2.2.5 Level-Sensitive Interrupts | 34 |
| Chapter 3 | |
| Parameter Descriptions | 37 |
| 3.1 Basic Config Parameters | 38 |
| 3.2 Port A Parameters | 40 |
| 3.3 Port B Parameters | 44 |
| 3.4 Port C Parameters | 46 |

| | |
|--|-----|
| 3.5 Port D Parameters | 48 |
| Chapter 4 | |
| Signal Descriptions | 51 |
| 4.1 APB Interface Signals | 53 |
| 4.2 Port A Signals | 57 |
| 4.3 Port B Signals | 59 |
| 4.4 Port C Signals | 61 |
| 4.5 Port D Signals | 63 |
| Chapter 5 | |
| Register Descriptions | 65 |
| 5.1 DW_apb_gpio_mem_map/DW_apb_gpio_addr_block Registers | 68 |
| 5.1.1 GPIO_SWPORTA_DR | 70 |
| 5.1.2 GPIO_SWPORTA_DDR | 71 |
| 5.1.3 GPIO_SWPORTA_CTL | 72 |
| 5.1.4 GPIO_SWPORTB_DR | 74 |
| 5.1.5 GPIO_SWPORTB_DDR | 75 |
| 5.1.6 GPIO_SWPORTB_CTL | 76 |
| 5.1.7 GPIO_SWPORTC_DR | 78 |
| 5.1.8 GPIO_SWPORTC_DDR | 79 |
| 5.1.9 GPIO_SWPORTC_CTL | 80 |
| 5.1.10 GPIO_SWPORTD_DR | 82 |
| 5.1.11 GPIO_SWPORTD_DDR | 83 |
| 5.1.12 GPIO_SWPORTD_CTL | 84 |
| 5.1.13 GPIO_INTEN | 86 |
| 5.1.14 GPIO_INTMASK | 88 |
| 5.1.15 GPIO_INTTYPE_LEVEL | 90 |
| 5.1.16 GPIO_INT_POLARITY | 92 |
| 5.1.17 GPIO_INTSTATUS | 94 |
| 5.1.18 GPIO_RAW_INTSTATUS | 95 |
| 5.1.19 GPIO_DEBOUNCE | 96 |
| 5.1.20 GPIO_PORTA_EOI | 98 |
| 5.1.21 GPIO_EXT_PORTA | 100 |
| 5.1.22 GPIO_EXT_PORTB | 101 |
| 5.1.23 GPIO_EXT_PORTC | 102 |
| 5.1.24 GPIO_EXT_PORTD | 103 |
| 5.1.25 GPIO_LS_SYNC | 104 |
| 5.1.26 GPIO_ID_CODE | 105 |
| 5.1.27 GPIO_INT_BOTHEDGE | 106 |
| 5.1.28 GPIO_VER_ID_CODE | 108 |
| 5.1.29 GPIO_CONFIG_REG2 | 109 |
| 5.1.30 GPIO_CONFIG_REG1 | 111 |
| Chapter 6 | |
| Programming the DW_apb_gpio | 117 |
| 6.1 Software Registers | 117 |
| 6.2 Software Drivers | 117 |
| 6.3 Programming Considerations | 117 |

Chapter 7

| | |
|---|-----|
| Verification | 119 |
| 7.1 Overview of DW_apb_gpio Testbench | 120 |
| 7.2 Overview of Vera Tests | 121 |
| 7.2.1 Resets | 121 |
| 7.2.2 APB Slave Interface | 121 |
| 7.2.3 Interrupts | 121 |
| 7.2.4 External Data Interface | 122 |
| 7.2.5 Auxiliary Data Interface | 122 |
| 7.3 Running Simulations from the Command Line | 122 |
| 7.4 Command Line Output Files | 122 |

Chapter 8

| | |
|--|-----|
| Integration Considerations | 123 |
| 8.1 Reading and Writing from an APB Slave | 123 |
| 8.1.1 Reading From Unused Locations | 123 |
| 8.1.2 32-bit Bus System | 124 |
| 8.1.3 16-bit Bus System | 125 |
| 8.1.4 8-bit Bus System | 125 |
| 8.2 Write Timing Operation | 126 |
| 8.3 Read Timing Operation | 127 |
| 8.4 Accessing Top-level Constraints | 127 |
| 8.5 Coherency | 128 |
| 8.5.1 Writing Coherently | 128 |
| 8.5.2 Reading Coherently | 134 |
| 8.6 Performance | 137 |
| 8.6.1 Power Consumption, Frequency, and Area Results | 137 |

Appendix A

| | |
|---|-----|
| Synchronizer Methods | 139 |
| A.1 Synchronizers Used in DW_apb_gpio | 140 |
| A.2 Synchronizer 1: Simple Double Register Synchronizer (DW_apb_gpio) | 141 |

Chapter B

| | |
|---------------------------------------|-----|
| Internal Parameter Descriptions | 143 |
|---------------------------------------|-----|

Appendix C

| | |
|----------------|-----|
| Glossary | 145 |
|----------------|-----|

| | |
|-------------|-----|
| Index | 149 |
|-------------|-----|

Revision History

This section tracks the significant documentation changes that occur from release-to-release and during a release from version 2.06b onward.

| Version | Date | Description |
|---------|--------------|---|
| 2.13a | July 2018 | <p>Added:</p> <ul style="list-style-type: none"> Added support for configurable synchronization depth through the following parameters: GPIO_PA_SYNC_DEPTH, GPIO_PB_SYNC_DEPTH, GPIO_PC_SYNC_DEPTH, and GPIO_PD_SYNC_DEPTH <p>Updated:</p> <ul style="list-style-type: none"> Version changed for 2018.07a release “Performance” on page 137 “Parameter Descriptions” on page 37, “Register Descriptions” on page 65, “Signal Descriptions” on page 51, and “Internal Parameter Descriptions” on page 143 are auto-extracted with change bars from the RTL <p>Removed:</p> <ul style="list-style-type: none"> Chapter 2, “Building and Verifying a Component or Subsystem” and added the contents in the newly created user guide. References to the GPIO Component Type register is removed; no such register exist in the hardware |
| 2.12a | October 2016 | <ul style="list-style-type: none"> Version changed for 2016.10a release “Parameter Descriptions” on page 37 and “Register Descriptions” on page 65 auto-extracted from the RTL Removed the “Running Leda on Generated Code with coreConsultant” section, and reference to Leda directory in Table 2-1 Removed the “Running Leda on Generated Code with coreAssembler” section, and reference to Leda directory in Table 2-4 Replaced Figure 2-2 and Figure 2-3 to remove references to Leda Added “Running VCS XPROP Analyzer” Added an entry for the xprop directory in Table 2-1 and Table 2-4. Moved “Internal Parameter Descriptions” to Appendix |

| Version | Date | Description |
|------------|-----------|--|
| 2.11a | June 2015 | <ul style="list-style-type: none"> ■ Added “Running SpyGlass® Lint and SpyGlass® CDC” ■ Added “Running SpyGlass on Generated Code with coreAssembler” ■ “Signal Descriptions” on page 51 auto-extracted from the RTL ■ Added Appendix A, “Synchronizer Methods” |
| 2.10a | June 2014 | <ul style="list-style-type: none"> ■ Updated version for 2014.06a release ■ Corrected the gpio_ext_portX description because reading back the data register value on gpio_ext_portX register has depreciated after 2.01a ■ Added “Performance” section in the “Integration Considerations” chapter ■ Corrected the External Input/Output Delays in Signals chapter |
| 2.09e-lp00 | Nov 2013 | <p>Added:</p> <ul style="list-style-type: none"> ■ Support for interrupt detection on both the positive and the negative edge of the external input signal ■ Section to explain this feature ■ Configuration parameter GPIO_INT_BOTH_EDGE <p>Updated:</p> <ul style="list-style-type: none"> ■ Register gpio_int_bothedge ■ Modified gpio_config_reg1 register to include the INTERRUPT_BOTH_EDGE_TYPE bit and debounce logic |
| 2.09e | May 2013 | Version change for 2013.05a release. Updated the template. |
| 2.09d | Sep 2012 | Added the part number on the cover and in Table 1-1 |
| 2.09d | Jun 2012 | Version change for 2012.06a release. |
| 2.09c | Mar 2012 | Clarified name of gpio_ext_portxN_rb signal in Control RTL block diagram; corrected read/write information for gpio_ls_sync register. |
| 2.09b | Nov 2011 | Version change for 2011.11a release. |
| 2.09a | Oct 2011 | Version change for 2011.10a release. |
| 2.08b | Jun 2011 | Updated system diagram in Figure 1-1; enhanced “Related Documents” section in Preface. |
| 2.08b | Apr 2011 | Version change for 2011.03a release. |
| 2.08a | Sep 2010 | Corrected names of include files and vcs command used for simulation |
| 2.07a | Dec 2009 | Updated databook to new template for consistency with other IIP/VIP/PHY databooks. |
| 2.07a | May 2009 | Removed references to QuickStarts, as they are no longer supported. |
| 2.07a | Oct 2008 | Version change for 2008.10a release. |
| 2.06c | Jun 2008 | Version change for 2008.06a release. |

| Version | Date | Description |
|---------|----------|--|
| 2.06b | Apr 2008 | Added recommendation that interrupt source be cleared prior to writing to gpio_porta_eoi register. |
| 2.06b | Nov 2007 | Updated to new title for consolidated release notes and installation guide; changed references of "Designware AMBA" to simply "DesignWare." |
| 2.06b | Sep 2007 | Corrected R/W information on some registers; corrected GPIO_DEBOUNCE parameter name from GPIO_DEB_CAP; correct gpio_ext_portx in "Reading External Signals". |
| 2.06b | Jun 2007 | Corrected gpio_config_reg1/2 register descriptions. |

Preface

This databook provides information that you need to interface the DesignWare APB General Purpose Programming I/O peripheral, referred to as DW_apb_gpio throughout the remainder of this databook. It is a component of the DesignWare Advanced Peripheral Bus (DW_apb) and conforms to the [AMBA Specification, Revision 2.0](#) from Arm®.

The information in this databook includes a functional description, signal 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 component.

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, “[Functional Description](#)” describes the functional operation of the DW_apb_gpio.
- Chapter 3, “[Parameter Descriptions](#)” identifies the configurable parameters supported by the DW_apb_gpio.
- Chapter 4, “[Signal Descriptions](#)” provides a list and description of the DW_apb_gpio signals.
- Chapter 5, “[Register Descriptions](#)” describes the programmable registers of the DW_apb_gpio.
- Chapter 6, “[Programming the DW_apb_gpio](#)” provides information needed to program the configured DW_apb_gpio.
- Chapter 7, “[Verification](#)” provides information on verifying the configured DW_apb_gpio.
- Chapter 8, “[Integration Considerations](#)” includes information you need to integrate the configured DW_apb_gpio into your design.
- [Appendix A, “Synchronizer Methods](#)”, documents the synchronizer methods (blocks of synchronizer functionality) used in DW_apb_gpio to cross clock boundaries.
- [Appendix B, “Internal Parameter Descriptions](#)” provides a list of internal parameter descriptions that might be indirectly referenced in expressions in the Signals and Registers chapters.
- [Appendix C, “Glossary](#)”, provides a glossary of general terms.

Related Documentation

- [DW_apb_gpio Driver Kit User Guide](#) – Contains information on the Driver Kit for the DW_apb_gpio; 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, see the *Guide to Documentation for DesignWare Synthesizable Components for AMBA 2 and AMBA 3 AXI*.

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 requested information, using one of the following methods:
 - *For fastest response*, use the SolvNet website. If you fill in your information as explained, 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 **Open A Support Case** 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_gpio
- **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 earlier) 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:
<https://www.synopsys.com/support/global-support-centers.html>

Product Code

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

Table 1-1 DesignWare APB Peripherals – Product Code: 3771-0

| Component Name | Description |
|----------------|---|
| DW_apb_gpio | General Purpose I/O pad control peripheral for the AMBA 2 APB bus |
| DW_apb_rap | Programmable controller for the remap and pause features of the DW_ahb interconnect |
| DW_apb_rtc | A configurable high range counter with an AMBA 2 APB slave interface |
| DW_apb_timers | Configurable system counters, controlled through an AMBA 2 APB interface |
| DW_apb_wdt | A programmable watchdog timer peripheral for the AMBA 2 APB bus |

1

Product Overview

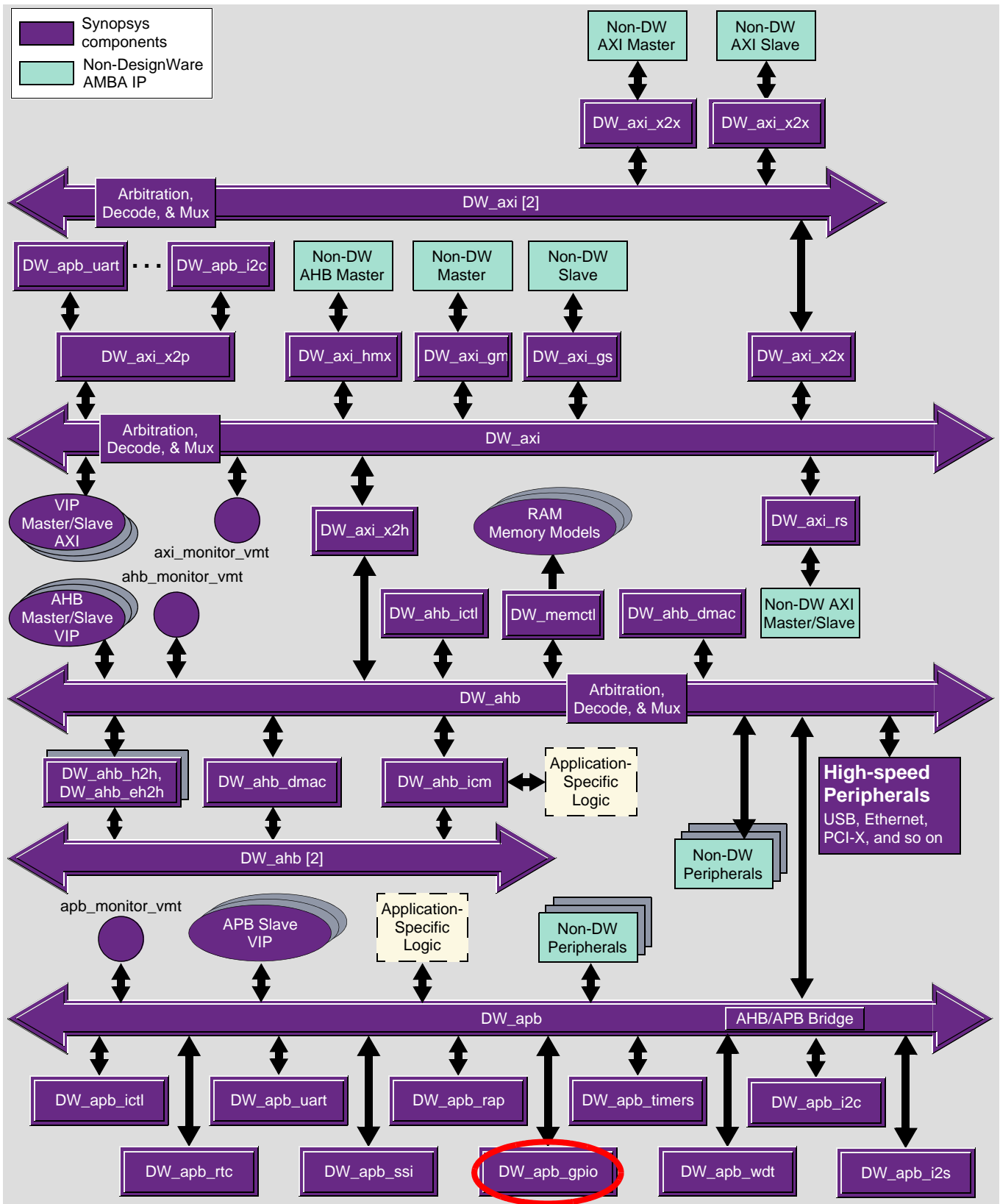
The DW_apb_gpio is a programmable General Purpose Programming I/O (GPIO) peripheral. This component is an AMBA 2.0-compliant Advanced Peripheral Bus (APB) slave device and is part of the family of DesignWare Synthesizable Components.

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.

Figure 1-1 Example of DW_apb_gpio in a Complete System



You can connect, configure, synthesize, and verify the DW_apb_gpio 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_gpio component, you might prefer to use coreConsultant, documentation for which is available in the [coreConsultant User Guide](#).

1.2 General Product Description

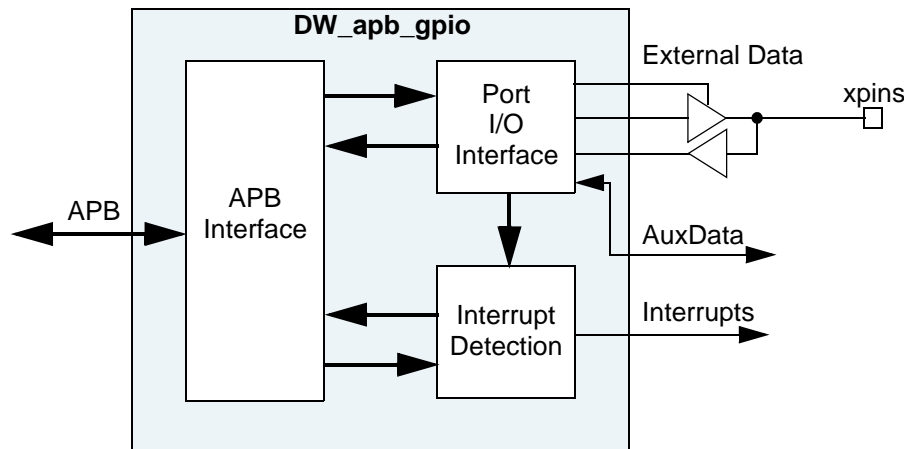
The Synopsys DW_apb_gpio is a component of the DesignWare Advanced Peripheral Bus (DW_apb) and conforms to the [AMBA Specification, Revision 2.0](#) from Arm®.

1.2.1 DW_apb_gpio Block Diagram

Figure 1-2 shows the following functional groupings of the main interfaces to the DW_apb_gpio block:

- APB interface to or from APB bridge
- External data Interface to or from I/O pads
- Auxiliary hardware data interface to or from auxiliary data sink or source
- Interrupt interface to or from interrupt controller

Figure 1-2 DW_apb_gpio Block Diagram



1.3 Features

The DW_apb_gpio is an APB slave and part of the DesignWare Synthesizable Components for AMBA 2; it supports the following features:

- Up to 128 independently configurable signals (if more than 128 signals are required, another DW_apb_gpio should be instantiated)
- Up to four ports, A to D, which are separately configurable
- Separate data registers and data direction registers for each signal
- Configurable hardware and software control for each signal, or for each bit of each signal

- Separate auxiliary data input, data output, and data control for each I/O in Hardware Control mode
- Independently controllable signal bits
- Configurable interrupt mode for Port A
- Configurable debounce logic with an external slow clock to debounce interrupts
- Option to generate single or multiple interrupts
- GPIO Component Version register
- Configurable reset values on output signals
- Configurable synchronization of interrupt signals

Source code for this component is available on a per-project basis as a DesignWare Core. Contact your local sales office for the details.

1.4 Standards Compliance

The DW_apb_gpio component conforms to the [AMBA Specification, Revision 2.0](#) from Arm®. Readers are assumed to be familiar with this specification.

1.5 Verification Environment Overview

The DW_apb_gpio includes an extensive verification environment, which sets up and invokes your selected simulation tool to execute tests that verify the functionality of the configured component. You can then analyze the results of the simulation.

The “[Verification](#)” on page [119](#) section discusses the specific procedures for verifying the DW_apb_gpio.

1.6 Licenses

Before you begin using the DW_apb_gpio, you must have a valid license. For more information, see “Licenses” in the [DesignWare Synthesizable Components for AMBA 2/AMBA 3 AXI Installation Guide](#).

1.7 Where To Go From Here

At this point, you may want to get started working with the DW_apb_gpio 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, see [Guide to coreTools Documentation](#).

For more information about configuring, synthesizing, and verifying just your DW_apb_gpio component, see “Overview of the coreConsultant Configuration and Integration Process” in [DesignWare Synthesizable Components for AMBA 2 User Guide](#).

For more information about implementing your DW_apb_gpio component within a DesignWare subsystem using coreAssembler, see “Overview of the coreAssembler Configuration and Integration Process” in [DesignWare Synthesizable Components for AMBA 2 User Guide](#).

2

Functional Description

This chapter describes the functional operation of the DW_apb_gpio.

2.1 Data and Control Flow

The DW_apb_gpio controls the output data and direction of external I/O pads. It also can read back the data on external pads using memory-mapped registers.

There are two methods for generating the default source of the input data, the output data, and the control of each signal—through software or hardware control. Software control occurs over the APB bus interface; hardware control is through the auxiliary hardware control interface.

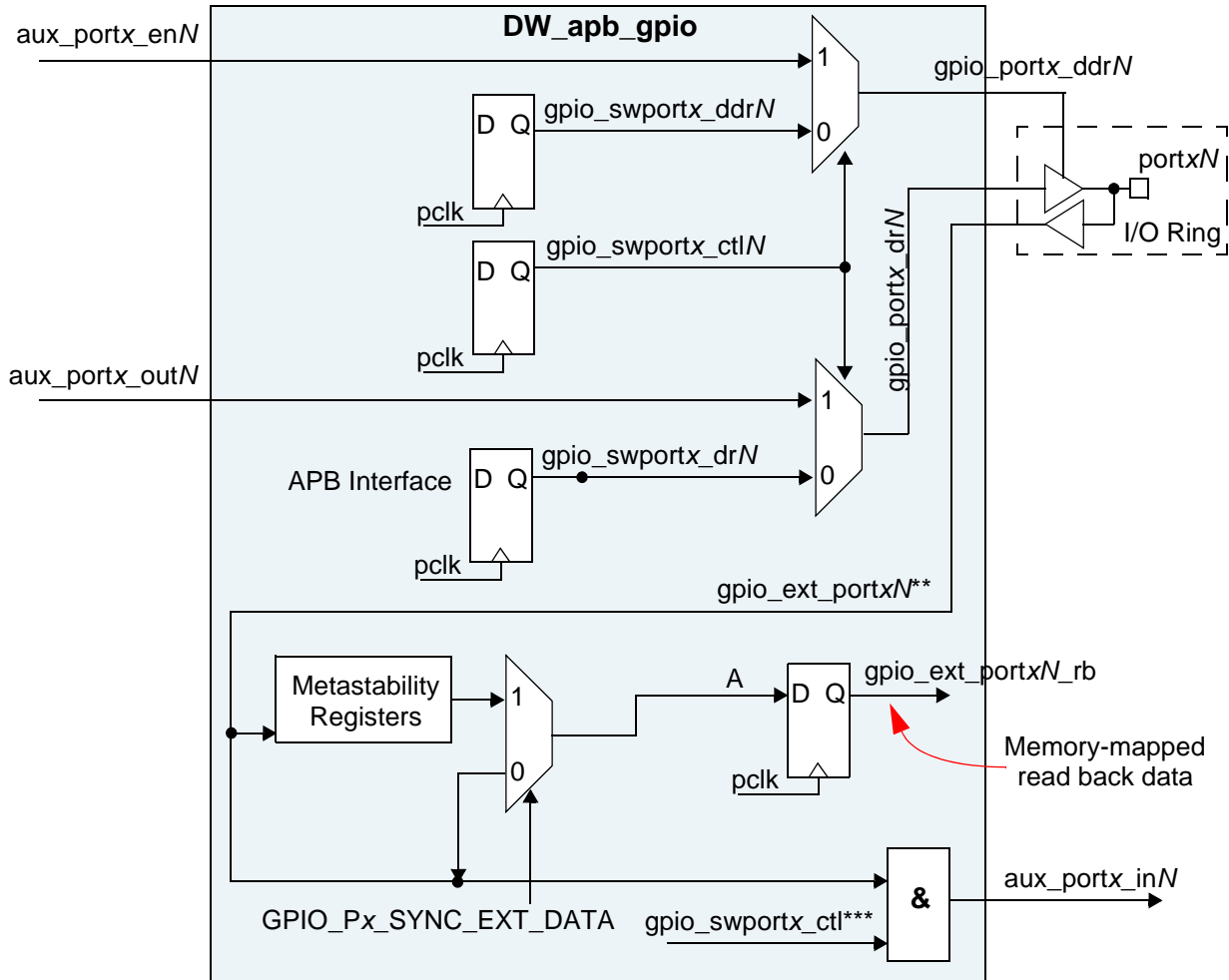
The software option always exists and is described in more detail in “[Software Control Mode](#)” on page 20. The hardware option for each signal exists only if you choose it during configuration time. Provided the hardware option is built, you can switch between software and hardware modes by writing to a control register for the corresponding signal. Also, the device can be configured so that you can individually switch between hardware and software modes for each bit of each signal, provided that hardware mode exists; for more detail, see “[Hardware Control Mode](#)” on page 21. The data and control flow for a signal are shown in [Figure 2-1](#).

**Note**

Each bit in each signal is individually controllable. Therefore, each register can be thought of as N individual registers, where N is the signal width.

Synchronization of the data to the bus clock is described in “[Synchronization of Data to System Clock](#)” on page 22. The default direction of any signal is set at configuration time. Later sections describe the I/O signals and memory-mapped registers in more detail.

Figure 2-1 Control RTL Block Diagram



N = 0 through GPIO_PWIDTH_X where "X" is A, B, C or D.

* These data are multiplexed onto `prdata` when a read from the `gpio_ext_portx` register occurs.

** This is a port signal. See "Signal Descriptions" on page 51 for more information.

*** If configuration parameter `GPIO_PORTN_SINGLE_CTL = 0`, this remains a single bit as shown. If `GPIO_PORTN_SINGLE_CTL = 1`, this becomes a bus, `gpio_swportx_ctlN`.

2.1.1 Software Control Mode

When a signal is configured for software control, the data and direction control for the signal are sourced from the data register (`gpio_swportx_dr`) and direction control register (`gpio_swportx_ddd`), where *x* is either a, b, c, or d.

Under software control, the direction of the external I/O pad is controlled by a write to the Port*x* data direction register (`gpio_swportx_ddd`). The data written to this memory-mapped register gets mapped onto an output signal, `gpio_portx_ddd`, of the DW_apb_gpio peripheral. This output signal controls the direction of an external I/O pad.

The data written to the Port*x* data register (`gpio_swportx_dr`) drives the output buffer of the I/O pad.

External data are input on the external data signal, `gpio_ext_portx`. Depending on whether `gpio_ext_portx` is configured as an input or output, the following occurs:

- Input – Reads the values on the signal
- Output – Reads the data register for Portx

The `gpio_ext_portx` register is read-only, meaning that it cannot be written from the APB software interface.

2.1.2 Hardware Control Mode

If a signal is configured for hardware control, it has external, auxiliary hardware signals controlling the data and the direction of Ports A through D. In this mode, the auxiliary data input signal (`aux_portx_out`) and direction control signal (`aux_portx_en`) are selected, where *x* is either a, b, c, or d.

The data direction of the external I/O pad, `gpio_portx_ddr`, is controlled through the auxiliary signal direction control signal, `aux_portx_en`.

For lines that are set to Output, the `gpio_portx_dr` and `gpio_portx_ddr` output signals drive the data and direction control onto the bidirectional pad that exists within the I/O ring of the SoC device. [Figure 2-1](#) on page 19 shows how the DW_apb_gpio peripheral controls the data and direction signals of an I/O PAD and data generation for the auxiliary source.

The `gpio_swportx_ctl` signal masks the value on the `gpio_ext_portx` external signal in order to generate `aux_portx_in`. The net result is that when hardware mode is selected, the value on `aux_port_in` output signal is equal to the value on the `gpio_ext_portx` input signal. When software mode is selected, the `aux_portx_in` output signal is always driven low. Setting bit 0 of `gpio_swportx_ctl` to 1 selects hardware mode for the entire signal if the parameter `GPIO_PORTX_SINGLE_CTL` is 1. If `GPIO_PORTX_SINGLE_CTL` is 0, setting bit *n* of `gpio_swportx_ctl` to 1 selects hardware mode for bit *n* of Portx. Setting bit 0 of `gpio_swportx_ctl` to 0 selects software mode for the entire signal if `GPIO_PORTX_SINGLE_CTL` is 1, while setting bit *n* of `gpio_swportx_ctl` to 0 selects software mode for bit *n* of Portx if `GPIO_PORTX_SINGLE_CTL` is 0.

2.1.3 Reading External Signals

The data on the external gpio signal is read by an APB read of the memory-mapped register, `gpio_ext_portx`. An APB read to the `gpio_ext_portx` register provides either the data on the `gpio_ext_portx` control lines or the contents of the `gpio_swportx_dr`, depending on the value of `gpio_swportx_ddr`.

[Figure 2-1](#) on page 19 shows how the hardware/software option is multiplexed, where the control lines for the multiplexing come from a memory-mapped register. It also shows the synchronization registers and the individual bit control of each data and data direction bit.

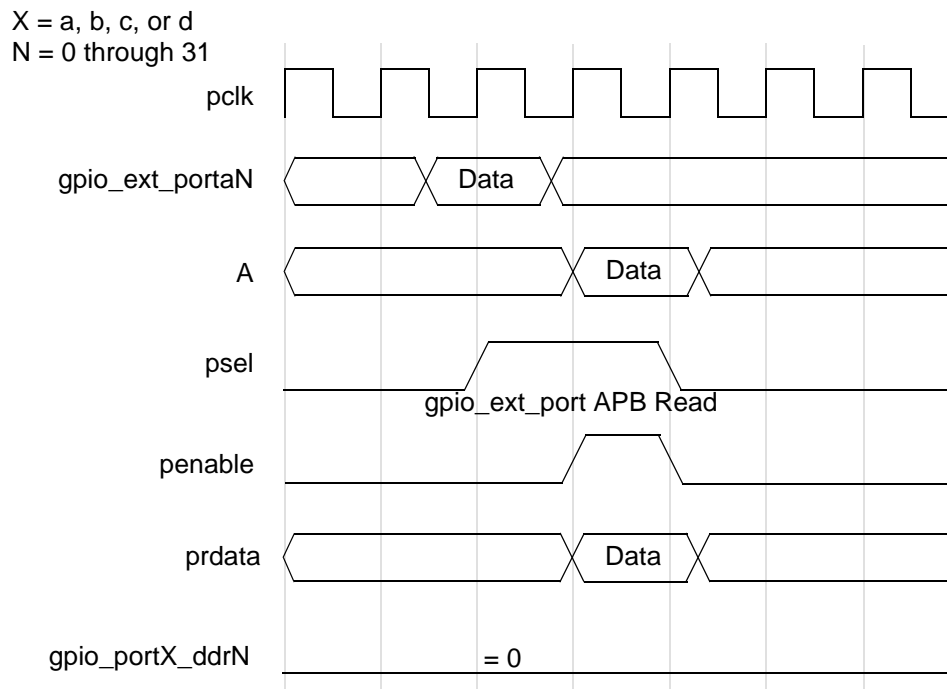


Note

The synchronization registers are optimized at the synthesis stage if the `GPIO_PA_SYNC_EXT_DATA` configuration parameter is equal to 0.

Figure 2-2 shows a timing diagram of a read to the `gpio_ext_portX` memory map registers when the direction is set to *Input* and the metastability registers are included.

Figure 2-2 Read Back of External `gpio_ext_portX` Data Timing



Note

The maximum data rate that can be read back on consecutive reads from `gpio_ext_portX` must be less than $pclk/2$. Two `pclk` cycles per read access is required due to the non-pipelined nature of the APB. The assumption is that the APB bridge does not lose ownership of the AHB during consecutive accesses when $PCLK=HCLK$.

2.1.4 Synchronization of Data to System Clock

Synchronization of `gpio_ext_portX` to `pclk` prior to an APB read is enabled if the corresponding signal configuration parameter `GPIO_PX_SYNC_EXT_DATA` is set ($x = A, B, C, \text{ or } D$); this is shown in Figure 2-1 on page 19.

2.2 Interrupts

Port A can be programmed to accept external signals as interrupt sources on any of the bits of the signal. The type of interrupt is programmable with one of the following settings:

- Active-high and level
- Active-low and level
- Rising edge
- Falling edge

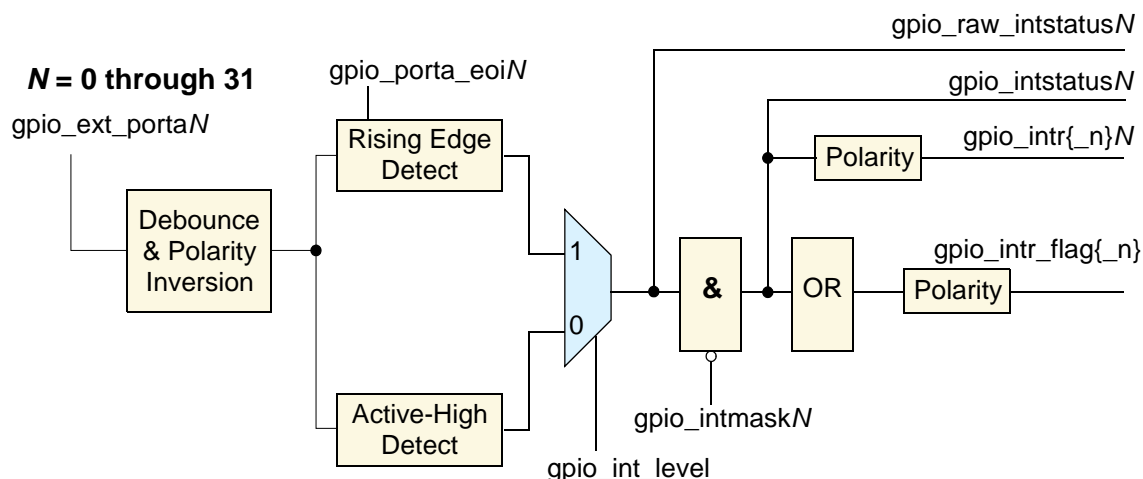
The interrupts can be masked by programming the `gpio_intmask` register. The interrupt status can be read before masking (called raw status) and after masking.

The interrupts are also combined into a single interrupt output signal, which has the same polarity as the individual interrupts. Either individual interrupts (`gpio_intr` or `gpio_intr_n`) or a single combined interrupt (`gpio_intr_flag` or `gpio_intr_flag_n`) can be generated. In order to mask the combined interrupt, all individual interrupts have to be masked. The single combined interrupt does not have its own mask bit.

Whenever Port A is configured for interrupts, the data direction must be set to Input and the mode must be set to Software for interrupts to be latched. If the data direction register is reprogrammed to Output or the mode register is programmed to enable Hardware mode, then any pending interrupts are not lost. However, no new interrupts are generated.

Figure 2-3 illustrates how the interrupts are generated and how the data flows. The signal names in the diagram correspond to either I/O signals or memory-mapped registers.

Figure 2-3 Interrupt RTL Block Diagram



Two interrupt request connection schemes are supported, and one scheme is chosen during configuration. The simplest connection scheme is where the combined interrupt `gpio_intr_flag` is generated by ORing together the bits of the `gpio_intr` bus. When only the combined interrupt request is used, then the `gpio_status` register must be read in the interrupt service routine (ISR) to find the source of the interrupt. When the individual interrupts lines are connected directly to the interrupt controller, then the `gpio_status` register does not have to be read by the ISR.

For edge-detected interrupts, the ISR can clear the interrupt by writing a 1 to the `gpio_porta_eoi` register for the corresponding bit to disable the interrupt. This write also clears the interrupt status and raw status registers. It is recommended that the interrupt source be cleared prior to writing to the `gpio_porta_eoi` register. Writing to the `gpio_porta_eoi` register has no effect on level-sensitive interrupts.

If level-sensitive interrupts cause the processor to interrupt, then the ISR can poll the `gpio_rawint` status register until the interrupt source disappears, or it can write to the `gpio_intmask` register to mask the interrupt before exiting the ISR. If the ISR exits without masking or disabling the interrupt prior to exiting, then the level-sensitive interrupt repeatedly requests an interrupt until the interrupt is cleared at the source.

If the `gpio_intr_flag` connection scheme is used and the interrupt service routine reads the `gpio_intr_status` register to find multiple pending interrupt requests, then it is up to the processor to prioritize these pending

interrupt requests. There are no restrictions on the number of edge-detected interrupts that can be cleared simultaneously by writing multiple 1's to the `gpio_porta_eoi` register.

2.2.1 Debounce Operation

If you have configured Port A to include the interrupt feature, the `DW_apb_gpio` can be configured to either include or exclude a debounce capability using the `GPIO_DEBOUNCE` parameter.

The external signal can be debounced to remove any spurious glitches that are less than one period of the external debouncing clock.

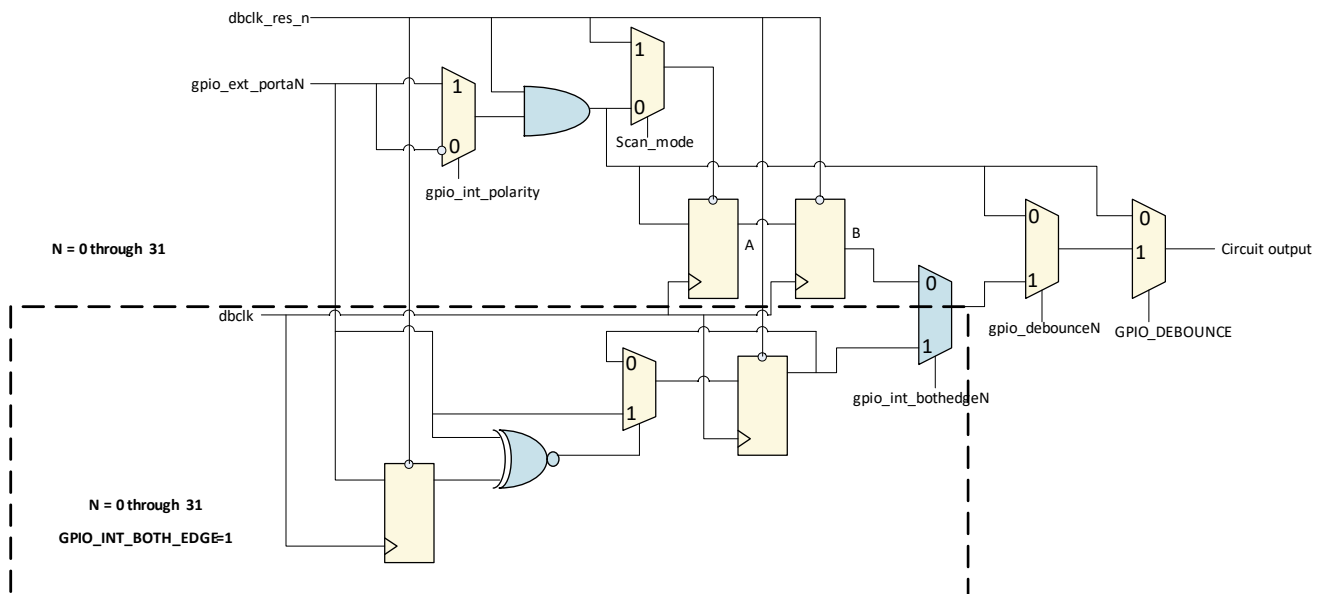
Figure 2-4 shows an RTL diagram of the debounce circuitry. The timing diagrams show an active-high input signal on `gpio_ext_portaN`. The polarity of the input signal detection is controlled by the memory-mapped signal, `gpio_int_polarity`. For a falling-edge- or active-low-sensitive input, the input is then inverted and the same debounce logic is used as for rising-edge or active-high level-sensitive interrupts.



Note

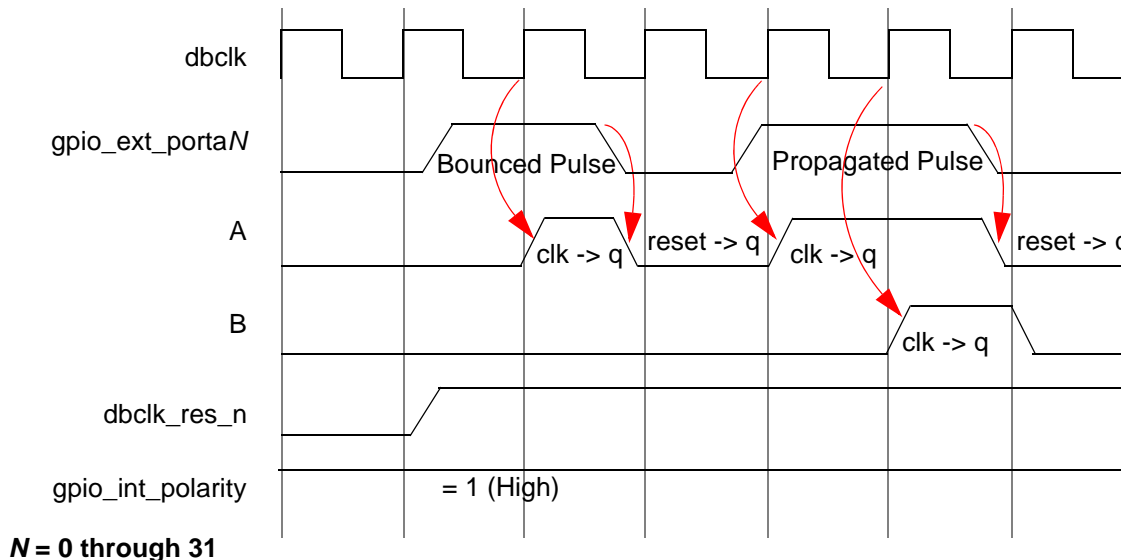
The logic to debounce both rising edge and falling edge is present when `GPIO_INT_BOTH_EDGE = 1`. The signal is debounced on either a single edge or on both edges, depending on the value of the `gpio_int_bothedgeN` register.

Figure 2-4 Debounce RTL Diagram



A timing diagram of the single-edge debounce circuitry is shown in [Figure 2-5](#).

Figure 2-5 Debounce Timing With Asynchronous Reset Flip-Flops and GPIO_INT_BOTH_EDGE = 0



When input interrupt signals are debounced using a debounce clock, the signals must be active for a minimum of two cycles of the debounce clock to guarantee that they are registered. Any input pulse widths less than a debounce clock period are bounced. A pulse width between one and two debounce clock widths may or may not propagate, depending on its phase relationship to the debounce clock. If the input pulse spans two rising edges of the debounce clock, it is registered. If it spans only one rising edge, it is not registered.

The timing diagram in [Figure 2-5](#) shows both cases: the input signal being bounced, and later, a propagated input signal. If the DW_apb_gpio supports debounce, then debouncing input signals on Port A can be enabled or disabled under software control.

The dbclk_res_n signal is asynchronously asserted and synchronously de-asserted to the debounce clock, dbclk. The system reset signal, presetn, is asynchronously asserted and synchronously de-asserted to pclk; synchronization must be external to the component. The pclk and dbclk signals are assumed to be asynchronous to each other.

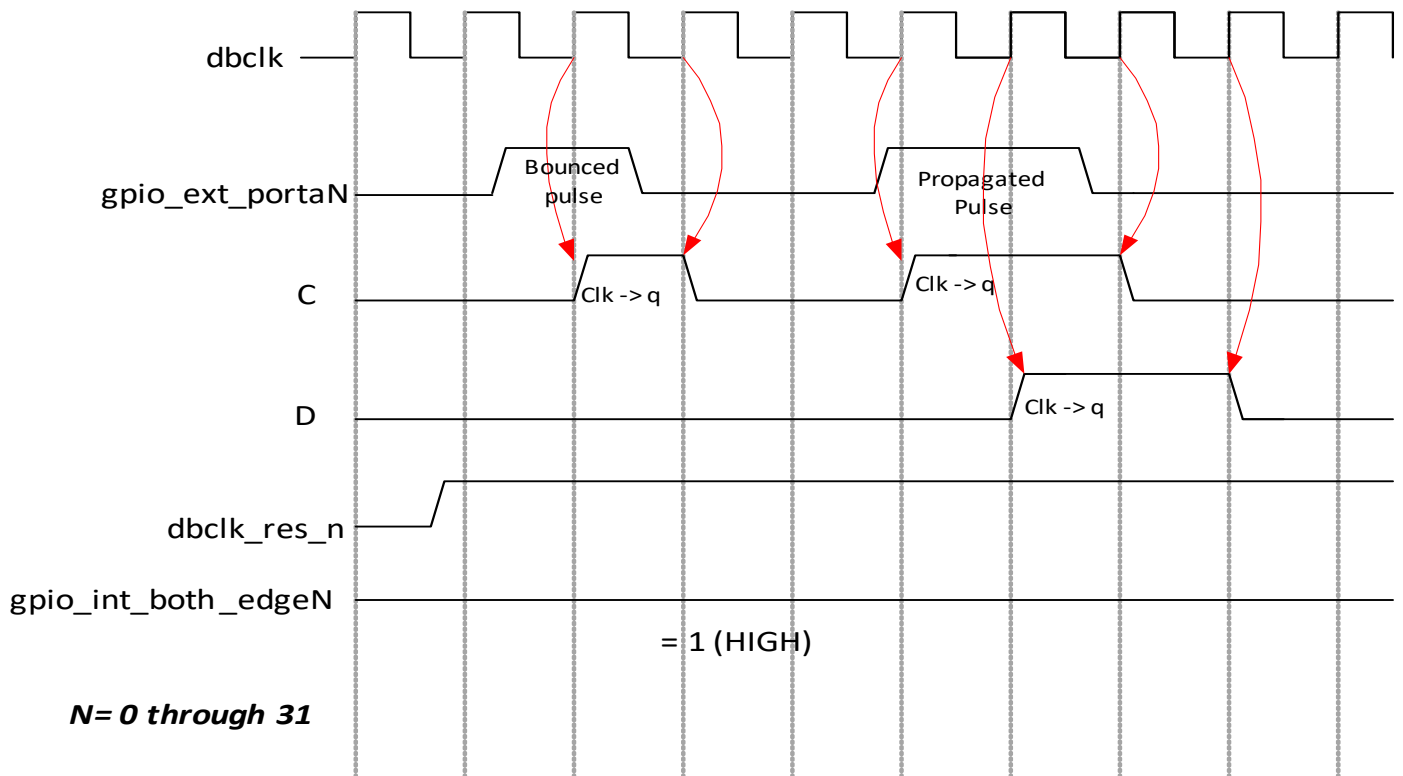


Note

The use of the debounce circuitry increases interrupt latency by two clock cycles of the debounce clock.

The debounce circuitry works with only asynchronous reset flip-flops.

[Figure 2-6](#) shows the debounce timing with parameter GPIO_INT_BOTH_EDGE = 1 and the gpio_int_bothedge register programmed to detect both edges.

Figure 2-6 Debounce Timing With GPIO_INT_BOTH_EDGE = 1

2.2.2 Synchronization of Interrupt Signals to the System Clock

Interrupt signals can be internally synchronized to a system clock, `pclk_intr`. Synchronization to `pclk_intr` must occur for edge-detect signals. Edge-detected interrupts to the processor are always synchronous to the system bus clock. With level-sensitive interrupts, synchronization is optional and under software control.

The `pclk_intr` signal is needed for systems that may have the `DW_apb_gpio` `pclk` bus clock gated off, but the system still wants to detect interrupts. It is assumed that this clock is synchronous to `pclk`. If interrupt detection is required only when `pclk` is running, then `pclk_intr` and `pclk` can be connected to the same clock source. If the system employs a gated `pclk` to the `DW_apb_gpio`, `pclk_intr` needs to be running for interrupt detection to occur.

The `gpio_intrclk_en` output signal is asserted when either edge-sensitive interrupts or level-sensitive interrupts requiring synchronization are enabled in the `DW_apb_gpio` block. Both cases require a clock for detection. Therefore, this signal can cause the external clock generator block to generate `pclk_intr`.



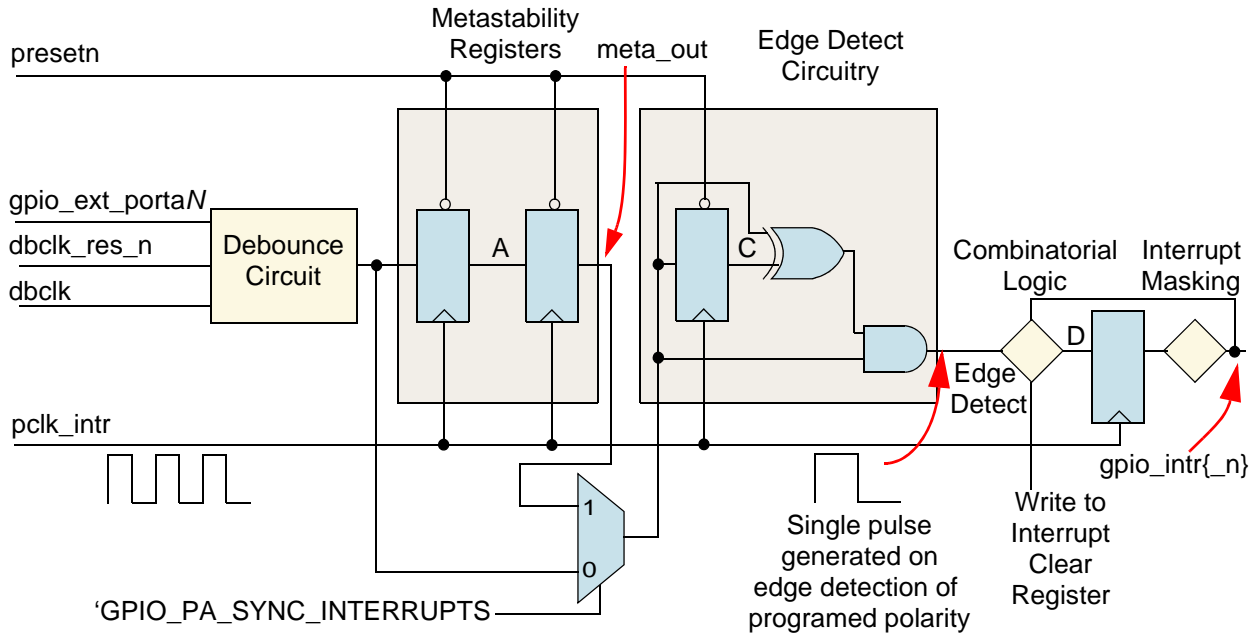
Note

`pclk` can be gated but `pclk_intr` should be running to detect the interrupts. Moreover, the internal interrupt detection/clear logic depends on `pclk_intr`, which requires `pclk_intr` to be running when APB reads/writes are performed with respect to `pclk`.

2.2.3 Interrupt Edge Detection - Single Edge

Figure 2-7 shows an RTL diagram of the synchronization and edge detection of interrupt sources on `gpio_ext_portaN` signals, when `GPIO_INT_BOTH_EDGE = 0`.

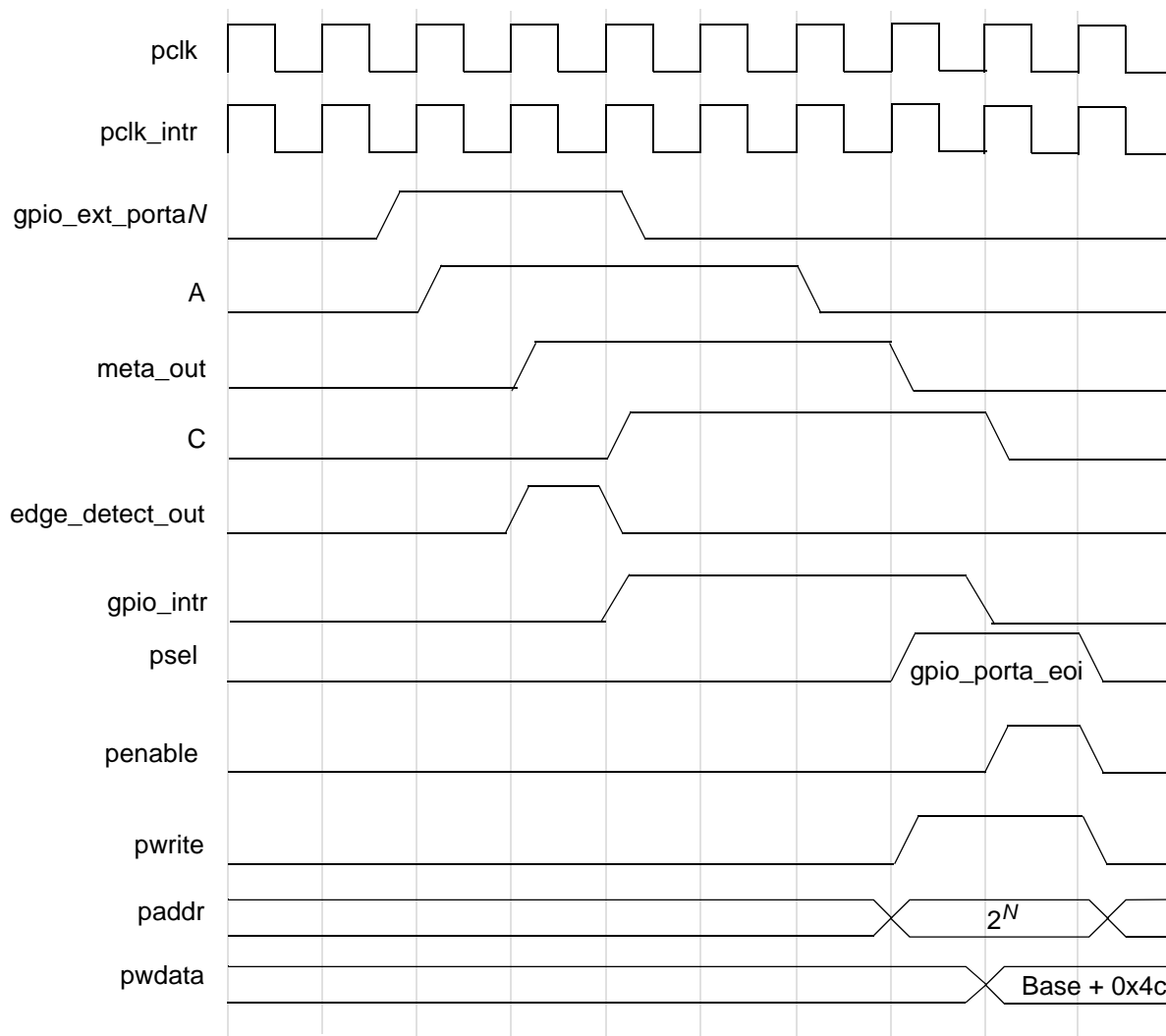
Figure 2-7 Synchronization and Edge Detect Interrupt Generation When `GPIO_INT_BOTH_EDGE = 0`



The MUX allows inclusion or exclusion of the metastability registers at configuration depending on the value of `GPIO_PA_SYNC_INTERRUPTS`. If this parameter is a 1, the registers are included.

Figure 2-8 shows a timing diagram in which an interrupt is generated on the rising edge of an input on Port A; this is where the debounce logic is disabled and Metastability registers are included. It also shows how an interrupt is cleared by a write to the interrupt clear register.

Figure 2-8 Interrupt Edge Detection and Interrupt Clear Timing when GPIO_SYNC_PA_INTERRUPTS = 1 (Metastability Included)

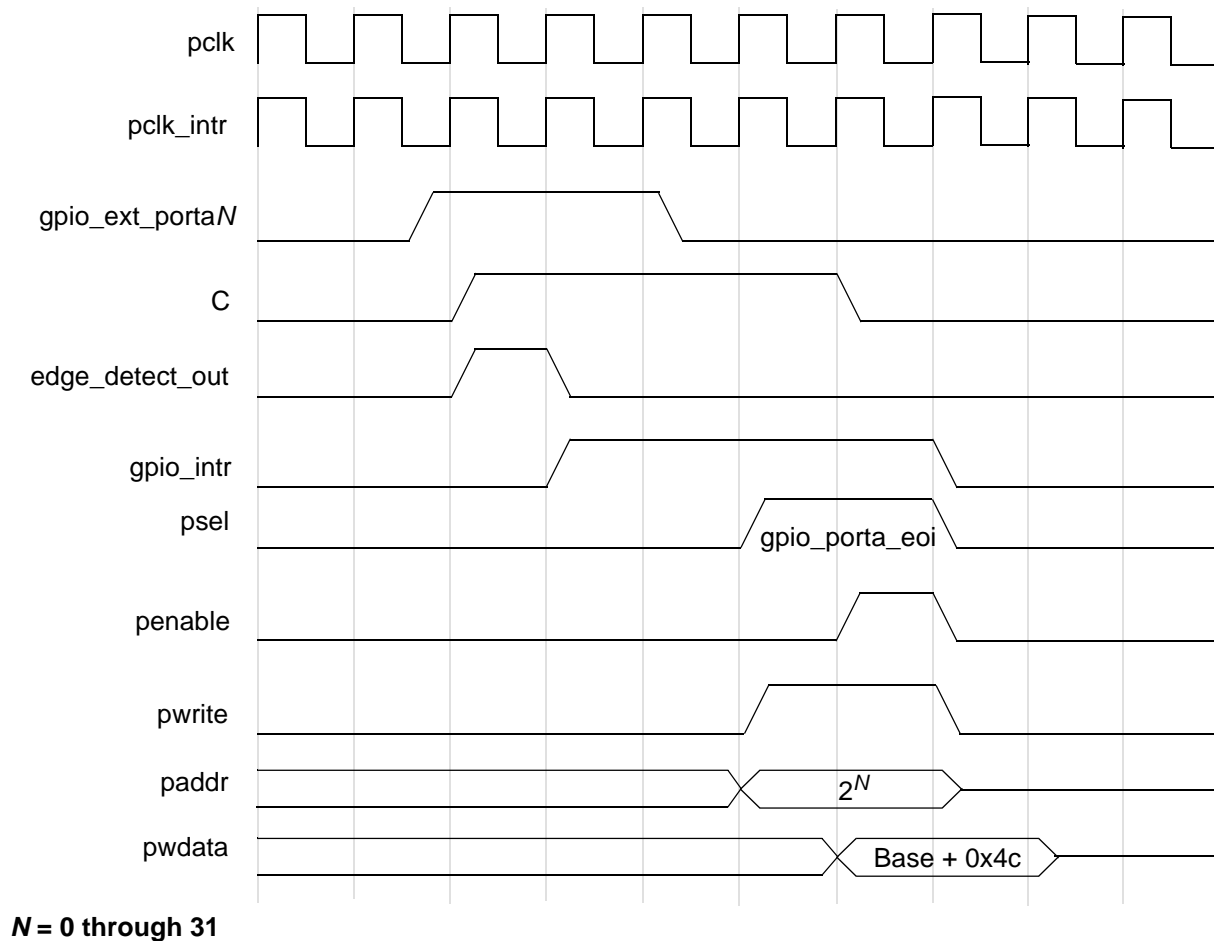


$N = 0$ through 31

A case may arise where the Interrupt Service Routine (ISR) writes to the interrupt clear register in order to clear an existing interrupt during the same clock cycle in which a new interrupt is detected. In such a case, writing to the interrupt clear register clears only the first interrupt. The second interrupt is not lost, since setting an interrupt has a higher priority than clearing it.

Figure 2-9 shows a timing diagram similar to Figure 2-8 on page 28, except that Metastability registers are removed. It also shows how an interrupt is cleared by a write to the interrupt clear register.

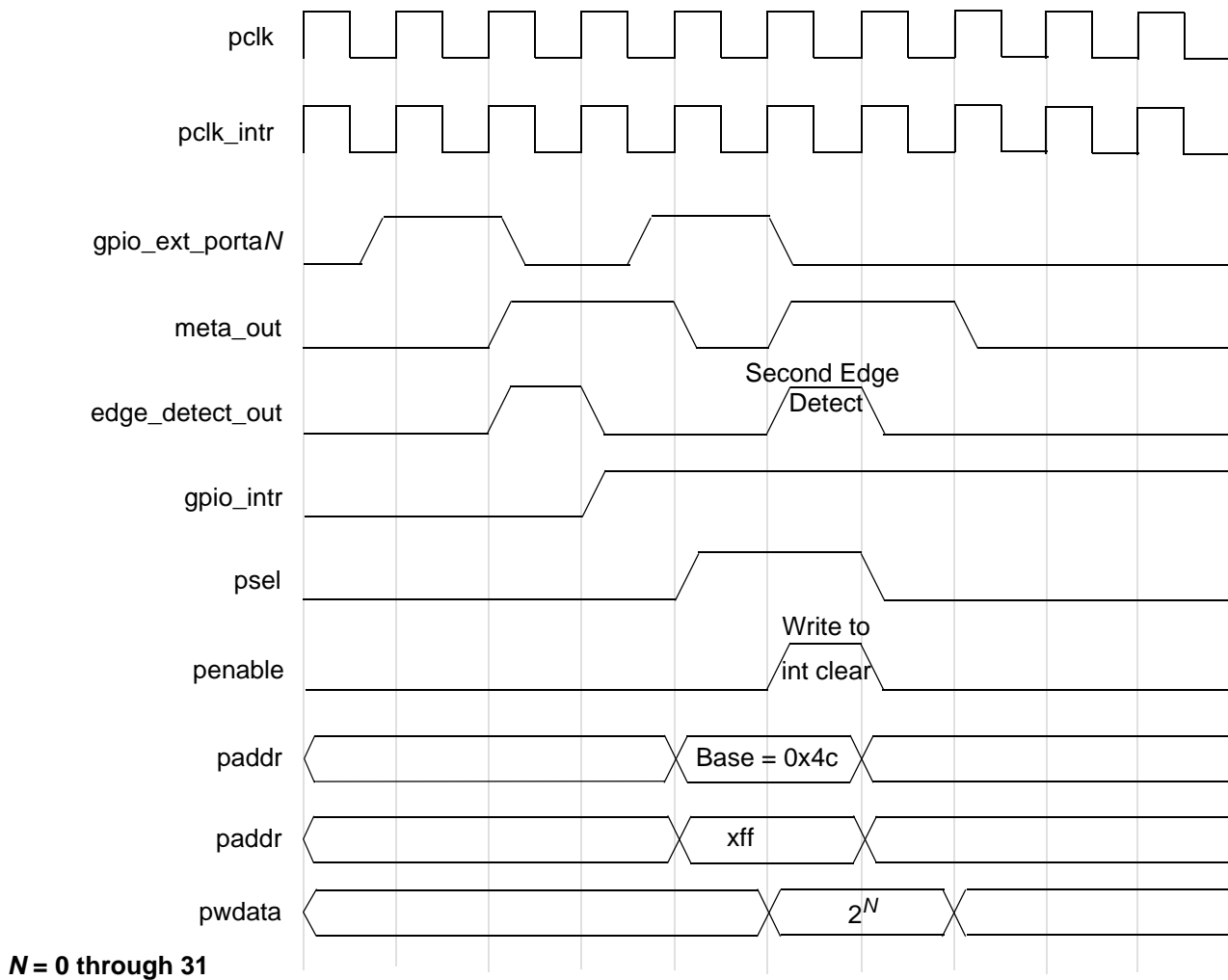
Figure 2-9 Interrupt Edge Detection and Interrupt Clear Timing when GPIO_SYNC_PA_INTERRUPTS = 0 (Metastability Removed)



Note

Since the Metastability registers are removed from the path, A and B waveforms shown in [Figure 2-8](#) no longer exist, and all subsequent signal delays are reduced by two clock cycles.

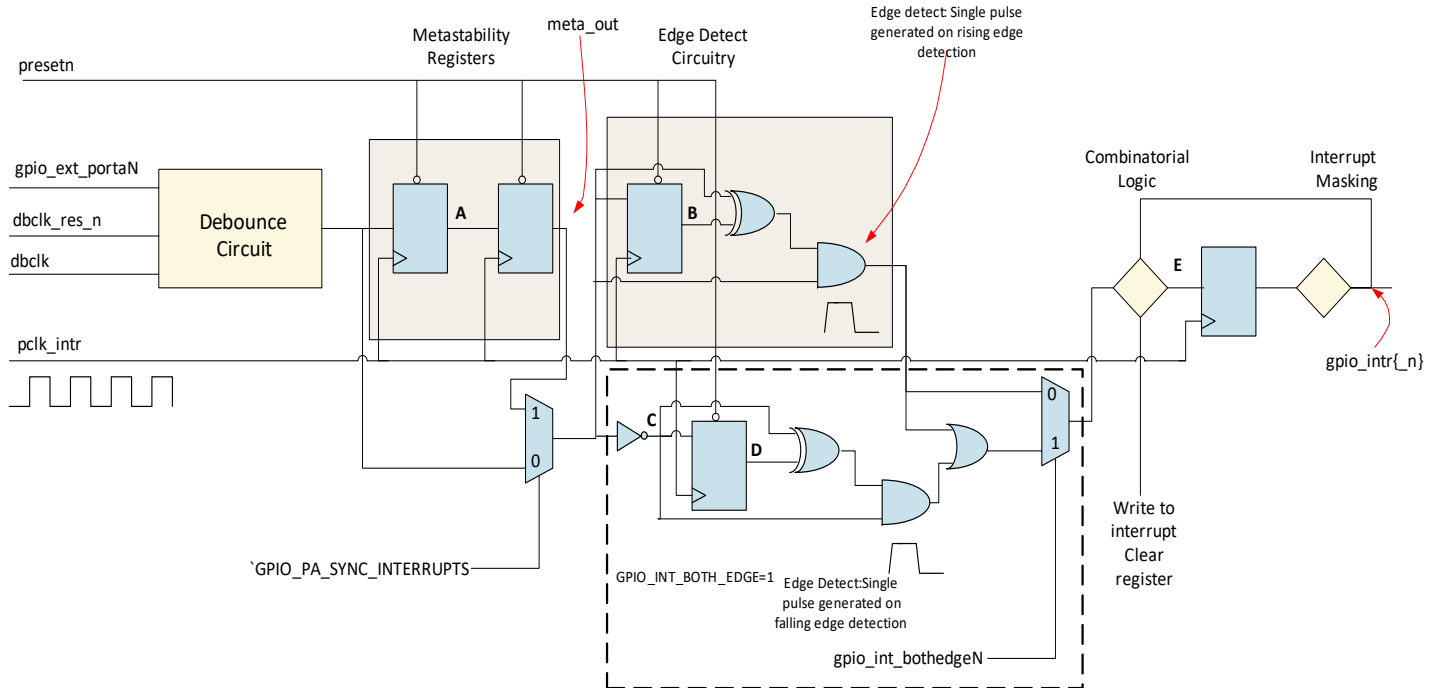
[Figure 2-10](#) shows such a case where the debounce logic is unused. In this timing diagram, `meta_out` and `edge_detect_out` are the outputs of the second metastability register and the edge detect logic, respectively. The second edge detection occurs on the same cycle as the write to the interrupt clear register. In this example, the write to the interrupt clear register does not clear the second interrupt, and the `gpio_intr{n}` signal is not de-asserted.

Figure 2-10 Write to Interrupt Clear Register, Coincident with Detection of New Interrupt

2.2.4 Interrupt Edge Detection - Both Edges

Interrupt detection logic for both the rising edge and the falling edge is available only when `GPIO_INT_BOTH_EDGE = 1` and this logic detects the interrupt on both the rising edge and the falling edge when the `gpio_int_bothedge` register is programmed to 1.

Figure 2-11 shows the synchronization and edge detect interrupt generation of the interrupt sources on `gpio_ext_portaN` signals when `GPIO_INT_BOTH_EDGE = 1`.

Figure 2-11 Synchronization and Edge Detect Interrupt Generation When GPIO_INT_BOTH_EDGE = 1

shows a timing diagram where an interrupt is generated on both the rising edge and the falling edge of an input on Port A, that is, with `GPIO_INT_BOTH_EDGE = 1` and `gpio_int_bothedge` programmed to detect both edges. In this scenario, debounce logic is disabled and metastability registers are included. This figure also shows how an interrupt is cleared by a write to the interrupt clear register.

Figure 2-12 Interrupt Edge Detection and Interrupt Clear Timing When GPIO_SYNC_PA_INTERRUPTS = 1 and GPIO_INT_BOTH_EDGE = 1 (Metastability Included)

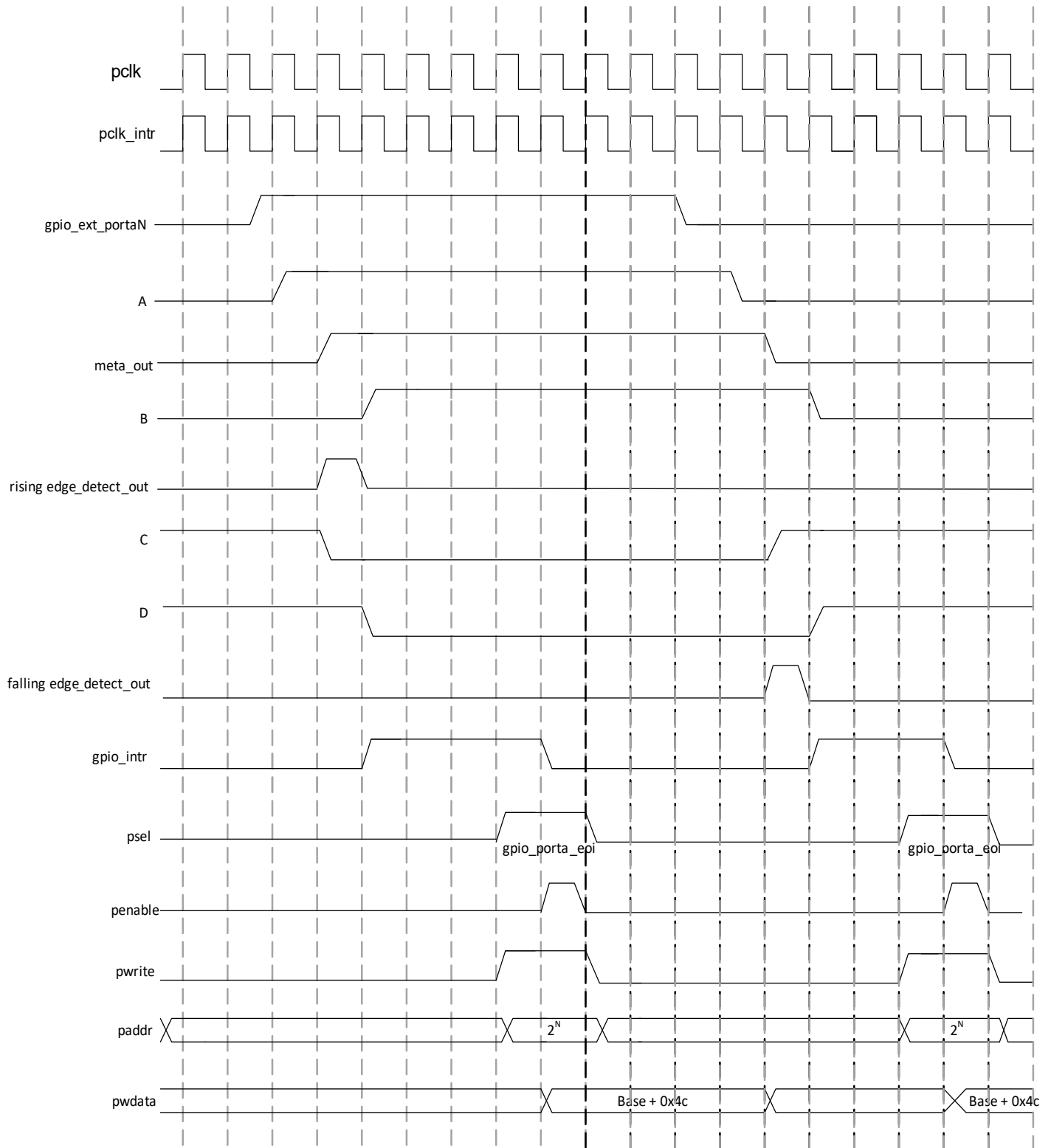
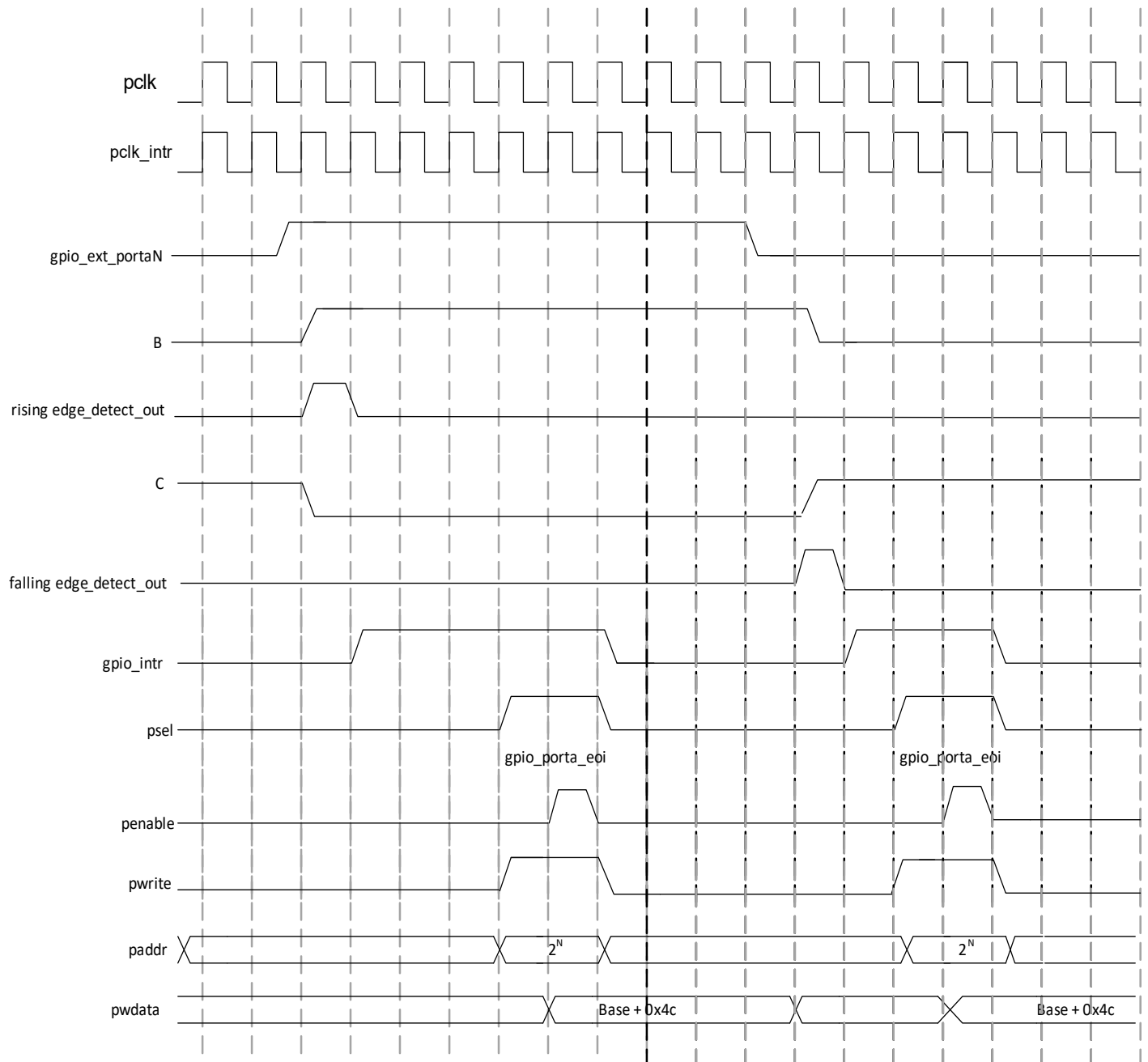


Figure 2-13 shows a timing diagram similar to , except that in this scenario, metastability registers are removed.

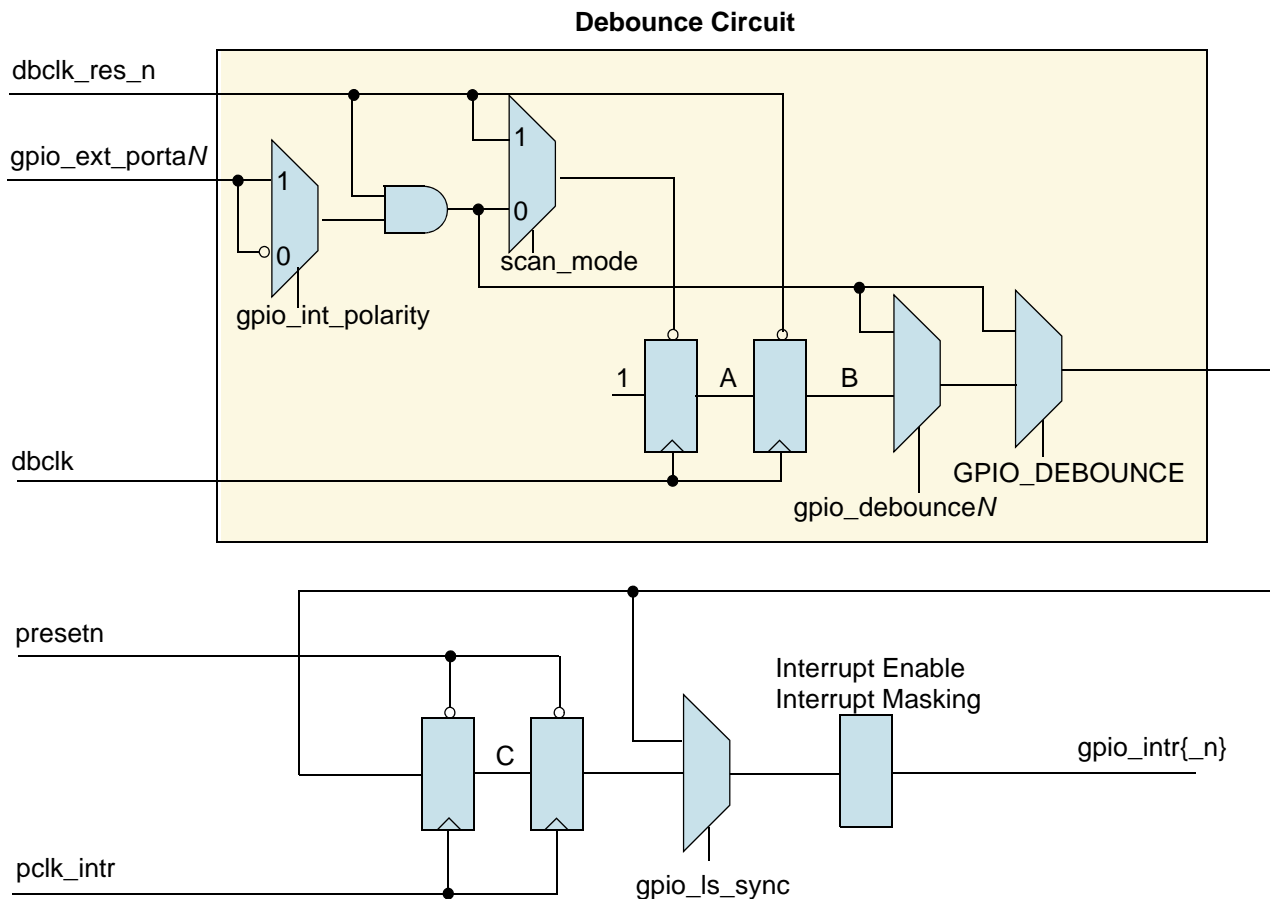
Figure 2-13 Interrupt Edge Detection and Interrupt Clear Timing When GPIO_SYNC_PA_INTERRUPTS = 0 and GPIO_INT_BOTH_EDGE = 1 (Metastability Removed)



2.2.5 Level-Sensitive Interrupts

Figure 2-14 shows the generation of level-sensitive interrupts. As for edge-detect interrupts, you can configure DW_apb_gpio with or without debounce logic.

Figure 2-14 Level-Sensitive Interrupt RTL Diagram



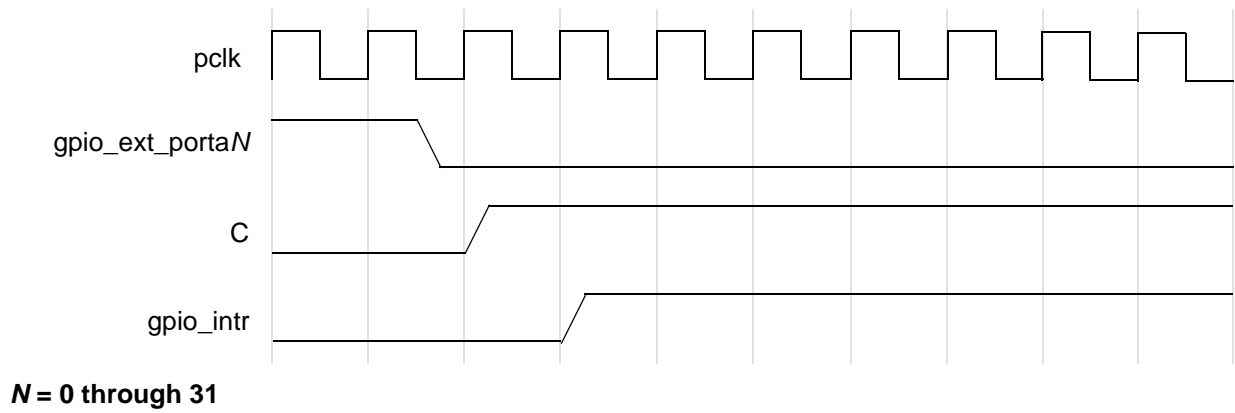
With level-sensitive interrupts, there is a choice of whether they are synchronized to the interrupt clock `pclk_intr` or are entirely combinational (aside from the debounce circuitry). The selection is done by programming the `gpio_ls_sync` (GPIO Level Sensitive Synchronous) register.

This is a memory-mapped bit that inserts two metastability registers clocked off of `pclk_intr` to synchronize the level-sensitive interrupts to `pclk_intr`. When `gpio_ls_sync` is not asserted, then there is no guarantee that the interrupt lines are synchronous to `pclk_intr`. A processor status register may need to be set to indicate asynchronous interrupts. When `gpio_ls_sync` is asserted, then the `pclk_intr` clock must be present to pass the interrupt to the interrupt controller block. The `gpio_intrclk_en` output signal is asserted when level-sensitive interrupts that are to be synchronized to `pclk_intr` are selected. The `gpio_intrclk_en` signal can be used in the clock generation block to turn on `pclk_intr`.

The input signal is inverted for active-low level-sensitive interrupts. The same detection logic is used here as is used for active-high level-sensitive interrupts.

Figure 2-15 shows the generation of an active-low level-sensitive interrupt where the debounce circuitry is disabled.

Figure 2-15 Active-Low Level-Sensitive Interrupt Generation Timing



3

Parameter Descriptions

This chapter details all the configuration parameters. **You can use the coreConsultant GUI configuration reports to determine the actual configured state of the controller.** 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 controller in coreConsultant, all TCL functions and parameters are evaluated completely; and the resulting values are displayed where appropriate in the coreConsultant GUI reports.

The parameter descriptions in this chapter include the **Enabled:** attribute which indicates the values required to be set on other parameters before you can change the value of this parameter.

These tables define all of the user configuration options for this component.

- Basic Config on [page 38](#)
- Port A on [page 40](#)
- Port B on [page 44](#)
- Port C on [page 46](#)
- Port D on [page 48](#)

3.1 Basic Config Parameters

Table 3-1 Basic Config Parameters

| Label | Description |
|------------------------|--|
| Basic Config | |
| Add encoded parameters | <p>Adds the encoded parameters that provides the firmware an easy and quick way of identifying the DesignWare component within an I/O memory map. Some critical design-time options determine how a driver must interact with the peripheral. There is a minimal area overhead when you include these parameters. Additionally, this option allows a self-configurable single driver to be developed for each component.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0x0) ■ true (0x1) <p>Default Value: true Enabled: Always Parameter Name: GPIO_ADD_ENCODED_PARAMS</p> |
| APB Data Bus Width | <p>Specifies the width of APB Data Bus to which this component is attached.</p> <p>Values: 8, 16, 32 Default Value: 32 Enabled: Always Parameter Name: APB_DATA_WIDTH</p> |
| Number of Ports | <p>Selects the number of ports supported.</p> <p>Values: 1, 2, 3, 4 Default Value: 4 Enabled: Always Parameter Name: GPIO_NUM_PORTS</p> |
| DW_apb_gpio ID | <p>Includes/Excludes the DW_apb_gpio ID register. If set to Include, provides an ID code value (set with GPIO_ID_NUM) that the system can read.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Exclude (0x0) ■ Include (0x1) <p>Default Value: Include Enabled: Always Parameter Name: GPIO_ID</p> |
| DW_apb_gpio ID Width | <p>Specifies the GPIO ID register width. The width of the identification code that is configured in GPIO_ID_NUM.</p> <p>Values: 1, ..., 32 Default Value: 32 Enabled: GPIO_ID == 1 Parameter Name: GPIO_ID_WIDTH</p> |

Table 3-1 Basic Config Parameters (Continued)

| Label | Description |
|----------------------|--|
| DW_apb_gpio ID Value | <p>Specifies the GPIO ID Value. The ID code value that the system reads back when the device identification is present.</p> <p>Values: 0x0, ..., POW_2_GPIO_ID_WIDTH</p> <p>Default Value: 0x0</p> <p>Enabled: GPIO_ID == 1</p> <p>Parameter Name: GPIO_ID_NUM</p> |

3.2 Port A Parameters

Table 3-2 Port A Parameters

| Label | Description |
|---------------------------------------|--|
| Port A | |
| Port A Width | <p>This parameter configures the width of Port A.</p> <p>Values: 1, ..., 32</p> <p>Default Value: 8</p> <p>Enabled: Always</p> <p>Parameter Name: GPIO_PWIDTH_A</p> |
| Port A Auxiliary H/W | <p>Port A can be configured with this parameter to have external, auxiliary hardware signals controlling the data and the direction of Port A rather than the software. If set to 0, the functionality for the hardware/software multiplexing is not implemented.</p> <p>Values:</p> <ul style="list-style-type: none"> ▪ Exclude (0x0) ▪ Include (0x1) <p>Default Value: Exclude</p> <p>Enabled: Always</p> <p>Parameter Name: GPIO_HW_PORTA</p> |
| Port A controlled from single source? | <p>Specifies the Port A hardware/software control. If set, all bits of Port A are either entirely under software control (if Port A Auxiliary H/W is excluded) or entirely under hardware control (if Port A Auxiliary H/W is included). If this parameter is not set, the "gpio_sw_porta" register determines which bits of the port are under hardware control and which are under software control.</p> <p>Values:</p> <ul style="list-style-type: none"> ▪ false (0x0) ▪ true (0x1) <p>Default Value: true</p> <p>Enabled: GPIO_HW_PORTA==1</p> <p>Parameter Name: GPIO_PORTA_SINGLE_CTL</p> |
| Port A Default Direction | <p>This parameter configures the default direction of Port A after power-on reset.</p> <p>Values:</p> <ul style="list-style-type: none"> ▪ Input (0) ▪ Output (1) <p>Default Value: Input</p> <p>Enabled: Always</p> <p>Parameter Name: GPIO_DFLT_DIR_A</p> |

Table 3-2 Port A Parameters (Continued)

| Label | Description |
|---------------------------|--|
| Port A Default Mode | <p>This parameter sets the default mode of Port A after a Power-On-Reset to either software or hardware.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ S/W (0) ■ H/W (1) <p>Default Value: S/W Enabled: GPIO_HW_PORTA == 1 Parameter Name: GPIO_DFLT_SRC_A</p> |
| Port A Debounce Logic | <p>Includes the Debounce logic and includes the capability of debouncing interrupts using an external slow clock.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Exclude (0x0) ■ Include (0x1) <p>Default Value: Exclude Enabled: GPIO_PORTA_INTR == 1 Parameter Name: GPIO_DEBOUNCE</p> |
| Port A Interrupt | <p>If Port A is required to be used as an interrupt source, then set this parameter to 1.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Exclude (0x0) ■ Include (0x1) <p>Default Value: Include Enabled: Always Parameter Name: GPIO_PORTA_INTR</p> |
| Port A Interrupt Polarity | <p>Sets the polarity on the output of Port A. The single combined interrupt and the separate individual interrupts share a common polarity.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Active Low (0) ■ Active High (1) <p>Default Value: Active Low Enabled: GPIO_PORTA_INTR == 1 Parameter Name: GPIO_INT_POL</p> |

Table 3-2 Port A Parameters (Continued)

| Label | Description |
|---------------------------------------|---|
| Single Interrupt Generated? | <p>Specifies that Port A can be configured to generate multiple interrupt signals or a single combined interrupt flag. When set to 1, the component generates a single combined interrupt flag.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0) ■ true (1) <p>Default Value: false Enabled: GPIO_PORTA_INTR == 1 Parameter Name: GPIO_INTR_IO</p> |
| Port A Read Back Data Synchronization | <p>This parameter controls the inclusion of metastability registers on the read back path when reading the external input signal gpio_ext_porta from the gpio_ext_porta memory-mapped registers.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Exclude (0) ■ Include (1) <p>Default Value: Exclude Enabled: Always Parameter Name: GPIO_PA_SYNC_EXT_DATA</p> |
| Port A Software Register Reset Value | <p>Specifies the Power-on-Reset value of Port A Software Register gpio_swporta.</p> <p>Values: 0x0, ..., 0xff Default Value: {multi} {GPIO_PWIDTH_A} {0b0} Enabled: Always Parameter Name: GPIO_SWPORTA_RESET</p> |
| Port A Interrupt Synchronisation | <p>Synchronizes Port A interrupts. If set, metastability flip-flops for Port A interrupts are instantiated as part of the component. Otherwise, metastability flip-flops are not instantiated, and it is assumed that interrupt synchronization is taken care outside of the component.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Exclude (0) ■ Include (1) <p>Default Value: Include Enabled: Always Parameter Name: GPIO_PA_SYNC_INTERRUPTS</p> |

Table 3-2 Port A Parameters (Continued)

| Label | Description |
|--|--|
| Port A Clock Domain Crossing Synchronization Depth | <p>Sets the number of synchronization stages to be placed on clock domain crossing signals of port A.</p> <ul style="list-style-type: none"> ■ 2: 2-stage synchronization with positive-edge capturing at both the stages ■ 3: 3-stage synchronization with positive-edge capturing at all stages ■ 4: 4-stage synchronization with positive-edge capturing at all stages <p>Values: 2, 3, 4 Default Value: 2 Enabled: GPIO_PA_SYNC_INTERRUPTS==1 GPIO_PA_SYNC_EXT_DATA==1 Parameter Name: GPIO_PA_SYNC_DEPTH</p> |
| Port A Interrupt Detection | <p>If Port A is required to generate interrupt on both rising and falling edges of the external input signal, then set this parameter to 1.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Exclude (0) ■ Include (1) <p>Default Value: Exclude Enabled: GPIO_PORTA_INTR == 1 Parameter Name: GPIO_INT_BOTH_EDGE</p> |

3.3 Port B Parameters

Table 3-3 Port B Parameters

| Label | Description |
|---------------------------------------|--|
| Port B | |
| Port B Width | <p>This parameter configures the width of Port B.</p> <p>Values: 1, ..., 32</p> <p>Default Value: 8</p> <p>Enabled: GPIO_NUM_PORTS > 1</p> <p>Parameter Name: GPIO_PWIDTH_B</p> |
| Port B Auxiliary H/W | <p>Indicates Port B Auxiliary H/W support. When set to 1, Port B has external, auxiliary hardware signals controlling the data and the direction of Port B rather than the software. If set to 0, then the functionality for the hardware-software multiplexing is not implemented.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Exclude (0x0) ■ Include (0x1) <p>Default Value: Exclude</p> <p>Enabled: GPIO_NUM_PORTS > 1</p> <p>Parameter Name: GPIO_HW_PORTB</p> |
| Port B controlled from single source? | <p>Indicates Port B Hardware/Software control. If set, all bits of Port B are either entirely under software control (if Port B Auxiliary H/W is excluded) or entirely under hardware control (if Port B Auxiliary H/W is included). If this parameter is not set, then the "gpio_sw_portb" register determines which bits of the port are under hardware control and which are under software control.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0x0) ■ true (0x1) <p>Default Value: true</p> <p>Enabled: GPIO_HW_PORTB==1 && GPIO_NUM_PORTS>1</p> <p>Parameter Name: GPIO_PORTB_SINGLE_CTL</p> |
| Port B Default Direction | <p>This parameter sets the default direction of Port B after Power-on-Reset.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Input (0) ■ Output (1) <p>Default Value: Input</p> <p>Enabled: GPIO_NUM_PORTS > 1</p> <p>Parameter Name: GPIO_DFLT_DIR_B</p> |

Table 3-3 Port B Parameters (Continued)

| Label | Description |
|--|---|
| Port B Default Mode | <p>Indicates default mode if the auxiliary H/W is supported on port B. The default source of the input data, the output data, and the control of Port B are configurable. This parameter sets the reset value of the gpio_portb_ctl register.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ S/W (0) ■ H/W (1) <p>Default Value: S/W Enabled: GPIO_NUM_PORTS > 1 && GPIO_HW_PORTB == 1 Parameter Name: GPIO_DFLT_SRC_B</p> |
| Port B Read Back Data Synchronization | <p>This parameter controls the inclusion of metastability registers on the read back path when reading the external input signal gpio_ext_portb from the gpio_ext_portb memory-mapped registers.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Exclude (0) ■ Include (1) <p>Default Value: Exclude Enabled: GPIO_NUM_PORTS > 1 Parameter Name: GPIO_PB_SYNC_EXT_DATA</p> |
| Port B Clock Domain Crossing Synchronization Depth | <p>Sets the number of synchronization stages to be placed on clock domain crossing signals of port B.</p> <ul style="list-style-type: none"> ■ 2: 2-stage synchronization with positive-edge capturing at both the stages ■ 3: 3-stage synchronization with positive-edge capturing at all stages ■ 4: 4-stage synchronization with positive-edge capturing at all stages <p>Values: 2, 3, 4 Default Value: 2 Enabled: GPIO_PB_SYNC_EXT_DATA==1 && GPIO_NUM_PORTS>1 Parameter Name: GPIO_PB_SYNC_DEPTH</p> |
| Port B Software Register Reset Value | <p>Indicates the Power-on-Reset value of Port B Software Register. This is the reset value of the gpio_swportb register.</p> <p>Values: 0x0, ..., 0xff Default Value: {multi} {GPIO_PWIDTH_B} {0b0} Enabled: GPIO_NUM_PORTS > 1 Parameter Name: GPIO_SWPORTB_RESET</p> |

3.4 Port C Parameters

Table 3-4 Port C Parameters

| Label | Description |
|---------------------------------------|--|
| Port C | |
| Port C Width | <p>This parameter sets the width of Port C.</p> <p>Values: 1, ..., 32</p> <p>Default Value: 8</p> <p>Enabled: GPIO_NUM_PORTS > 2</p> <p>Parameter Name: GPIO_PWIDTH_C</p> |
| Port C Auxiliary H/W | <p>Indicates the Port C Auxiliary H/W Support. When set to 1, Port C has external, auxiliary hardware signals controlling the data and the direction of Port C, rather than the software. If set to 0, then the functionality for the hardware-software multiplexing is not implemented.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Exclude (0x0) ■ Include (0x1) <p>Default Value: Exclude</p> <p>Enabled: GPIO_NUM_PORTS > 2</p> <p>Parameter Name: GPIO_HW_PORTC</p> |
| Port C controlled from single source? | <p>Indicates Port C Hardware/Software Control. If set, all bits of Port C are either entirely under software control (if Port C Auxiliary H/W is excluded) or entirely under hardware control (if Port C Auxiliary H/W is included). If this parameter is not set, then the "gpio_sw_portc" register determines which bits of the port are under hardware control and which are under software control.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0x0) ■ true (0x1) <p>Default Value: true</p> <p>Enabled: GPIO_HW_PORTC==1 && GPIO_NUM_PORTS > 2</p> <p>Parameter Name: GPIO_PORTC_SINGLE_CTL</p> |
| Port C Default Direction | <p>This parameter sets the default direction of Port C after Power-on-Reset.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Input (0) ■ Output (1) <p>Default Value: Input</p> <p>Enabled: GPIO_NUM_PORTS > 2</p> <p>Parameter Name: GPIO_DFLT_DIR_C</p> |

Table 3-4 Port C Parameters (Continued)

| Label | Description |
|--|---|
| Port C Default Mode | <p>Indicates default mode if auxiliary H/W supported on port C. The default source of the input data, the output data, and the control of Port C are configurable. This parameter sets the reset value of the gpio_portc_ctl register. Power On Reset to either S/W or H/W.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ S/W (0) ■ H/W (1) <p>Default Value: S/W</p> <p>Enabled: GPIO_NUM_PORTS > 2 && GPIO_HW_PORTC == 1</p> <p>Parameter Name: GPIO_DFLT_SRC_C</p> |
| Port C Read Back Data Synchronization | <p>Indicates Port C Read Back Data Synchronization. This parameter controls inclusion of metastability registers on the read back path when reading the external input signal gpio_ext_portc from the gpio_ext_portc memory-mapped registers.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Exclude (0) ■ Include (1) <p>Default Value: Exclude</p> <p>Enabled: GPIO_NUM_PORTS > 2</p> <p>Parameter Name: GPIO_PC_SYNC_EXT_DATA</p> |
| Port C Clock Domain Crossing Synchronization Depth | <p>Sets the number of synchronization stages to be placed on clock domain crossing signals of port C.</p> <ul style="list-style-type: none"> ■ 2: 2-stage synchronization with positive-edge capturing at both the stages ■ 3: 3-stage synchronization with positive-edge capturing at all stages ■ 4: 4-stage synchronization with positive-edge capturing at all stages <p>Values: 2, 3, 4</p> <p>Default Value: 2</p> <p>Enabled: GPIO_PC_SYNC_EXT_DATA==1 && GPIO_NUM_PORTS>2</p> <p>Parameter Name: GPIO_PC_SYNC_DEPTH</p> |
| Port C Software Register Reset Value | <p>Indicates Power-on-Reset value of the Port C Software Register. This is the reset value of the gpio_swportc register.</p> <p>Values: 0x0, ..., 0xff</p> <p>Default Value: {multi} {GPIO_PWIDTH_C} {0b0}</p> <p>Enabled: GPIO_NUM_PORTS > 2</p> <p>Parameter Name: GPIO_SWPORTC_RESET</p> |

3.5 Port D Parameters

Table 3-5 Port D Parameters

| Label | Description |
|---------------------------------------|--|
| Port D | |
| Port D Width | <p>This parameter configures the width of Port D.</p> <p>Values: 1, ..., 32</p> <p>Default Value: 8</p> <p>Enabled: GPIO_NUM_PORTS > 3</p> <p>Parameter Name: GPIO_PWIDTH_D</p> |
| Port D Auxiliary H/W | <p>Indicates Port D Auxiliary H/W Support. When set to 1, Port D has external, auxiliary hardware signals controlling the data and the direction of Port D, rather than software. If set to 0, then the functionality for the hardware-software multiplexing is not implemented.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Exclude (0x0) ■ Include (0x1) <p>Default Value: Exclude</p> <p>Enabled: GPIO_NUM_PORTS > 3</p> <p>Parameter Name: GPIO_HW_PORTD</p> |
| Port D controlled from single source? | <p>Indicates Port D Hardware/Software Control. If set, all bits of Port D are either entirely under software control (if Port D Auxiliary H/W is excluded) or entirely under hardware control (if Port C Auxiliary H/W is included). If this parameter is not set, then the "gpio_sw_portd" register determines which bits of the port are under hardware control and which are under software control.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ false (0x0) ■ true (0x1) <p>Default Value: true</p> <p>Enabled: GPIO_HW_PORTD==1 && GPIO_NUM_PORTS > 3</p> <p>Parameter Name: GPIO_PORTD_SINGLE_CTL</p> |
| Port D Default Direction | <p>This parameter sets the default direction of Port D after Power On Reset.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Input (0) ■ Output (1) <p>Default Value: Input</p> <p>Enabled: GPIO_NUM_PORTS > 3</p> <p>Parameter Name: GPIO_DFLT_DIR_D</p> |

Table 3-5 Port D Parameters (Continued)

| Label | Description |
|--|---|
| Port D Default Mode | <p>Indicates the default mode if auxiliary H/W is supported on port D. The default source of the input data, the output data, and the control of Port D are configurable. This parameter sets the reset value of the gpio_portd_ctl register.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ S/W (0) ■ H/W (1) <p>Default Value: S/W Enabled: GPIO_NUM_PORTS > 3 && GPIO_HW_PORTD == 1 Parameter Name: GPIO_DFLT_SRC_D</p> |
| Port D Read Back Data Synchronization | <p>Indicates the Port D Read Back Data Synchronization. This parameter controls the inclusion of metastability registers on the read back path when reading the gpio_ext_portd external input signal from the gpio_ext_portd memory-mapped registers.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ Exclude (0) ■ Include (1) <p>Default Value: Exclude Enabled: GPIO_NUM_PORTS > 3 Parameter Name: GPIO_PD_SYNC_EXT_DATA</p> |
| Port D Clock Domain Crossing Synchronization Depth | <p>Sets the number of synchronization stages to be placed on clock domain crossing signals of port D.</p> <ul style="list-style-type: none"> ■ 2: 2-stage synchronization with positive-edge capturing at both the stages ■ 3: 3-stage synchronization with positive-edge capturing at all stages ■ 4: 4-stage synchronization with positive-edge capturing at all stages <p>Values: 2, 3, 4 Default Value: 2 Enabled: GPIO_PD_SYNC_EXT_DATA==1 && GPIO_NUM_PORTS>3 Parameter Name: GPIO_PD_SYNC_DEPTH</p> |
| Port D Software Register Reset Value | <p>Indicates Power-on-Reset value of Port D Software Register. This is the reset value of the gpio_swportd register.</p> <p>Values: 0x0, ..., 0xff Default Value: {multi} {GPIO_PWIDTH_D} {0b0} Enabled: GPIO_NUM_PORTS > 3 Parameter Name: GPIO_SWPORTD_RESET</p> |

4

Signal Descriptions

This chapter details all possible I/O signals in the controller. 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 controller. It is for reference purposes only.

When you configure the controller 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 controller in coreConsultant, all TCL functions and parameters are evaluated completely; and the resulting values are displayed where appropriate in the coreConsultant GUI reports.

In addition to describing the function of each signal, the signal descriptions in this chapter include the following information:

Active State: Indicates whether the signal is active high or active low. When a signal is not intended to be used in a particular application, then this signal needs to be tied or driven to the inactive state (opposite of the active state).

Registered: Indicates whether or not the signal is registered directly inside the IP boundary without intervening logic (excluding simple buffers). A value of No does not imply that the signal is not synchronous, only that there is some combinatorial logic between the signal's origin or destination register and the boundary of the controller. A value of N/A indicates that this information is not provided for this IP title.

Synchronous to: Indicates which clock(s) in the IP sample this input (drive for an output) when considering all possible configurations. A particular configuration might not have all of the clocks listed. This clock might not be the same as the clock that your application logic should use to clock (sample/drive) this pin. For more details, consult the clock section in the databook.

Exists: Name of configuration parameter(s) that populates this signal in your configuration.

Validated by: Assertion or de-assertion of signal(s) that validates the signal being described.

Attributes used with Synchronous To

- Clock name - The name of the clock that samples an input or drive and output.
- None - This attribute may be used for clock inputs, hard-coded outputs, feed-through (direct or combinatorial), dangling inputs, unused inputs and asynchronous outputs.
- Asynchronous - This attribute is used for asynchronous inputs and asynchronous resets.

The I/O signals are grouped as follows:

- APB Interface on [page 53](#)
- Port A on [page 57](#)
- Port B on [page 59](#)
- Port C on [page 61](#)
- Port D on [page 63](#)

4.1 APB Interface Signals

| | | |
|-------------|---|------------------|
| pclk | - | gpio_intr_flag |
| pclk_intr | - | gpio_intr |
| presetn | - | gpio_intr_flag_n |
| penable | - | gpio_intr_n |
| pwrite | - | gpio_intrclk_en |
| pwrdata | - | prdata |
| paddr | - | |
| psel | - | |
| dbclk | - | |
| dbclk_res_n | - | |
| scan_mode | - | |

Table 4-1 APB Interface Signals

| Port Name | I/O | Description |
|-----------|-----|--|
| pclk | I | <p>APB clock. This clock times all bus transfers. All signal timings are related to the rising edge of pclk.</p> <p>Exists: Always</p> <p>Synchronous To: None</p> <p>Registered: N/A</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p> |
| pclk_intr | I | <p>Optional. APB clock used in detection of edge-sensitive interrupts and in synchronization of level-sensitive interrupts. (Free-running implementation)</p> <p>Exists: GPIO_PORTA_INTR==1</p> <p>Synchronous To: pclk</p> <p>Registered: N/A</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: N/A</p> |
| presetn | I | <p>APB Reset Signal. The bus reset signal is used to reset the system and the bus on the DesignWare interface. Asynchronous assertion, synchronous de-assertion. The reset must be synchronously de-asserted after the rising edge of pclk. DW_apb_gpio does not contain logic to perform this synchronization, so it must be provided externally.</p> <p>Exists: Always</p> <p>Synchronous To: Asynchronous</p> <p>Registered: N/A</p> <p>Power Domain: SINGLE_DOMAIN</p> <p>Active State: Low</p> |

Table 4-1 APB Interface Signals (Continued)

| Port Name | I/O | Description |
|------------------------------|-----|---|
| penable | I | APB enable control that indicates the second cycle of the APB. Exists: Always Synchronous To: pclk Registered: No Power Domain: SINGLE_DOMAIN Active State: High |
| pwrite | I | APB write control. Exists: Always Synchronous To: pclk Registered: No Power Domain: SINGLE_DOMAIN Active State: High |
| pwdata[(APB_DATA_WIDTH-1):0] | I | APB write data bus. Exists: Always Synchronous To: pclk Registered: Yes Power Domain: SINGLE_DOMAIN Active State: N/A |
| paddr[GPIO_ADDR_SLICE_LHS:0] | I | APB address bus. Exists: Always Synchronous To: pclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A |
| psel | I | APB peripheral select. Exists: Always Synchronous To: pclk Registered: No Power Domain: SINGLE_DOMAIN Active State: High |
| dbclk | I | Optional. Debounce clock used to remove glitches from interrupt signal lines. Exists: (GPIO_DEBOUNCE==1) && (GPIO_PORTA_INTR==1) Synchronous To: None Registered: N/A Power Domain: SINGLE_DOMAIN Active State: N/A |

Table 4-1 APB Interface Signals (Continued)

| Port Name | I/O | Description |
|--------------------------------|-----|---|
| dbclk_res_n | I | Optional. Reset from debounce clock domain with asynchronous assertion, synchronous de-assertion. Exists: (GPIO_DEBOUNCE==1) && (GPIO_PORTA_INTR==1) Synchronous To: Asynchronous Registered: N/A Power Domain: SINGLE_DOMAIN Active State: Low |
| scan_mode | I | Optional. Active-high to indicate part in scan mode. This signal must be asserted during scan testing in order to ensure that all flip-flops in the design can be controlled and observed during scan testing; at all other times, this signal must be de-asserted. Exists: GPIO_DEBOUNCE==1 Synchronous To: Asynchronous Registered: No Power Domain: SINGLE_DOMAIN Active State: High |
| gpio_intr_flag | O | Optional. Active High Combined interrupt status. Exists: (GPIO_PORTA_INTR==1) && (GPIO_INT_POL==1) && (GPIO_INTR_IO==GPIO_COMBINED) Synchronous To: pclk Registered: No Power Domain: SINGLE_DOMAIN Active State: High |
| gpio_intr[(GPIO_PWIDTH_A-1):0] | O | Optional. Active high Interrupt status to interrupt controller. Exists: (GPIO_PORTA_INTR==1) && (GPIO_INT_POL==1) && (GPIO_INTR_IO==GPIO_INDIVIDUAL) Synchronous To: pclk Registered: No Power Domain: SINGLE_DOMAIN Active State: High |
| gpio_intr_flag_n | O | Optional. Active low Combined interrupt status. Exists: (GPIO_PORTA_INTR==1) && (GPIO_INT_POL==0) && (GPIO_INTR_IO==GPIO_COMBINED) Synchronous To: pclk Registered: No Power Domain: SINGLE_DOMAIN Active State: Low |

Table 4-1 APB Interface Signals (Continued)

| Port Name | I/O | Description |
|----------------------------------|-----|---|
| gpio_intr_n[(GPIO_PWIDTH_A-1):0] | O | Optional. Active low Interrupt status to interrupt controller. Exists: (GPIO_PORTA_INTR==1) && (GPIO_INT_POL==0) && (GPIO_INTR_IO==GPIO_INDIVIDUAL) Synchronous To: pclk Registered: No Power Domain: SINGLE_DOMAIN Active State: Low |
| gpio_intrclk_en | O | Optional. When this is asserted, it signals that pclk_intr must be running to detect interrupts. This is asserted when either edge-sensitive interrupts are enabled or synchronization of level-sensitive interrupts is enabled. Exists: GPIO_PORTA_INTR==1 Synchronous To: pclk Registered: Yes Power Domain: SINGLE_DOMAIN Active State: High |
| prdata[(APB_DATA_WIDTH-1):0] | O | APB read data. Exists: Always Synchronous To: pclk Registered: Yes Power Domain: SINGLE_DOMAIN Active State: N/A |

4.2 Port A Signals



Table 4-2 Port A Signals

| Port Name | I/O | Description |
|-------------------------------------|-----|---|
| aux_porta_out[(GPIO_PWIDTH_A-1):0] | I | Optional. Auxiliary hardware output data (Port A). Exists: GPIO_HW_PORTA==1 Synchronous To: pclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A |
| aux_porta_en[(GPIO_PWIDTH_A-1):0] | I | Optional. Auxiliary hardware direction control (Port A). 0 = Input 1 = Output Exists: GPIO_HW_PORTA==1 Synchronous To: pclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A |
| aux_porta_in[(GPIO_PWIDTH_A-1):0] | O | Optional. Auxiliary hardware input data (Port A). Exists: GPIO_HW_PORTA==1 Synchronous To: pclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A |
| gpio_ext_porta[(GPIO_PWIDTH_A-1):0] | I | Input data (Port A). Exists: Always Synchronous To: ((GPIO_PA_SYNC_INTERRUPTS == 1 && GPIO_PORTA_INTR == 1) (GPIO_PA_SYNC_EXT_DATA == 1) (GPIO_DEBOUNCE == 1)) ? "Asynchronous" : "pclk" Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A |

Table 4-2 Port A Signals (Continued)

| Port Name | I/O | Description |
|-------------------------------------|-----|---|
| gpio_porta_dr[(GPIO_PWIDTH_A-1):0] | O | Output data (Port A). Exists: Always Synchronous To: pclk Registered: GPIO_HW_PORTA==0? Yes : No Power Domain: SINGLE_DOMAIN Active State: N/A |
| gpio_porta_ddr[(GPIO_PWIDTH_A-1):0] | O | Data direction control (Port A). Exists: Always Synchronous To: pclk Registered: GPIO_HW_PORTA==0? Yes : No Power Domain: SINGLE_DOMAIN Active State: N/A |

4.3 Port B Signals



Table 4-3 Port B Signals

| Port Name | I/O | Description |
|-------------------------------------|-----|--|
| aux_portb_out[(GPIO_PWIDTH_B-1):0] | I | Optional. Auxiliary hardware output data (Port B). Exists: (GPIO_HW_PORTB==1) && (GPIO_NUM_PORTS>1) Synchronous To: pclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A |
| aux_portb_en[(GPIO_PWIDTH_B-1):0] | I | Optional. Auxiliary hardware direction control (Port B). 0 = Input 1 = Output Exists: (GPIO_HW_PORTB==1) && (GPIO_NUM_PORTS>1) Synchronous To: pclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A |
| aux_portb_in[(GPIO_PWIDTH_B-1):0] | O | Optional. Auxiliary hardware input data (Port B). Exists: (GPIO_HW_PORTB==1) && (GPIO_NUM_PORTS>1) Synchronous To: pclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A |
| gpio_ext_portb[(GPIO_PWIDTH_B-1):0] | I | Input data (Port B). Exists: GPIO_NUM_PORTS>1 Synchronous To: pclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A |

Table 4-3 Port B Signals (Continued)

| Port Name | I/O | Description |
|-------------------------------------|-----|---|
| gpio_portb_dr[(GPIO_PWIDTH_B-1):0] | O | Output data (Port B). Exists: GPIO_NUM_PORTS>1 Synchronous To: pclk Registered: GPIO_HW_PORTB==0? Yes : No Power Domain: SINGLE_DOMAIN Active State: N/A |
| gpio_portb_ddr[(GPIO_PWIDTH_B-1):0] | O | Data direction control (Port B). Exists: GPIO_NUM_PORTS>1 Synchronous To: pclk Registered: GPIO_HW_PORTB==0? Yes : No Power Domain: SINGLE_DOMAIN Active State: N/A |

4.4 Port C Signals



Table 4-4 Port C Signals

| Port Name | I/O | Description |
|-------------------------------------|-----|--|
| aux_portc_out[(GPIO_PWIDTH_C-1):0] | I | Optional. Auxiliary hardware output data (Port C). Exists: (GPIO_HW_PORTC==1) && (GPIO_NUM_PORTS>2) Synchronous To: pclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A |
| aux_portc_en[(GPIO_PWIDTH_C-1):0] | I | Optional. Auxiliary hardware direction control (Port C). 0 = Input 1 = Output Exists: (GPIO_HW_PORTC==1) && (GPIO_NUM_PORTS>2) Synchronous To: pclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A |
| aux_portc_in[(GPIO_PWIDTH_C-1):0] | O | Optional. Auxiliary hardware input data (Port C). Exists: (GPIO_HW_PORTC==1) && (GPIO_NUM_PORTS>2) Synchronous To: pclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A |
| gpio_ext_portc[(GPIO_PWIDTH_C-1):0] | I | Input data (Port C). Exists: GPIO_NUM_PORTS>2 Synchronous To: pclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A |

Table 4-4 Port C Signals (Continued)

| Port Name | I/O | Description |
|-------------------------------------|-----|---|
| gpio_portc_dr[(GPIO_PWIDTH_C-1):0] | O | Output data (Port C). Exists: GPIO_NUM_PORTS>2 Synchronous To: pclk Registered: GPIO_HW_PORTC==0? Yes : No Power Domain: SINGLE_DOMAIN Active State: N/A |
| gpio_portc_ddr[(GPIO_PWIDTH_C-1):0] | O | Data direction control (Port C). Exists: GPIO_NUM_PORTS>2 Synchronous To: pclk Registered: GPIO_HW_PORTC==0? Yes : No Power Domain: SINGLE_DOMAIN Active State: N/A |

4.5 Port D Signals

aux_portd_out -
 aux_portd_en -
 gpio_ext_portd -

- aux_portd_in
 - gpio_portd_dr
 - gpio_portd_dds

Table 4-5 Port D Signals

| Port Name | I/O | Description |
|-------------------------------------|-----|--|
| aux_portd_out[(GPIO_PWIDTH_D-1):0] | I | Optional. Auxiliary hardware output data (Port D). Exists: (GPIO_HW_PORTD==1) && (GPIO_NUM_PORTS>3) Synchronous To: pclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A |
| aux_portd_en[(GPIO_PWIDTH_D-1):0] | I | Optional. Auxiliary hardware direction control (Port D). 0 = Input 1 = Output Exists: (GPIO_HW_PORTD==1) && (GPIO_NUM_PORTS>3) Synchronous To: pclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A |
| aux_portd_in[(GPIO_PWIDTH_D-1):0] | O | Optional. Auxiliary hardware input data (Port D). Exists: (GPIO_HW_PORTD==1) && (GPIO_NUM_PORTS>3) Synchronous To: pclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A |
| gpio_ext_portd[(GPIO_PWIDTH_D-1):0] | I | Input data (Port D). Exists: GPIO_NUM_PORTS>3 Synchronous To: pclk Registered: No Power Domain: SINGLE_DOMAIN Active State: N/A |

Table 4-5 Port D Signals (Continued)

| Port Name | I/O | Description |
|-------------------------------------|-----|---|
| gpio_portd_dr[(GPIO_PWIDTH_D-1):0] | O | Output data (Port D). Exists: GPIO_NUM_PORTS>3 Synchronous To: pclk Registered: GPIO_HW_PORTD==0? Yes : No Power Domain: SINGLE_DOMAIN Active State: N/A |
| gpio_portd_ddr[(GPIO_PWIDTH_D-1):0] | O | Data direction control (Port D). Exists: GPIO_NUM_PORTS>3 Synchronous To: pclk Registered: GPIO_HW_PORTD==0? Yes : No Power Domain: SINGLE_DOMAIN Active State: N/A |

5

Register Descriptions

This chapter details all possible registers in the controller. They are arranged hierarchically into maps and blocks (banks). For configurable IP titles, your actual configuration might not contain all of these registers.

Attention: For configurable IP titles, do not use this document to determine the exact attributes of your register map. It is for reference purposes only.

When you configure the controller in coreConsultant, you must access the register attributes for your actual configuration at `workspace/report/ComponentRegisters.html` or `workspace/report/ComponentRegisters.xml` after you have completed the report creation activity. That report comes from the exact same source as this chapter but removes all the registers 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 Offset and Memory Access values 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 controller in coreConsultant, all TCL functions and parameters are evaluated completely; and the resulting values are displayed where appropriate in the coreConsultant GUI reports.

Exists Expressions

These expressions indicate the combination of configuration parameters required for a register, field, or block to exist in the memory map. The expression is only valid in the local context and does not indicate the conditions for existence of the parent. For example, the expression for a bit field in a register assumes that the register exists and does not include the conditions for existence of the register.

Offset

The term *Offset* is synonymous with *Address*.

Memory Access Attributes

The Memory Access attribute is defined as `<ReadBehavior>/<WriteBehavior>` which are defined in the following table.

Table 5-1 Possible Read and Write Behaviors

| Read (or Write) Behavior | Description |
|-----------------------------|--|
| RC | A read clears this register field. |
| RS | A read sets this register field. |
| RM | A read modifies the contents of this register field. |
| Wo | You can only write to this register once field. |
| W1C | A write of 1 clears this register field. |
| W1S | A write of 1 sets this register field. |
| W1T | A write of 1 toggles this register field. |
| W0C | A write of 0 clears this register field. |
| W0S | A write of 0 sets this register field. |
| W0T | A write of 0 toggles this register field. |
| WC | Any write clears this register field. |
| WS | Any write sets this register field. |
| WM | Any write toggles this register field. |
| no Read Behavior attribute | You cannot read this register. It is Write-Only. |
| no Write Behavior attribute | You cannot write to this register. It is Read-Only. |

Table 5-2 Memory Access Examples

| Memory Access | Description |
|---------------|--|
| R | Read-only register field. |
| W | Write-only register field. |
| R/W | Read/write register field. |
| R/W1C | You can read this register field. Writing 1 clears it. |
| RC/W1C | Reading this register field clears it. Writing 1 clears it. |
| R/Wo | You can read this register field. You can only write to it once. |

Special Optional Attributes

Some register fields might use the following optional attributes.

Table 5-3 Optional Attributes

| Attribute | Description |
|------------|--|
| Volatile | As defined by the IP-XACT specification. If true, indicates in the case of a write followed by read, or in the case of two consecutive reads, there is no guarantee as to what is returned by the read on the second transaction or that this return value is consistent with the write or read of the first transaction. The element implies there is some additional mechanism by which this field can acquire new values other than by reads/writes/resets and other access methods known to IP-XACT. For example, when the core updates the register field contents. |
| Testable | As defined by the IP-XACT specification. Possible values are unconstrained, untestable, readOnly, writeAsRead, restore. Untestable means that this field is untestable by a simple automated register test. For example, the read-write access of the register is controlled by a pin or another register. readOnly means that you should not write to this register; only read from it. This might apply for a register that modifies the contents of another register. |
| Reset Mask | As defined by the IP-XACT specification. Indicates that this register field has an unknown reset value. For example, the reset value is set by another register or an input pin; or the register is implemented using RAM. |
| * Varies | Indicates that the memory access (or reset) attribute (read, write behavior) is not fixed. For example, the read-write access of the register is controlled by a pin or another register. Or when the access depends on some configuration parameter; in this case the post-configuration report in coreConsultant gives the actual access value. |

Component Banks/Blocks

The following table shows the address blocks for each memory map. Follow the link for an address block to see a table of its registers.

Table 5-4 Address Banks/Blocks for Memory Map: DW_apb_gpio_mem_map

| Address Block | Description |
|---|--|
| DW_apb_gpio_addr_block on page 68 | DW_apb_gpio address block Exists: Always |

5.1 DW_apb_gpio_mem_map/DW_apb_gpio_addr_block Registers

DW_apb_gpio address block. Follow the link for the register to see a detailed description of the register.

Table 5-5 Registers for Address Block: DW_apb_gpio_mem_map/DW_apb_gpio_addr_block

| Register | Offset | Description |
|---|--------|--|
| GPIO_SWPORTA_DR on page 70 | 0x0 | Port A data register |
| GPIO_SWPORTA_DDR on page 71 | 0x4 | Port A Data Direction Register |
| GPIO_SWPORTA_CTL on page 72 | 0x8 | Port A data source register |
| GPIO_SWPORTB_DR on page 74 | 0xc | Port B data register |
| GPIO_SWPORTB_DDR on page 75 | 0x10 | Port B Data Direction Register |
| GPIO_SWPORTB_CTL on page 76 | 0x14 | Port B data source register |
| GPIO_SWPORTC_DR on page 78 | 0x18 | Port C data register |
| GPIO_SWPORTC_DDR on page 79 | 0x1c | Port C Data Direction Register |
| GPIO_SWPORTC_CTL on page 80 | 0x20 | Port C data source register |
| GPIO_SWPORTD_DR on page 82 | 0x24 | Port D data register |
| GPIO_SWPORTD_DDR on page 83 | 0x28 | Port D Data Direction Register |
| GPIO_SWPORTD_CTL on page 84 | 0x2c | Port D data source register |
| GPIO_INTEN on page 86 | 0x30 | Interrupt enable register Note: This register is available only if Port A is configured to generate... |
| GPIO_INTMASK on page 88 | 0x34 | Interrupt mask register Note: This register is available only if Port A is configured to generate... |
| GPIO_INTTYPE_LEVEL on page 90 | 0x38 | Interrupt level Note: This register is available only if Port A is configured to generate interrupts... |
| GPIO_INT_POLARITY on page 92 | 0x3c | Interrupt polarity Note: This register is available only if Port A is configured to generate interrupts... |
| GPIO_INTSTATUS on page 94 | 0x40 | Interrupt status Note: This register is available only if Port A is configured to generate interrupts... |
| GPIO_RAW_INTSTATUS on page 95 | 0x44 | Raw interrupt status Note: This register is available only if Port A is configured to generate... |
| GPIO_DEBOUNCE on page 96 | 0x48 | Debounce enable Note: This register is available only if Port A is configured to generate interrupts... |
| GPIO_PORTA_EOI on page 98 | 0x4c | Port A clear interrupt register Note: This register is available only if Port A is configured to... |
| GPIO_EXT_PORTA on page 100 | 0x50 | External port A register |

Table 5-5 Registers for Address Block: DW_apb_gpio_mem_map/DW_apb_gpio_addr_block (Continued)

| Register | Offset | Description |
|---|--------|--|
| GPIO_EXT_PORTB on page 101 | 0x54 | Port B external port register |
| GPIO_EXT_PORTC on page 102 | 0x58 | External port C register |
| GPIO_EXT_PORTD on page 103 | 0x5c | Port D external port register |
| GPIO_LS_SYNC on page 104 | 0x60 | Synchronization level |
| GPIO_ID_CODE on page 105 | 0x64 | GPIO ID code |
| GPIO_INT_BOTHEDGE on page 106 | 0x68 | Interrupt Both Edge type Note: This register is available only if PORT A is configured to generate... |
| GPIO_VER_ID_CODE on page 108 | 0x6c | GPIO Component Version |
| GPIO_CONFIG_REG2 on page 109 | 0x70 | GPIO Configuration Register 2 This register is a read-only register that is present when the configuration... |
| GPIO_CONFIG_REG1 on page 111 | 0x74 | GPIO Configuration Register 1 This register is present when the configuration parameter GPIO_ADD_ENCODED_PARAMS... |

5.1.1 GPIO_SWPORTA_DR

- **Name:** Port A data register
- **Description:** Port A data register
- **Size:** 32 bits
- **Offset:** 0x0
- **Exists:** Always

| | |
|----------------------|------|
| RSVD_GPIO_SWPORTA_DR | 31:y |
| GPIO_SWPORTA_DR | x:0 |

Table 5-6 Fields for Register: GPIO_SWPORTA_DR

| Bits | Name | Memory Access | Description |
|------|----------------------|---------------|--|
| 31:y | RSVD_GPIO_SWPORTA_DR | R | RSVD_GPIO_SWPORTA_DR Reserved bits - read as zero Value After Reset: 0x0 Exists: Always Range Variable[y]: GPIO_PWIDTH_A |
| x:0 | GPIO_SWPORTA_DR | R/W | Values written to this register are output on the I/O signals for Port A if the corresponding data direction bits for Port A are set to Output mode and the corresponding control bit for Port A is set to Software mode. The value read back is equal to the last value written to this register. Value After Reset: GPIO_SWPORTA_RESET Exists: Always Range Variable[x]: GPIO_PWIDTH_A - 1 |

5.1.2 GPIO_SWPORTA_DDR

- **Name:** Port A Data Direction Register
- **Description:** Port A Data Direction Register
- **Size:** 32 bits
- **Offset:** 0x4
- **Exists:** Always

| | |
|-----------------------|------|
| RSVD_GPIO_SWPORTA_DDR | 31:y |
| GPIO_SWPORTA_DDR | x:0 |

Table 5-7 Fields for Register: GPIO_SWPORTA_DDR

| Bits | Name | Memory Access | Description |
|------|-----------------------|---------------|--|
| 31:y | RSVD_GPIO_SWPORTA_DDR | R | RSVD_GPIO_SWPORTA_DDR Reserved bits - read as zero Value After Reset: 0x0 Exists: Always Range Variable[y]: GPIO_PWIDTH_A |
| x:0 | GPIO_SWPORTA_DDR | R/W | Values written to this register independently control the direction of the corresponding data bit in Port A. The default direction can be configured as input or output after system reset through the GPIO_DFLT_DIR_A parameter. Values: <ul style="list-style-type: none"> ■ 0x0 (IN): Input Direction ■ 0x1 (OUT): Output Direction Value After Reset: {(GPIO_DFLT_DIR_A==1) ? (pow (2, GPIO_PWIDTH_A)-1 : 0)} Exists: Always Range Variable[x]: GPIO_PWIDTH_A - 1 |

5.1.3 GPIO_SWPORTA_CTL

- **Name:** Port A data source register
- **Description:** Port A data source register
- **Size:** 32 bits
- **Offset:** 0x8
- **Exists:** GPIO_HW_PORTA==1

| | |
|-----------------------|------|
| RSVD_GPIO_SWPORTA_CTL | 31:y |
| GPIO_SWPORTA_CTL | x:0 |

Table 5-8 Fields for Register: GPIO_SWPORTA_CTL

| Bits | Name | Memory Access | Description |
|------|-----------------------|---------------|---|
| 31:y | RSVD_GPIO_SWPORTA_CTL | R | RSVD_GPIO_SWPORTA_CTL Reserved bits - read as zero Value After Reset: 0x0 Exists: Always Range Variable[y]: GPIO_SWPORTA_CTL_REG_SIZE |

Table 5-8 Fields for Register: GPIO_SWPORTA_CTL (Continued)

| Bits | Name | Memory Access | Description |
|------|------------------|---------------|--|
| x:0 | GPIO_SWPORTA_CTL | R/W | <p>The data and control source for a signal can come from either software or hardware; this bit selects between them. The default source is configurable through the GPIO_DFLT_DIR_A configuration parameter. If GPIO_PORTA_SINGLE_CTL = 0, the register will contain one bit for each bit of the signal. Upon reset in this case, the value of GPIO_DFLT_SRC_A is replicated across all bits of the signal so that all bits power up with the same operating mode. Furthermore, the default source of each bit of the signal can subsequently be changed by writing to the corresponding bit of this register. This register is not available unless GPIO_HW_PORTA = 1.</p> <p>If GPIO_PORTA_SINGLE_CTL = 1, then the reset value is GPIO_DFLT_SRC_A. If GPIO_PORTA_SINGLE_CTL = 0, then the reset value is {GPIO_PWIDTH_A{GPIO_DFLT_SRC_A in each bit}}.</p> <p>The values of this field depends on the size. Individual bit values will indicate whether the data source is hardware or software.</p> <p>0 : Indicates Software mode 1 : Indicates Hardware mode</p> <p>Value After Reset: GPIO_DFLT_SRC_RESET_A Exists: Always Range Variable[x]: GPIO_SWPORTA_CTL_REG_SIZE - 1</p> |

5.1.4 GPIO_SWPORTB_DR

- **Name:** Port B data register
- **Description:** Port B data register
- **Size:** 32 bits
- **Offset:** 0xc
- **Exists:** GPIO_NUM_PORTS>1

| | |
|----------------------|------|
| RSVD_GPIO_SWPORTB_DR | 31:y |
| GPIO_SWPORTB_DR | x:0 |

Table 5-9 Fields for Register: GPIO_SWPORTB_DR

| Bits | Name | Memory Access | Description |
|------|----------------------|---------------|--|
| 31:y | RSVD_GPIO_SWPORTB_DR | R | RSVD_GPIO_SWPORTB_DR Reserved bits - read as zero Value After Reset: 0x0 Exists: Always Range Variable[y]: GPIO_PWIDTH_B |
| x:0 | GPIO_SWPORTB_DR | R/W | Values written to this register are output on the I/O signals for Port B if the corresponding data direction bits for Port B are set to Output mode and the corresponding control bit for Port B is set to Software mode. The value read back is equal to the last value written to this register. Value After Reset: GPIO_SWPORTB_RESET Exists: Always Range Variable[x]: GPIO_PWIDTH_B - 1 |

5.1.5 GPIO_SWPORTB_DDR

- **Name:** Port B Data Direction Register
- **Description:** Port B Data Direction Register
- **Size:** 32 bits
- **Offset:** 0x10
- **Exists:** GPIO_NUM_PORTS>1

| | |
|-----------------------|------|
| RSVD_GPIO_SWPORTB_DDR | 31:y |
| GPIO_SWPORTB_DDR | x:0 |

Table 5-10 Fields for Register: GPIO_SWPORTB_DDR

| Bits | Name | Memory Access | Description |
|------|-----------------------|---------------|--|
| 31:y | RSVD_GPIO_SWPORTB_DDR | R | RSVD_GPIO_SWPORTB_DDR Reserved bits - read as zero Value After Reset: 0x0 Exists: Always Range Variable[y]: GPIO_PWIDTH_B |
| x:0 | GPIO_SWPORTB_DDR | R/W | Values written to this register independently control the direction of the corresponding data bit in Port B. The default direction can be configured as input or output after system reset through the GPIO_DFLT_DIR_B parameter. Values: <ul style="list-style-type: none"> ■ 0x0 (IN): Input Direction (default) ■ 0x1 (OUT): Output Direction Value After Reset: {(GPIO_DFLT_DIR_B==1) ? (pow (2, GPIO_PWIDTH_B)-1 : 0)} Exists: Always Range Variable[x]: GPIO_PWIDTH_B - 1 |

5.1.6 GPIO_SWPORTB_CTL

- **Name:** Port B data source register
- **Description:** Port B data source register
- **Size:** 32 bits
- **Offset:** 0x14
- **Exists:** GPIO_HW_PORTB==1 && GPIO_NUM_PORTS>1

| | |
|-----------------------|------|
| RSVD_GPIO_SWPORTB_CTL | 31:y |
| GPIO_SWPORTB_CTL | x:0 |

Table 5-11 Fields for Register: GPIO_SWPORTB_CTL

| Bits | Name | Memory Access | Description |
|------|-----------------------|---------------|---|
| 31:y | RSVD_GPIO_SWPORTB_CTL | R | RSVD_GPIO_SWPORTB_CTL Reserved bits - read as zero Value After Reset: 0x0 Exists: Always Range Variable[y]: GPIO_SWPORTB_CTL_REG_SIZE |

Table 5-11 Fields for Register: GPIO_SWPORTB_CTL (Continued)

| Bits | Name | Memory Access | Description |
|------|------------------|---------------|---|
| x:0 | GPIO_SWPORTB_CTL | R/W | <p>The data and control source for a signal can come from either software or hardware; this bit selects between them. The default source is configurable through the GPIO_DFLT_DIR_B configuration parameter.</p> <p>If GPIO_PORTB_SINGLE_CTL = 0, the register will contain one bit for each bit of the signal. Upon reset in this case, the value of GPIO_DFLT_SRC_B is replicated across all bits of the signal so that all bits power up with the same operating mode. Furthermore, the default source of each bit of the signal can subsequently be changed by writing to the corresponding bit of this register. This register is not available unless GPIO_HW_PORTB = 1. </p> <p>If GPIO_PORTB_SINGLE_CTL = 1, then the reset value is GPIO_DFLT_SRC_B. If GPIO_PORTB_SINGLE_CTL = 0, then the reset value is {GPIO_PWIDTH_B{GPIO_DFLT_SRC_B in each bit}}.</p> <p>The values of this field depends on the size. Individual bit values will indicate whether the data source is hardware or software.</p> <p>0 : Indicates Software mode 1 : Indicates Hardware mode</p> <p>Value After Reset: GPIO_DFLT_SRC_RESET_B Exists: Always Range Variable[x]: GPIO_SWPORTB_CTL_REG_SIZE - 1</p> |

5.1.7 GPIO_SWPORTC_DR

- **Name:** Port C data register
- **Description:** Port C data register
- **Size:** 32 bits
- **Offset:** 0x18
- **Exists:** GPIO_NUM_PORTS>2

| | |
|----------------------|------|
| RSVD_GPIO_SWPORTC_DR | 31:y |
| GPIO_SWPORTC_DR | x:0 |

Table 5-12 Fields for Register: GPIO_SWPORTC_DR

| Bits | Name | Memory Access | Description |
|------|----------------------|---------------|--|
| 31:y | RSVD_GPIO_SWPORTC_DR | R | RSVD_GPIO_SWPORTC_DR Reserved bits - read as zero Value After Reset: 0x0 Exists: Always Range Variable[y]: GPIO_PWIDTH_C |
| x:0 | GPIO_SWPORTC_DR | R/W | Values written to this register are output on the I/O signals for Port C if the corresponding data direction bits for Port C are set to Output mode and the corresponding control bit for Port C is set to Software mode. The value read back is equal to the last value written to this register. Reset Value: GPIO_SWPORTC_RESET Value After Reset: GPIO_SWPORTC_RESET Exists: Always Range Variable[x]: GPIO_PWIDTH_C - 1 |

5.1.8 GPIO_SWPORTC_DDR

- **Name:** Port C Data Direction Register
- **Description:** Port C Data Direction Register
- **Size:** 32 bits
- **Offset:** 0x1c
- **Exists:** GPIO_NUM_PORTS>2

| | |
|-----------------------|------|
| RSVD_GPIO_SWPORTC_DDR | 31:y |
| GPIO_SWPORTC_DDR | x:0 |

Table 5-13 Fields for Register: GPIO_SWPORTC_DDR

| Bits | Name | Memory Access | Description |
|------|-----------------------|---------------|--|
| 31:y | RSVD_GPIO_SWPORTC_DDR | R | RSVD_GPIO_SWPORTC_DDR Reserved bits - read as zero Value After Reset: 0x0 Exists: Always Range Variable[y]: GPIO_PWIDTH_C |
| x:0 | GPIO_SWPORTC_DDR | R/W | Values written to this register independently control the direction of the corresponding data bit in Port C. The default direction can be configured as input or output after system reset through the GPIO_DFLT_DIR_C parameter. Values: <ul style="list-style-type: none"> ■ 0x0 (IN): Input Direction (default) ■ 0x1 (OUT): Output Direction Value After Reset: {(GPIO_DFLT_DIR_C==1) ? (pow (2, GPIO_PWIDTH_C)-1 : 0)} Exists: Always Range Variable[x]: GPIO_PWIDTH_C - 1 |

5.1.9 GPIO_SWPORTC_CTL

- **Name:** Port C data source register
- **Description:** Port C data source register
- **Size:** 32 bits
- **Offset:** 0x20
- **Exists:** GPIO_HW_PORTC==1 && GPIO_NUM_PORTS>2

| | |
|-----------------------|------|
| RSVD_GPIO_SWPORTC_CTL | 31:y |
| GPIO_SWPORTC_CTL | x:0 |

Table 5-14 Fields for Register: GPIO_SWPORTC_CTL

| Bits | Name | Memory Access | Description |
|------|-----------------------|---------------|---|
| 31:y | RSVD_GPIO_SWPORTC_CTL | R | RSVD_GPIO_SWPORTC_CTL Reserved bits - read as zero Value After Reset: 0x0 Exists: Always Range Variable[y]: GPIO_SWPORTC_CTL_REG_SIZE |

Table 5-14 Fields for Register: GPIO_SWPORTC_CTL (Continued)

| Bits | Name | Memory Access | Description |
|------|------------------|---------------|--|
| x:0 | GPIO_SWPORTC_CTL | R/W | <p>The data and control source for a signal can come from either software or hardware; this bit selects between them. The default source is configurable through the GPIO_DFLT_DIR_C configuration parameter. If GPIO_PORTC_SINGLE_CTL = 0, the register will contain one bit for each bit of the signal. Upon reset in this case, the value of GPIO_DFLT_SRC_C is replicated across all bits of the signal so that all bits power up with the same operating mode. Furthermore, the default source of each bit of the signal can subsequently be changed by writing to the corresponding bit of this register. This register is not available unless GPIO_HW_PORTC = 1.</p> <p>If GPIO_PORTC_SINGLE_CTL = 1, then the reset value is GPIO_DFLT_SRC_C. If GPIO_PORTC_SINGLE_CTL = 0, then the reset value is {GPIO_PWIDTH_C{GPIO_DFLT_SRC_C in each bit}}.</p> <p>The values of this field depends on the size. Individual bit values will indicate whether the data source is hardware or software.</p> <p>0 : Indicates Software mode 1 : Indicates Hardware mode</p> <p>Value After Reset: GPIO_DFLT_SRC_RESET_C Exists: Always Range Variable[x]: GPIO_SWPORTC_CTL_REG_SIZE - 1</p> |

5.1.10 GPIO_SWPORTD_DR

- **Name:** Port D data register
- **Description:** Port D data register
- **Size:** 32 bits
- **Offset:** 0x24
- **Exists:** GPIO_NUM_PORTS==4

| | |
|----------------------|------|
| RSVD_GPIO_SWPORTD_DR | 31:y |
| GPIO_SWPORTD_DR | x:0 |

Table 5-15 Fields for Register: GPIO_SWPORTD_DR

| Bits | Name | Memory Access | Description |
|------|----------------------|---------------|--|
| 31:y | RSVD_GPIO_SWPORTD_DR | R | RSVD_GPIO_SWPORTD_DR Reserved bits - read as zero Value After Reset: 0x0 Exists: Always Range Variable[y]: GPIO_PWIDTH_D |
| x:0 | GPIO_SWPORTD_DR | R/W | Values written to this register are output on the I/O signals for Port D if the corresponding data direction bits for Port D are set to Output mode and the corresponding control bit for Port D is set to Software mode. The value read back is equal to the last value written to this register. Value After Reset: GPIO_SWPORTD_RESET Exists: Always Range Variable[x]: GPIO_PWIDTH_D - 1 |

5.1.11 GPIO_SWPORTD_DDR

- **Name:** Port D Data Direction Register
- **Description:** Port D Data Direction Register
- **Size:** 32 bits
- **Offset:** 0x28
- **Exists:** GPIO_NUM_PORTS==4

| | |
|-----------------------|------|
| RSVD_GPIO_SWPORTD_DDR | 31:y |
| GPIO_SWPORTD_DDR | x:0 |

Table 5-16 Fields for Register: GPIO_SWPORTD_DDR

| Bits | Name | Memory Access | Description |
|------|-----------------------|---------------|--|
| 31:y | RSVD_GPIO_SWPORTD_DDR | R | RSVD_GPIO_SWPORTD_DDR Reserved bits - read as zero Value After Reset: 0x0 Exists: Always Range Variable[y]: GPIO_PWIDTH_D |
| x:0 | GPIO_SWPORTD_DDR | R/W | Values written to this register independently control the direction of the corresponding data bit in Port D. The default direction can be configured as input or output after system reset through the GPIO_DFLT_DIR_D parameter. Values: <ul style="list-style-type: none"> ■ 0x0 (IN): Input Direction (default) ■ 0x1 (OUT): Output Direction Value After Reset: {(GPIO_DFLT_DIR_D==1) ? (pow (2, GPIO_PWIDTH_D))-1 : 0} Exists: Always Range Variable[x]: GPIO_PWIDTH_D - 1 |

5.1.12 GPIO_SWPORTD_CTL

- **Name:** Port D data source register
- **Description:** Port D data source register
- **Size:** 32 bits
- **Offset:** 0x2c
- **Exists:** GPIO_HW_PORTD==1 && GPIO_NUM_PORTS>3

| | |
|-----------------------|------|
| RSVD_GPIO_SWPORTD_CTL | 31:y |
| GPIO_SWPORTD_CTL | x:0 |

Table 5-17 Fields for Register: GPIO_SWPORTD_CTL

| Bits | Name | Memory Access | Description |
|------|-----------------------|---------------|---|
| 31:y | RSVD_GPIO_SWPORTD_CTL | R | RSVD_GPIO_SWPORTD_CTL Reserved bits - read as zero Value After Reset: 0x0 Exists: Always Range Variable[y]: GPIO_SWPORTD_CTL_REG_SIZE |

Table 5-17 Fields for Register: GPIO_SWPORTD_CTL (Continued)

| Bits | Name | Memory Access | Description |
|------|------------------|---------------|---|
| x:0 | GPIO_SWPORTD_CTL | R/W | <p>The data and control source for a signal can come from either software or hardware; this bit selects between them. The default source is configurable through the GPIO_DFLT_DIR_D configuration parameter.</p> <p>If GPIO_PORTD_SINGLE_CTL = 0, the register will contain one bit for each bit of the signal. Upon reset in this case, the value of GPIO_DFLT_SRC_D is replicated across all bits of the signal so that all bits power up with the same operating mode. Furthermore, the default source of each bit of the signal can subsequently be changed by writing to the corresponding bit of this register. This register is not available unless GPIO_HW_PORTD = 1.</p> <p>If GPIO_PORTD_SINGLE_CTL = 1, then the reset value is GPIO_DFLT_SRC_D. If GPIO_PORTD_SINGLE_CTL = 0, then the reset value is {GPIO_PWIDTH_D{GPIO_DFLT_SRC_D in each bit}}.</p> <p>The values of this field depends on the size. Individual bit values will indicate whether the data source is hardware or software.</p> <p>0 : Indicates Software mode 1 : Indicates Hardware mode</p> <p>Value After Reset: GPIO_DFLT_SRC_RESET_D Exists: Always Range Variable[x]: GPIO_SWPORTD_CTL_REG_SIZE - 1</p> |

5.1.13 GPIO_INTEN

- **Name:** Interrupt enable register
- **Description:** Interrupt enable register

Note: This register is available only if Port A is configured to generate interrupts (GPIO_PORTA_INTR = Include (1)).

- **Size:** 32 bits
- **Offset:** 0x30
- **Exists:** GPIO_PORTA_INTR==1



Table 5-18 Fields for Register: GPIO_INTEN

| Bits | Name | Memory Access | Description |
|------|-----------------|---------------|---|
| 31:y | RSVD_GPIO_INTEN | R | RSVD_GPIO_INTEN Reserved bits - read as zero Value After Reset: 0x0 Exists: Always Range Variable[y]: GPIO_PWIDTH_A |

Table 5-18 Fields for Register: GPIO_INTEN (Continued)

| Bits | Name | Memory Access | Description |
|------|------------|---------------|---|
| x:0 | GPIO_INTEN | R/W | <p>Allows each bit of Port A to be configured for interrupts. By default the generation of interrupts is disabled. Whenever a 1 is written to a bit of this register, it configures the corresponding bit on Port A to become an interrupt; otherwise, Port A operates as a normal GPIO signal. Interrupts are disabled on the corresponding bits of Port A if the corresponding data direction register is set to Output or if Port A mode is set to Hardware.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ 0x0 (DISABLED): Interrupt is disabled ■ 0x1 (ENABLED): Interrupt is enabled <p>Value After Reset: 0x0</p> <p>Exists: Always</p> <p>Range Variable[x]: GPIO_PWIDTH_A - 1</p> |

5.1.14 GPIO_INTMASK

- **Name:** Interrupt mask register
- **Description:** Interrupt mask register

Note: This register is available only if Port A is configured to generate interrupts (GPIO_PORTA_INTR = Include (1)).

- **Size:** 32 bits
- **Offset:** 0x34
- **Exists:** GPIO_PORTA_INTR==1

| | |
|-------------------|------|
| RSVD_GPIO_INTMASK | 31:y |
| GPIO_INTMASK | x:0 |

Table 5-19 Fields for Register: GPIO_INTMASK

| Bits | Name | Memory Access | Description |
|------|-------------------|---------------|---|
| 31:y | RSVD_GPIO_INTMASK | R | RSVD_GPIO_INTMASK Reserved bits - read as zero Value After Reset: 0x0 Exists: Always Range Variable[y]: GPIO_PWIDTH_A |

Table 5-19 Fields for Register: GPIO_INTMASK (Continued)

| Bits | Name | Memory Access | Description |
|------|--------------|---------------|--|
| x:0 | GPIO_INTMASK | R/W | <p>Controls whether an interrupt on Port A can create an interrupt for the interrupt controller by not masking it. By default, all interrupts bits are unmasked. Whenever a 1 is written to a bit in this register, it masks the interrupt generation capability for this signal; otherwise interrupts are allowed through. The unmasked status can be read as well as the resultant status after masking.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ 0x0 (DISABLED): Interrupt bits are unmasked ■ 0x1 (ENABLED): Mask interrupt <p>Value After Reset: 0x0</p> <p>Exists: Always</p> <p>Range Variable[x]: GPIO_PWIDTH_A - 1</p> |

5.1.15 GPIO_INTTYPE_LEVEL

- **Name:** Interrupt level

- **Description:** Interrupt level

Note: This register is available only if Port A is configured to generate interrupts (GPIO_PORTA_INTR = Include (1)).

- **Size:** 32 bits
- **Offset:** 0x38
- **Exists:** GPIO_PORTA_INTR==1

| | |
|-------------------------|------|
| RSVD_GPIO_INTTYPE_LEVEL | 31:y |
| GPIO_INTTYPE_LEVEL | x:0 |

Table 5-20 Fields for Register: GPIO_INTTYPE_LEVEL

| Bits | Name | Memory Access | Description |
|------|-------------------------|---------------|---|
| 31:y | RSVD_GPIO_INTTYPE_LEVEL | R | RSVD_GPIO_INTTYPE_LEVEL Reserved bits - read as zero Value After Reset: 0x0 Exists: Always Range Variable[y]: GPIO_PWIDTH_A |

Table 5-20 Fields for Register: GPIO_INTTYPE_LEVEL (Continued)

| Bits | Name | Memory Access | Description |
|------|--------------------|---------------|--|
| x:0 | GPIO_INTTYPE_LEVEL | R/W | <p>Controls the type of interrupt that can occur on Port A. Whenever a 0 is written to a bit of this register, it configures the interrupt type to be level-sensitive; otherwise, it is edge-sensitive.</p> <p>Values:</p> <ul style="list-style-type: none">■ 0x0 (LEVEL_SENSITIVE): Interrupt is level sensitive■ 0x1 (EDGE_SENSITIVE): Interrupt is edge sensitive <p>Value After Reset: 0x0</p> <p>Exists: Always</p> <p>Range Variable[x]: GPIO_PWIDTH_A - 1</p> |

5.1.16 GPIO_INT_POLARITY

- **Name:** Interrupt polarity
- **Description:** Interrupt polarity

Note: This register is available only if Port A is configured to generate interrupts (GPIO_PORTA_INTR = Include (1)).

- **Size:** 32 bits
- **Offset:** 0x3c
- **Exists:** GPIO_PORTA_INTR==1

| | |
|------------------------|-------------------|
| RSVD_GPIO_INT_POLARITY | GPIO_INT_POLARITY |
| 31:y | x:0 |

Table 5-21 Fields for Register: GPIO_INT_POLARITY

| Bits | Name | Memory Access | Description |
|------|------------------------|---------------|--|
| 31:y | RSVD_GPIO_INT_POLARITY | R | RSVD_GPIO_INT_POLARITY Reserved bits - read as zero Value After Reset: 0x0 Exists: Always Range Variable[y]: GPIO_PWIDTH_A |

Table 5-21 Fields for Register: GPIO_INT_POLARITY (Continued)

| Bits | Name | Memory Access | Description |
|------|-------------------|---------------|--|
| x:0 | GPIO_INT_POLARITY | R/W | <p>Controls the polarity of edge or level sensitivity that can occur on input of Port A. Whenever a 0 is written to a bit of this register, it configures the interrupt type to falling-edge or active-low sensitive; otherwise, it is rising-edge or active-high sensitive.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ 0x0 (ACTIVE_LOW): Active Low polarity ■ 0x1 (ACTIVE_HIGH): Active High polarity <p>Value After Reset: 0x0</p> <p>Exists: Always</p> <p>Range Variable[x]: GPIO_PWIDTH_A - 1</p> |

5.1.17 GPIO_INTSTATUS

- **Name:** Interrupt status
- **Description:** Interrupt status

Note: This register is available only if Port A is configured to generate interrupts (GPIO_PORTA_INTR = Include (1)).

- **Size:** 32 bits
- **Offset:** 0x40
- **Exists:** GPIO_PORTA_INTR==1

| | |
|---------------------|------|
| RSVD_GPIO_INTSTATUS | 31:y |
| GPIO_INTSTATUS | x:0 |

Table 5-22 Fields for Register: GPIO_INTSTATUS

| Bits | Name | Memory Access | Description |
|------|---------------------|---------------|---|
| 31:y | RSVD_GPIO_INTSTATUS | R | RSVD_GPIO_INTSTATUS Reserved bits - read as zero Value After Reset: 0x0 Exists: Always Volatile: true Range Variable[y]: GPIO_PWIDTH_A |
| x:0 | GPIO_INTSTATUS | R | Interrupt status of Port A. Values: <ul style="list-style-type: none"> ■ 0x0 (INACTIVE): Inactive ■ 0x1 (ACTIVE): Active Value After Reset: 0x0 Exists: Always Volatile: true Range Variable[x]: GPIO_PWIDTH_A - 1 |

5.1.18 GPIO_RAW_INTSTATUS

- **Name:** Raw interrupt status
- **Description:** Raw interrupt status

Note: This register is available only if Port A is configured to generate interrupts (GPIO_PORTA_INTR = Include (1)).

- **Size:** 32 bits
- **Offset:** 0x44
- **Exists:** GPIO_PORTA_INTR==1

| | |
|-----------------------|------|
| RSVD_GPIO_RAW_INTMASK | 31:y |
| GPIO_RAW_INTSTATUS | x:0 |

Table 5-23 Fields for Register: GPIO_RAW_INTSTATUS

| Bits | Name | Memory Access | Description |
|------|-----------------------|---------------|---|
| 31:y | RSVD_GPIO_RAW_INTMASK | R | RSVD_GPIO_RAW_INTMASK Reserved bits - read as zero Value After Reset: 0x0 Exists: Always Volatile: true Range Variable[y]: GPIO_PWIDTH_A |
| x:0 | GPIO_RAW_INTSTATUS | R | Raw interrupt of status of Port A (premasking bits) Values: <ul style="list-style-type: none"> ■ 0x0 (INACTIVE): Inactive ■ 0x1 (ACTIVE): Active Value After Reset: 0x0 Exists: Always Volatile: true Range Variable[x]: GPIO_PWIDTH_A - 1 |

5.1.19 GPIO_DEBOUNCE

- **Name:** Debounce enable

- **Description:** Debounce enable

Note: This register is available only if Port A is configured to generate interrupts (GPIO_PORTA_INTR = Include (1)) and when the debounce logic is included (GPIO_DEBOUNCE = Include (1)).

- **Size:** 32 bits

- **Offset:** 0x48

- **Exists:** GPIO_PORTA_INTR==1 && GPIO_DEBOUNCE==1



Table 5-24 Fields for Register: GPIO_DEBOUNCE

| Bits | Name | Memory Access | Description |
|------|--------------------|---------------|--|
| 31:y | RSVD_GPIO_DEBOUNCE | R | RSVD_GPIO_DEBOUNCE Reserved bits - read as zero Value After Reset: 0x0 Exists: Always Range Variable[y]: GPIO_PWIDTH_A |

Table 5-24 Fields for Register: GPIO_DEBOUNCE (Continued)

| Bits | Name | Memory Access | Description |
|------|---------------|---------------|--|
| x:0 | GPIO_DEBOUNCE | R/W | <p>Controls whether an external signal that is the source of an interrupt needs to be debounced to remove any spurious glitches. Writing a 1 to a bit in this register enables the debouncing circuitry. A signal must be valid for two periods of an external clock before it is internally processed.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ 0x0 (DISABLED): No debounce ■ 0x1 (ENABLED): Enable debounce <p>Value After Reset: 0x0</p> <p>Exists: Always</p> <p>Range Variable[x]: GPIO_PWIDTH_A - 1</p> |

5.1.20 GPIO_PORTA_EOI

- **Name:** Port A clear interrupt register
- **Description:** Port A clear interrupt register

Note: This register is available only if Port A is configured to generate interrupts (GPIO_PORTA_INTR = Include (1)) and when the debounce logic is included (GPIO_DEBOUNCE = Include (1)).

- **Size:** 32 bits
- **Offset:** 0x4c
- **Exists:** GPIO_PORTA_INTR==1

| | |
|---------------------|------|
| RSVD_GPIO_PORTA_EOI | 31:y |
| GPIO_PORTA_EOI | x:0 |

Table 5-25 Fields for Register: GPIO_PORTA_EOI

| Bits | Name | Memory Access | Description |
|------|---------------------|---------------|---|
| 31:y | RSVD_GPIO_PORTA_EOI | W | RSVD_GPIO_PORTA_EOI Reserved bits - read as zero Value After Reset: 0x0 Exists: Always Range Variable[y]: GPIO_PWIDTH_A |

Table 5-25 Fields for Register: GPIO_PORTA_EOI (Continued)

| Bits | Name | Memory Access | Description |
|------|----------------|---------------|---|
| x:0 | GPIO_PORTA_EOI | W | <p>Controls the clearing of edge type interrupts from Port A. When a 1 is written into a corresponding bit of this register, the interrupt is cleared. All interrupts are cleared when Port A is not configured for interrupts.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ 0x0 (DISABLED): No interrupt clear ■ 0x1 (ENABLED): Clear Interrupt <p>Value After Reset: 0x0</p> <p>Exists: Always</p> <p>Range Variable[x]: GPIO_PWIDTH_A - 1</p> |

5.1.21 GPIO_EXT_PORTA

- **Name:** External port A register
- **Description:** External port A register
- **Size:** 32 bits
- **Offset:** 0x50
- **Exists:** Always

| | |
|---------------------|------|
| RSVD_GPIO_EXT_PORTA | 31:y |
| GPIO_EXT_PORTA | x:0 |

Table 5-26 Fields for Register: GPIO_EXT_PORTA

| Bits | Name | Memory Access | Description |
|------|---------------------|---------------|---|
| 31:y | RSVD_GPIO_EXT_PORTA | R | RSVD_GPIO_EXT_PORTA Reserved bits - read as zero Value After Reset: 0x0 Exists: Always Volatile: true Range Variable[y]: GPIO_PWIDTH_A |
| x:0 | GPIO_EXT_PORTA | R | This register always reflects the signals value on the External Port A. Value After Reset: 0x0 Exists: Always Volatile: true Range Variable[x]: GPIO_PWIDTH_A - 1 |

5.1.22 GPIO_EXT_PORTB

- **Name:** Port B external port register
- **Description:** Port B external port register
- **Size:** 32 bits
- **Offset:** 0x54
- **Exists:** GPIO_NUM_PORTS>1



Table 5-27 Fields for Register: GPIO_EXT_PORTB

| Bits | Name | Memory Access | Description |
|------|---------------------|---------------|---|
| 31:y | RSVD_GPIO_EXT_PORTB | R | RSVD_GPIO_EXT_PORTB Reserved bits - read as zero Value After Reset: 0x0 Exists: Always Volatile: true Range Variable[y]: GPIO_PWIDTH_B |
| x:0 | GPIO_EXT_PORTB | R | This register always reflects the signals value on the External Port B. Value After Reset: 0x0 Exists: Always Volatile: true Range Variable[x]: GPIO_PWIDTH_B - 1 |

5.1.23 GPIO_EXT_PORTC

- **Name:** External port C register
- **Description:** External port C register
- **Size:** 32 bits
- **Offset:** 0x58
- **Exists:** GPIO_NUM_PORTS>2

| | |
|---------------------|------|
| RSVD_GPIO_EXT_PORTC | 31:y |
| GPIO_EXT_PORTC | x:0 |

Table 5-28 Fields for Register: GPIO_EXT_PORTC

| Bits | Name | Memory Access | Description |
|------|---------------------|---------------|--|
| 31:y | RSVD_GPIO_EXT_PORTC | R | RSVD_GPIO_EXT_PORTC Reserved bits - read as zero Value After Reset: 0x0 Exists: Always Volatile: true Range Variable[y]: GPIO_PWIDTH_C |
| x:0 | GPIO_EXT_PORTC | R | This register always reflects the signals value on the External Port C. Reset Value: 0x0 Value After Reset: 0x0 Exists: Always Volatile: true Range Variable[x]: GPIO_PWIDTH_C - 1 |

5.1.24 GPIO_EXT_PORTD

- **Name:** Port D external port register
- **Description:** Port D external port register
- **Size:** 32 bits
- **Offset:** 0x5c
- **Exists:** GPIO_NUM_PORTS==4

| | |
|---------------------|------|
| RSVD_GPIO_EXT_PORTD | 31:y |
| GPIO_EXT_PORTD | x:0 |

Table 5-29 Fields for Register: GPIO_EXT_PORTD

| Bits | Name | Memory Access | Description |
|------|---------------------|---------------|--|
| 31:y | RSVD_GPIO_EXT_PORTD | R | RSVD_GPIO_EXT_PORTD Reserved bits - read as zero Value After Reset: 0x0 Exists: Always Volatile: true Range Variable[y]: GPIO_PWIDTH_D |
| x:0 | GPIO_EXT_PORTD | R | This register always reflects the signals value on the External Port D. Reset Value: 0x0 Value After Reset: 0x0 Exists: Always Volatile: true Range Variable[x]: GPIO_PWIDTH_D - 1 |

5.1.25 GPIO_LS_SYNC

- **Name:** Synchronization level
- **Description:** Synchronization level
- **Size:** 32 bits
- **Offset:** 0x60
- **Exists:** Always

| | |
|-------------------|------|
| RSVD_GPIO_LS_SYNC | 31:1 |
| GPIO_LS_SYNC | 0 |

Table 5-30 Fields for Register: GPIO_LS_SYNC

| Bits | Name | Memory Access | Description |
|------|-------------------|---------------|---|
| 31:1 | RSVD_GPIO_LS_SYNC | R | RSVD_GPIO_LS_SYNC Reserved bits - read as zero Value After Reset: 0x0 Exists: Always |
| 0 | GPIO_LS_SYNC | * Varies | Writing a 1 to this register results in all level-sensitive interrupts being synchronized to pclk_intr. Values: <ul style="list-style-type: none"> ■ 0x0 (DISABLED): No synchronization to pclk_int (default) ■ 0x1 (ENABLED): Synchronize to pclk_intr Value After Reset: 0x0 Exists: Always Memory Access: "(GPIO_PORTA_INTR==1) ? \"read-write\" : \"read-only\"" |

5.1.26 GPIO_ID_CODE

- **Name:** GPIO ID code
- **Description:** GPIO ID code
- **Size:** 32 bits
- **Offset:** 0x64
- **Exists:** Always

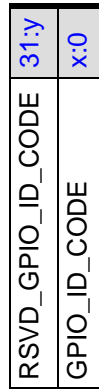


Table 5-31 Fields for Register: GPIO_ID_CODE

| Bits | Name | Memory Access | Description |
|------|-------------------|---------------|---|
| 31:y | RSVD_GPIO_ID_CODE | R | RSVD_GPIO_ID_CODE Reserved bits - read as zero Value After Reset: 0x0 Exists: Always Range Variable[y]: GPIO_ID_WIDTH |
| x:0 | GPIO_ID_CODE | R | This is a user-specified code that a system can read. It can be used for chip identification, and so on. Value After Reset: GPIO_ID_NUM Exists: Always Range Variable[x]: GPIO_ID_WIDTH - 1 |

5.1.27 GPIO_INT_BOTHEDGE

- **Name:** Interrupt Both Edge type
- **Description:** Interrupt Both Edge type

Note: This register is available only if PORT A is configured to generate interrupts (GPIO_PORTA_INTR = Include(1)) and interrupt detection is configured to generate on both rising and falling edges of external input signal (GPIO_INT_BOTH_EDGE = Include(1)).

- **Size:** 32 bits
- **Offset:** 0x68
- **Exists:** GPIO_INT_BOTH_EDGE==1

| | |
|------------------------|------|
| RSVD_GPIO_INT_BOTHEDGE | 31:y |
| GPIO_INT_BOTHEDGE | x:0 |

Table 5-32 Fields for Register: GPIO_INT_BOTHEDGE

| Bits | Name | Memory Access | Description |
|------|------------------------|---------------|--|
| 31:y | RSVD_GPIO_INT_BOTHEDGE | R | RSVD_GPIO_INT_BOTHEDGE Reserved bits - read as zero Value After Reset: 0x0 Exists: Always Range Variable[y]: GPIO_PWIDTH_A |

Table 5-32 Fields for Register: GPIO_INT_BOTHEGE (Continued)

| Bits | Name | Memory Access | Description |
|------|------------------|---------------|--|
| x:0 | GPIO_INT_BOTHEGE | R/W | <p>Controls the edge type of interrupt that can occur on Port A.</p> <ul style="list-style-type: none"> - Whenever a particular bit is programmed to 1, it enables the generation of interrupts on both the rising edge and the falling edge of an external input signal corresponding to that bit on port A. - The values programmed in the registers gpio_inttype_level and gpio_int_polarity for this particular bit are not considered when the corresponding bit of this register is set to 1. - Whenever a particular bit is programmed to 0, the interrupt type depends on the value of the corresponding bits in the gpio_inttype_level and gpio_int_polarity registers. <p>Values:</p> <ul style="list-style-type: none"> ■ 0x0 (DISABLED): single edge sensitive ■ 0x1 (ENABLED): both edge sensitive <p>Value After Reset: 0x0</p> <p>Exists: Always</p> <p>Range Variable[x]: GPIO_PWIDTH_A - 1</p> |

5.1.28 GPIO_VER_ID_CODE

- **Name:** GPIO Component Version
- **Description:** GPIO Component Version
- **Size:** 32 bits
- **Offset:** 0x6c
- **Exists:** Always

GPIO_VER_ID_CODE 31:0

Table 5-33 Fields for Register: GPIO_VER_ID_CODE

| Bits | Name | Memory Access | Description |
|------|------------------|---------------|---|
| 31:0 | GPIO_VER_ID_CODE | R | ASCII value for each number in the version, followed by *. For example 32_31_32_2A represents the version 2.12*. Value After Reset: GPIO_VERSION_ID Exists: Always |

5.1.29 GPIO_CONFIG_REG2

- **Name:** GPIO Configuration Register 2
- **Description:** GPIO Configuration Register 2

This register is a read-only register that is present when the configuration parameter `GPIO_ADD_ENCODED_PARAMS` is set to True. If this configuration is set to False, then this register reads back 0.

- **Size:** 32 bits
- **Offset:** 0x70
- **Exists:** Always

| | |
|-----------------------|-------|
| RSVD_GPIO_CONFIG_REG2 | 31:20 |
| ENCODED_ID_PWIDTH_D | 19:15 |
| ENCODED_ID_PWIDTH_C | 14:10 |
| ENCODED_ID_PWIDTH_B | 9:5 |
| ENCODED_ID_PWIDTH_A | 4:0 |

Table 5-34 Fields for Register: GPIO_CONFIG_REG2

| Bits | Name | Memory Access | Description |
|-------|-----------------------|---------------|--|
| 31:20 | RSVD_GPIO_CONFIG_REG2 | R | RSVD_GPIO_CONFIG_REG2 Reserved bits - read as zero Value After Reset: 0x0 Exists: Always |
| 19:15 | ENCODED_ID_PWIDTH_D | R | The value of this register is equal to <code>GPIO_PWIDTH_D-1</code> . Value After Reset: {(GPIO_ADD_ENCODED_PARAMS==1) ? GPIO_ENCODED_PWIDTH_D : 0} Exists: Always |

Table 5-34 Fields for Register: GPIO_CONFIG_REG2 (Continued)

| Bits | Name | Memory Access | Description |
|-------|---------------------|---------------|--|
| 14:10 | ENCODED_ID_PWIDTH_C | R | The value of this register is equal to GPIO_PWIDTH_C-1. Value After Reset: {{GPIO_ADD_ENCODED_PARAMS==1} ? GPIO_ENCODED_PWIDTH_C : 0} Exists: Always |
| 9:5 | ENCODED_ID_PWIDTH_B | R | The value of this register is equal to GPIO_PWIDTH_B-1. Value After Reset: {{GPIO_ADD_ENCODED_PARAMS==1} ? GPIO_ENCODED_PWIDTH_B : 0} Exists: Always |
| 4:0 | ENCODED_ID_PWIDTH_A | R | The value of this register is equal to GPIO_PWIDTH_A-1. Value After Reset: {{GPIO_ADD_ENCODED_PARAMS==1} ? GPIO_ENCODED_PWIDTH_A : 0} Exists: Always |

5.1.30 GPIO_CONFIG_REG1

- **Name:** GPIO Configuration Register 1
- **Description:** GPIO Configuration Register 1

This register is present when the configuration parameter GPIO_ADD_ENCODED_PARAMS is set to True. If this parameter is set to False, this register reads back zero (0).

- **Size:** 32 bits
- **Offset:** 0x74
- **Exists:** Always

| | |
|--------------------------|-------|
| RSVD_GPIO_CONFIG_REG1 | 31:22 |
| INTERRUPT_BOTH_EDGE_TYPE | 21 |
| ENCODED_ID_WIDTH | 20:16 |
| GPIO_ID | 15 |
| ADD_ENCODED_PARAMS | 14 |
| DEBOUNCE | 13 |
| PORTA_INTR | 12 |
| HW_PORTD | 11 |
| HW_PORTC | 10 |
| HW_PORTB | 9 |
| HW_PORTA | 8 |
| PORTD_SINGLE_CTL | 7 |
| PORTC_SINGLE_CTL | 6 |
| PORTB_SINGLE_CTL | 5 |
| PORTA_SINGLE_CTL | 4 |
| NUM_PORTS | 3:2 |
| APB_DATA_WIDTH | 1:0 |

Table 5-35 Fields for Register: GPIO_CONFIG_REG1

| Bits | Name | Memory Access | Description |
|-------|-----------------------|---------------|--|
| 31:22 | RSVD_GPIO_CONFIG_REG1 | R | RSVD_GPIO_CONFIG_REG1 Reserved bits - read as zero Value After Reset: 0x0 Exists: Always |

Table 5-35 Fields for Register: GPIO_CONFIG_REG1 (Continued)

| Bits | Name | Memory Access | Description |
|-------|--------------------------|---------------|--|
| 21 | INTERRUPT_BOTH_EDGE_TYPE | R | <p>The value of this register is derived from the GPIO_INT_BOTH_EDGE configuration parameter</p> <p>Values:</p> <ul style="list-style-type: none"> ■ 0x0 (DISABLED): Interrupt generation on rising or falling edge ■ 0x1 (ENABLED): Interrupt generation on both rising and falling edge <p>Value After Reset: {(GPIO_ADD_ENCODED_PARAMS==1) ? PORTA_INT_BOTHEDGE : 0}</p> <p>Exists: Always</p> |
| 20:16 | ENCODED_ID_WIDTH | R | <p>The value of this register is derived from the GPIO_ID_WIDTH configuration parameter.</p> <p>Value After Reset: {(GPIO_ADD_ENCODED_PARAMS==1) ? GPIO_ENCODED_ID_WIDTH : 0}</p> <p>Exists: Always</p> |
| 15 | GPIO_ID | R | <p>The value of this register is derived from the GPIO_ID configuration parameter.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ 0x0 (DISABLED): GPIO_ID not included ■ 0x1 (ENABLED): GPIO_ID is included <p>Value After Reset: {(GPIO_ADD_ENCODED_PARAMS==1) ? GPIO_L_ID : 0}</p> <p>Exists: Always</p> |
| 14 | ADD_ENCODED_PARAMS | R | <p>The value of this register is derived from the GPIO_ADD_ENCODED_PARAMS configuration parameter.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ 0x0 (DISABLED): Encoded parameters not added ■ 0x1 (ENABLED): Encoded parameters added <p>Value After Reset: GPIO_ADD_ENCODED_PARAMS</p> <p>Exists: Always</p> |

Table 5-35 Fields for Register: GPIO_CONFIG_REG1 (Continued)

| Bits | Name | Memory Access | Description |
|------|------------|---------------|---|
| 13 | DEBOUNCE | R | <p>The value of this register is derived from the GPIO_DEBOUNCE configuration parameter.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ 0x0 (DISABLED): Exclude debounce capability ■ 0x1 (ENABLED): Include debounce capability <p>Value After Reset: {{GPIO_ADD_ENCODED_PARAMS==1) ? GPIO_L_DEBOUNCE : 0}</p> <p>Exists: Always</p> |
| 12 | PORTA_INTR | R | <p>The value of this register is derived from the GPIO_PORTA_INTR configuration parameter.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ 0x0 (DISABLED): PORT A is not used as an interrupt source ■ 0x1 (ENABLED): PORT A is required to be used as an interrupt source <p>Value After Reset: {{GPIO_ADD_ENCODED_PARAMS==1) ? GPIO_L_PORTA_INTR : 0}</p> <p>Exists: Always</p> |
| 11 | HW_PORTD | R | <p>The value of this register is derived from the GPIO_HW_PORTD configuration parameter.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ 0x0 (DISABLED): Port D has external, auxiliary hardware signals excluded ■ 0x1 (ENABLED): Port D has external, auxiliary hardware signals included <p>Value After Reset: {{GPIO_ADD_ENCODED_PARAMS==1) ? GPIO_L_HW_PORTD : 0}</p> <p>Exists: Always</p> |

Table 5-35 Fields for Register: GPIO_CONFIG_REG1 (Continued)

| Bits | Name | Memory Access | Description |
|------|----------|---------------|---|
| 10 | HW_PORTC | R | <p>The value of this register is derived from the GPIO_HW_PORTC configuration parameter.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ 0x0 (DISABLED): Port C has external, auxiliary hardware signals excluded ■ 0x1 (ENABLED): Port C has external, auxiliary hardware signals included <p>Value After Reset: {{GPIO_ADD_ENCODED_PARAMS==1} ? GPIO_L_HW_PORTC : 0}</p> <p>Exists: Always</p> |
| 9 | HW_PORTB | R | <p>The value of this register is derived from the GPIO_HW_PORTB configuration parameter.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ 0x0 (DISABLED): Port B has external, auxiliary hardware signals excluded ■ 0x1 (ENABLED): Port B has external, auxiliary hardware signals included <p>Value After Reset: {{GPIO_ADD_ENCODED_PARAMS==1} ? GPIO_L_HW_PORTB : 0}</p> <p>Exists: Always</p> |
| 8 | HW_PORTA | R | <p>The value of this register is derived from the GPIO_HW_PORTA configuration parameter.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ 0x0 (DISABLED): Port A has external, auxiliary hardware signals excluded ■ 0x1 (ENABLED): Port A has external, auxiliary hardware signals included <p>Value After Reset: {{GPIO_ADD_ENCODED_PARAMS==1} ? GPIO_L_HW_PORTA : 0}</p> <p>Exists: Always</p> |

Table 5-35 Fields for Register: GPIO_CONFIG_REG1 (Continued)

| Bits | Name | Memory Access | Description |
|------|------------------|---------------|--|
| 7 | PORTD_SINGLE_CTL | R | <p>The value of this register is derived from the GPIO_PORTD_SINGLE_CTL configuration parameter.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ 0x0 (DISABLED): PORTD is not controlled from a single source ■ 0x1 (ENABLED): PORTD is controlled from a single source <p>Value After Reset: {{GPIO_ADD_ENCODED_PARAMS==1} ? PORTD_SINGLE_CTL : 0}</p> <p>Exists: Always</p> |
| 6 | PORTC_SINGLE_CTL | R | <p>The value of this register is derived from the GPIO_PORTC_SINGLE_CTL configuration parameter.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ 0x0 (DISABLED): PORTC is not controlled from a single source ■ 0x1 (ENABLED): PORTC is controlled from a single source <p>Value After Reset: {{GPIO_ADD_ENCODED_PARAMS==1} ? PORTC_SINGLE_CTL : 0}</p> <p>Exists: Always</p> |
| 5 | PORTB_SINGLE_CTL | R | <p>The value of this register is derived from the GPIO_PORTB_SINGLE_CTL configuration parameter.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ 0x0 (DISABLED): PORTB is not controlled from a single source ■ 0x1 (ENABLED): PORTB is controlled from a single source <p>Value After Reset: {{GPIO_ADD_ENCODED_PARAMS==1} ? PORTB_SINGLE_CTL : 0}</p> <p>Exists: Always</p> |

Table 5-35 Fields for Register: GPIO_CONFIG_REG1 (Continued)

| Bits | Name | Memory Access | Description |
|------|------------------|---------------|---|
| 4 | PORTA_SINGLE_CTL | R | <p>The value of this register is derived from the GPIO_PORTA_SINGLE_CTL configuration parameter.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ 0x0 (DISABLED): PORTA is not controlled from a single source ■ 0x1 (ENABLED): PORTA is controlled from a single source <p>Value After Reset: {{GPIO_ADD_ENCODED_PARAMS==1) ? PORTA_SINGLE_CTL : 0}</p> <p>Exists: Always</p> |
| 3:2 | NUM_PORTS | R | <p>The value of this register is derived from the GPIO_NUM_PORT configuration parameter.</p> <p>Values:</p> <ul style="list-style-type: none"> ■ 0x0 (NUM_PORTS_1): Number of ports is 1 ■ 0x1 (NUM_PORTS_2): Number of ports is 2 ■ 0x2 (NUM_PORTS_3): Number of ports is 3 ■ 0x3 (NUM_PORTS_4): Number of ports is 4 <p>Value After Reset: {{GPIO_ADD_ENCODED_PARAMS==1) ? GPIO_ENCODED_NUM_PORTS : 0}</p> <p>Exists: Always</p> |
| 1:0 | APB_DATA_WIDTH | R | <p>The value of this register is derived from the GPIO_APB_DATA_WIDTH configuration parameter.</p> <p>Note: 0x3 = Reserved</p> <p>Values:</p> <ul style="list-style-type: none"> ■ 0x0 (APB_8BITS): APB DATA WIDTH is 8 bits ■ 0x1 (APB_16BITS): APB DATA WIDTH is 16 bits ■ 0x2 (APB_32BITS): APB DATA WIDTH is 32 bits <p>Value After Reset: {{GPIO_ADD_ENCODED_PARAMS==1) ? GPIO_ENCODED_APB_WIDTH : 0}</p> <p>Exists: Always</p> |

6

Programming the DW_apb_gpio

The DW_apb_gpio can be programmed through software registers or the DW_apb_gpio low-level software driver.

6.1 Software Registers

The software registers are described in more detail in [“Register Descriptions”](#) on page 65.

6.2 Software Drivers

The DesignWare Synthesizable Components for AMBA 2 includes a Driver Kit for the DW_apb_gpio component. This low-level Driver Kit allows you to easily program a DW_apb_gpio 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_gpio minimizing usage errors
- Rapid software development with minimum overhead
- Detailed knowledge of DW_apb_gpio register bit fields not required
- Easy integration of DW_apb_gpio 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_gpio Driver Kit. However, you can access some Driver Kit files and documentation in \$DESIGNWARE_HOME/drivers/DW_apb_gpio/latest. For more information about the Driver Kit, see the [DW_apb_gpio 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.

6.3 Programming Considerations

- Reading from an unused location or unused bits in a particular register always returns zeros. There is no error mechanism in the APB.
- Programming the DW_apb_gpio registers for interrupt capability, edge-sensitive or level-sensitive interrupts, and interrupt polarity should be completed prior to enabling the interrupts on Port A in order to prevent spurious glitches on the interrupt lines to the interrupt controller.

- Writing to the interrupt clear register clears an edge-detected interrupt and has no effect on a level-sensitive interrupt. If, for example, the GPIO_PORTA_INTR configuration parameter is equal to 0, then the following all read back 0:
 - gpio_inten
 - gpio_intmask
 - gpio_int_level
 - gpio_int_polarity
 - gpio_int_status
 - gpio_raw_intstatus
 - gpio_debounce
 - gpio_ls_sync
 - gpio_porta_eoi
- When reading back registers that are no longer present due to configuration parameters settings, then 0 is read back. For example, if APB_DATA_WITDH = 32 bits and GPIO_PWIDTH_A = 8, then the top 24 bits read back 0.

7

Verification

This chapter provides an overview of the testbench available for DW_apb_gpio verification. Once you have configured the DW_apb_gpio in coreConsultant and have set up the verification environment, you can automatically run simulations.

**Note**

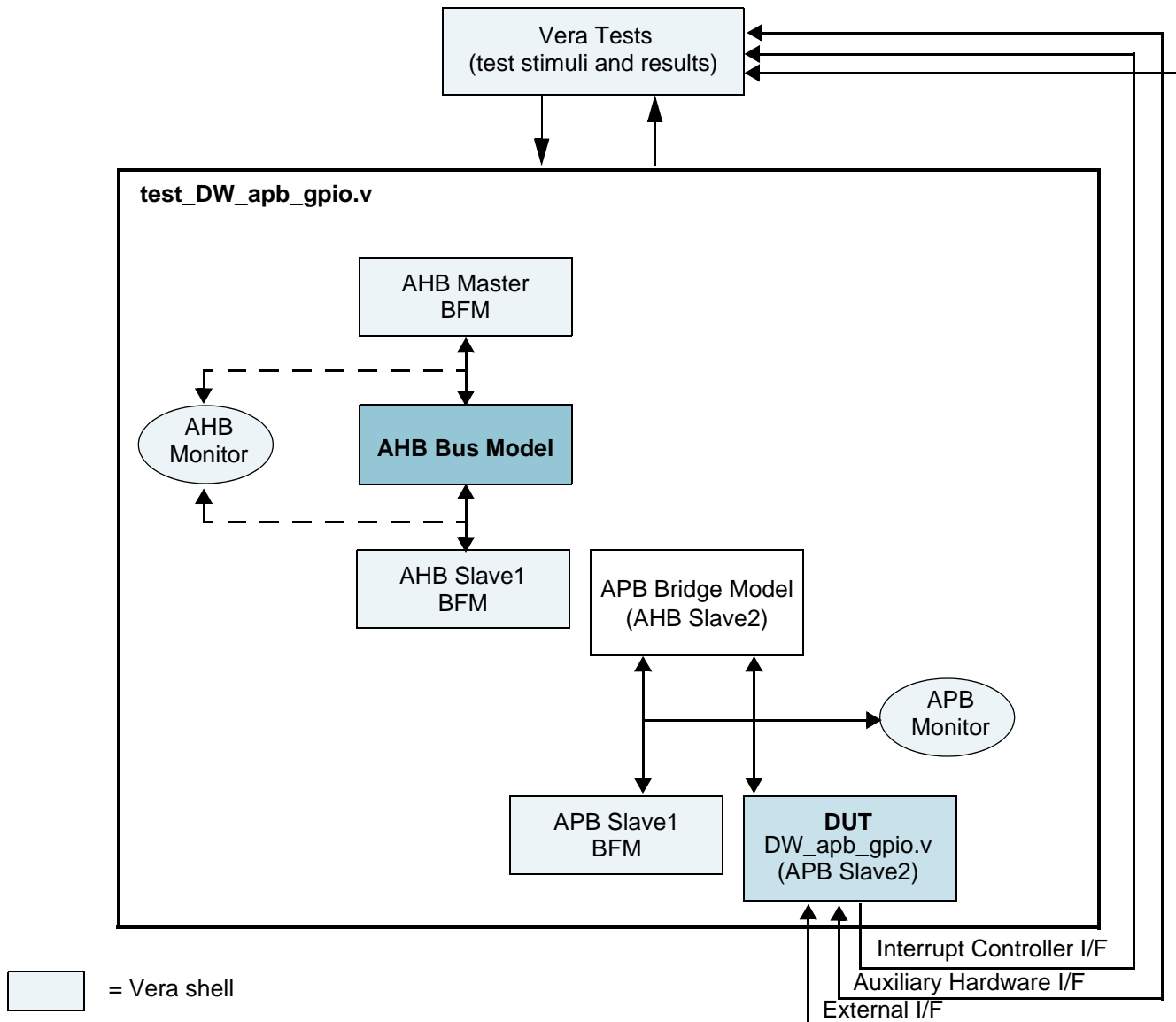
The DW_apb_gpio verification testbench is built with DesignWare Verification IP (VIP). Make sure you have the supported version of the VIP components for this release, otherwise, you may experience some tool compatibility problems. For more information about supported tools in this release, see the following web page:

www.synopsys.com/products/designware/docs/doc/amba/latest/dw_amba_install.pdf

7.1 Overview of DW_apb_gpio Testbench

As illustrated in Figure 7-1, the DW_apb_gpio verification environment includes an instantiation of the design under test (DUT), AHB and APB bridge bus models, and a Vera shell.

Figure 7-1 DW_apb_gpio Testbench



The Vera shell consists of an AHB master bus functional model (BFM), two AHB slave BFMs, an AHB monitor, APB slave BFMs, an APB monitor, test stimuli, BFM configuration, and test results. The AHB monitor tracks activity from the AHB master and slave BFMs; the APB monitor oversees activity from the APB slave BFMs.

The testbench checks for all possible user configurations selected in the Specify Configuration task of coreConsultant. The testbench also determines that the component is AMBA-compliant and includes a self-checking mechanism.

7.2 Overview of Vera Tests

The DW_apb_gpio verification testbench performs the following set of tests.

7.2.1 Resets

This test ensures that after power-on reset (POR), the memory-mapped registers are reset to their intended value. The reset value of some registers is hardwired, whereas the reset value of other registers is a configurable coreConsultant parameter. This test is transparent to the coreConsultant parameter value.

This test asynchronously asserts POR and synchronously de-asserts to the bus clock, pclk. It reads back each register in the memory map. The test reads back each byte of the memory-mapped register and may require multiple APB reads, depending on the APB_BUS_WIDTH parameter value.



Note

For certain coreConsultant parameter values, the reset value of some registers cannot be read back. When the default value of the gpio_portx_ddr (set by GPIO_DFLT_DIR_X) is set to output, then the reset value of gpio_ext_portx is read back on the APB bus. If the default value of the gpio_portx_ddr (set by GPIO_DFLT_DIR_X) is set to input, then the value on the gpio_ext_portx port is read back and not the reset value.

7.2.2 APB Slave Interface

This set of tests verifies the correct operation of the APB Slave Interface and memory-mapped registers, all of which are written to and read from the APB interface. The tests verify the correct operation for APB_DATA_WIDTH of 8-, 16- and 32-bit widths.

7.2.3 Interrupts

This set of tests verifies the correct functionality of the logic pertaining to interrupt generation, including the debounce logic. The functionality of the following are verified:

- gpio_intstatus
- gpio_raw_intstatus
- gpio_inten
- gpio_intmask
- gpio_int_polarity
- gpio_inttype_level
- gpio_debounce registers
- gpio_ls_sync registers
- gpio_intr output signal
- gpio_intr_flag output signal

7.2.4 External Data Interface

This set of tests verifies the correct operation of the logic pertaining to the external data interface. The tests are repeated for each of the four ports.

7.2.5 Auxiliary Data Interface

This set of tests verifies the correct operation of the logic pertaining to the auxiliary hardware. The tests are repeated for each of the four ports.

7.3 Running Simulations from the Command Line

To run simulations from a UNIX command line, a simulation model must be generated through the coreConsultant GUI. In addition, all tests and test options must be configured in the Verification tab of the GUI. Then, simulations can be run as follows:

- To run all tests selected in the GUI, change your working directory to `DW_apb_gpio/sim` and then execute the following command:

```
runtest.sh
```

- To run single tests, change the working directory to `DW_apb_gpio/sim` and run the following:

```
runtest --simulator selected_simulator --test test_name
```

The *selected_simulator* is the one chosen in the GUI; it does not work if it is not configured in the GUI. The *test_name* is the name of the selected test and the subdirectory where the test is located. For example, to run the simple register write/read test using VCS, run the following:

```
runtest --simulator vcs --test test_reg_wr_rd
```

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

7.4 Command Line Output Files

The `runtest.log` file is generated in `workspace/sim` only as a result of running simulations from coreConsultant or coreAssembler. The `runtest.log` file provides a pass/fail result for the particular simulation, as well as some detailed information. The `test.log` file located in `workspace/sim/test_name` is generated when tests are run from the GUI or command line, and provides more detail on each specific test simulation, in addition to the pass/fail status. The waveforms are also written to this directory, when enabled. To enable waveform generation from the command line, the switch `DumpEnabled` must be set as follows:

```
runtest --simulator vcs --DumpEnabled 1 --test test_reg_wr_rd
```

If the simulation results match expected results, the simulation successfully completes 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`.

8

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.

8.1 Reading and Writing from an APB Slave

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

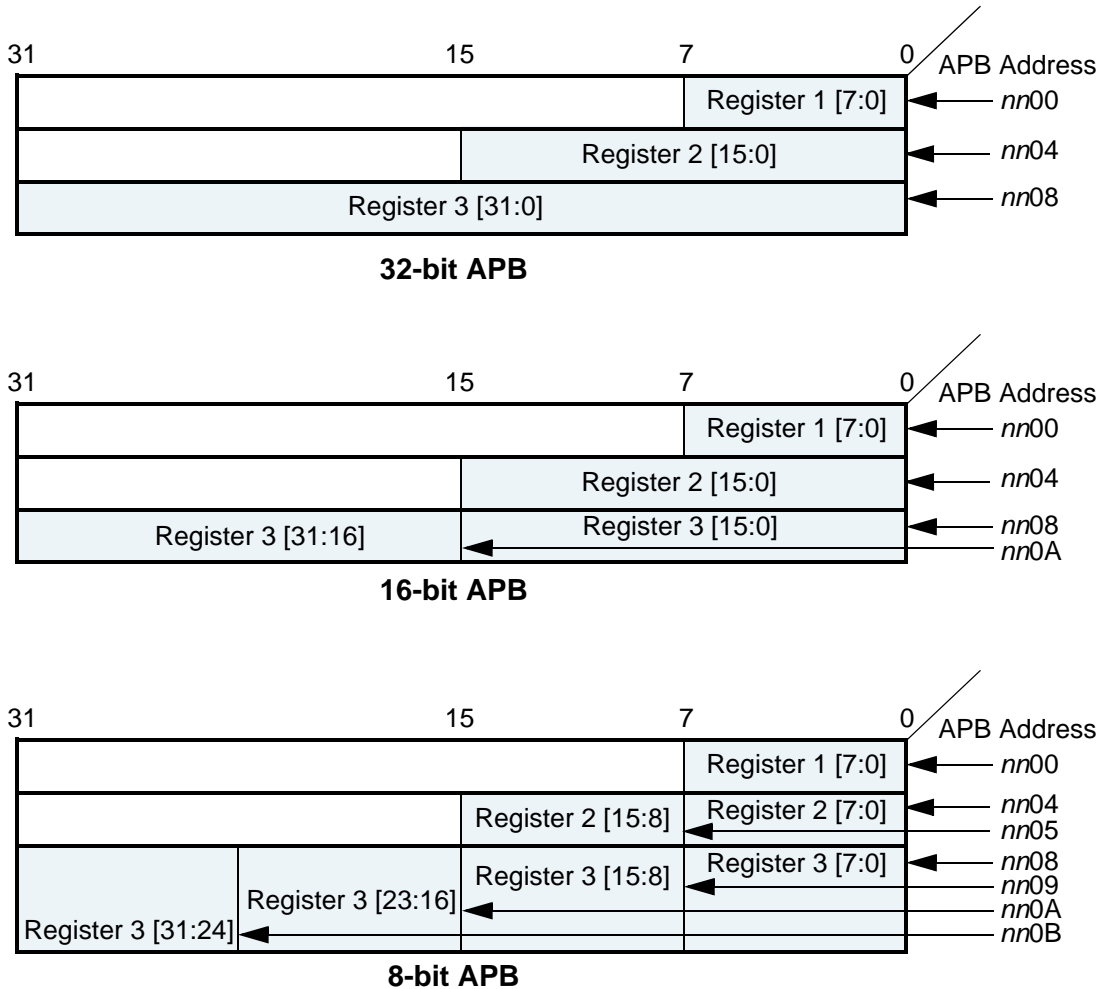
- The size of the APB peripheral should always be set equal to the size of the APB data bus, if possible.
- The APB bus has no concept of a transfer size or a byte lane, unlike the DW_ahb.
- The APB slave subsystem is little endian; the DW_apb performs the conversion from a big-endian AHB to the little-endian APB.
- All APB slave programming registers are aligned on 32-bit boundaries, irrespective of the APB bus size.
- The maximum APB_DATA_WIDTH is 32 bits. Registers larger than this occupies more than one location in the memory map.
- The DW_apb does not return any ERROR, SPLIT, or RETRY responses; it always returns an OKAY response to the AHB.
- For all bus widths:
 - In the case of a read transaction, registers less than the full bus width returns zeros in the unused upper bits.
 - Writing to bit locations larger than the register width does not have any effect. 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.

8.1.1 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 an APB slave and, therefore, in the DW_apb.

The following sections 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 8-1 Read/Write Locations for Different APB Bus Data Widths



8.1.2 32-bit Bus System

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

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.



Note

If you write to an address location not on a 32-bit boundary, the bottom bits are ignored/not used.

8.1.3 16-bit Bus System

For 16-bit bus systems, two scenarios exist, as illustrated in the previous picture:

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

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

2. The register to be written to or read from is >16 and ≤ 32 bits

In this case, two AHB transactions are required, which in turn creates two APB transactions, 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.

**Note**

If you write to an address location not on a 16-bit boundary, the bottom bits are ignored/not used.

8.1.4 8-bit Bus System

For 8-bit bus systems, three scenarios exist, as illustrated in the previous picture:

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

In this case, the register can be read or written with one transaction. In the case of a read transaction, registers less than 8 bits wide returns zeros in the unused bits. Writing to bit locations larger than the register width causes nothing to happen, that is, only the pertinent bits are written to the register.

2. The register to be written to or read from is >8 and ≤ 16 bits

In this case, two AHB transactions are required, which in turn creates two APB transactions, 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 or read from is >16 and ≤ 32 bits

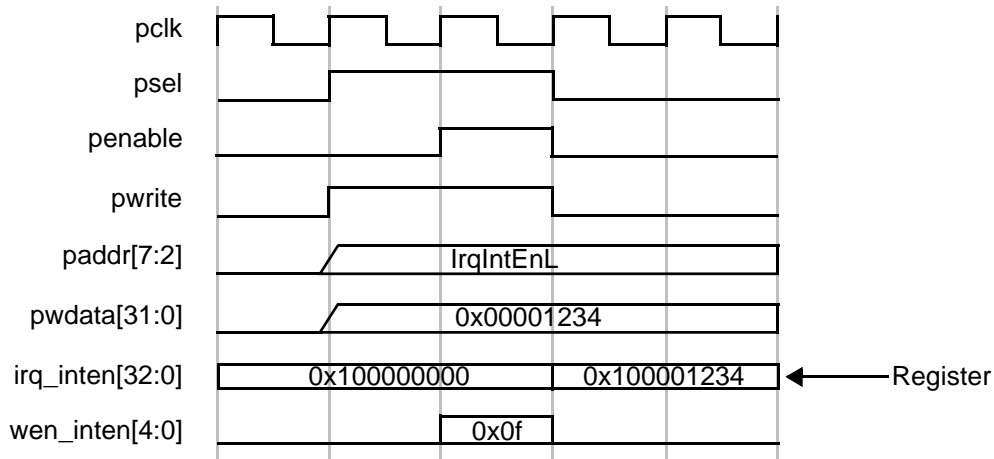
In this case, four AHB transactions are required, which in turn creates four APB transactions, to read or write the register. The first transaction should read/write the lower byte and the second transaction the second byte, and so on.

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

8.2 Write Timing Operation

A timing diagram of an APB write transaction for an APB peripheral register (an earlier version of the DW_apb_ictl) is shown in the following figure. Data, address, and control signals are aligned. The APB frame lasts for two cycles when psel is high.

Figure 8-2 APB Write Transaction



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 APB 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.

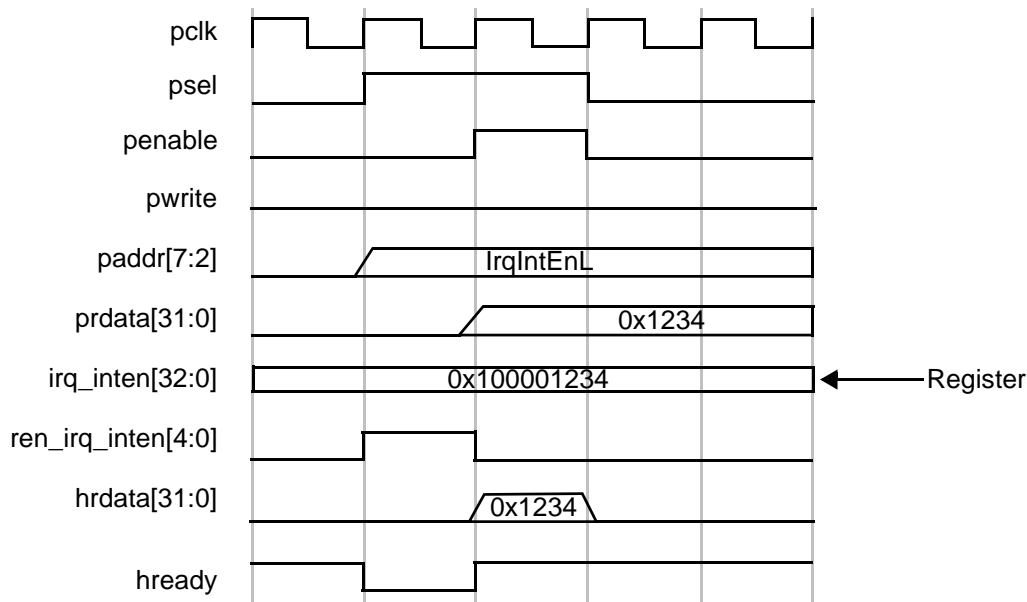
A write from the AHB to the APB does not require the AHB system bus to stall until the transfer on the APB has completed. A write to the APB can be followed by a read transaction from another AHB peripheral (not the DW_apb).

The timing example 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.

8.3 Read Timing Operation

A timing diagram of an APB read transaction for an APB peripheral (an earlier version of the DW_apb_ictl) is shown in the following figure. The APB frame lasts for two cycles, when psel is high.

Figure 8-3 APB Read Transaction



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

8.4 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>
```

8.5 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.

8.5.1 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 8-1 Upper Byte Generation

| Load Register Width | Upper Byte Bus 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 |

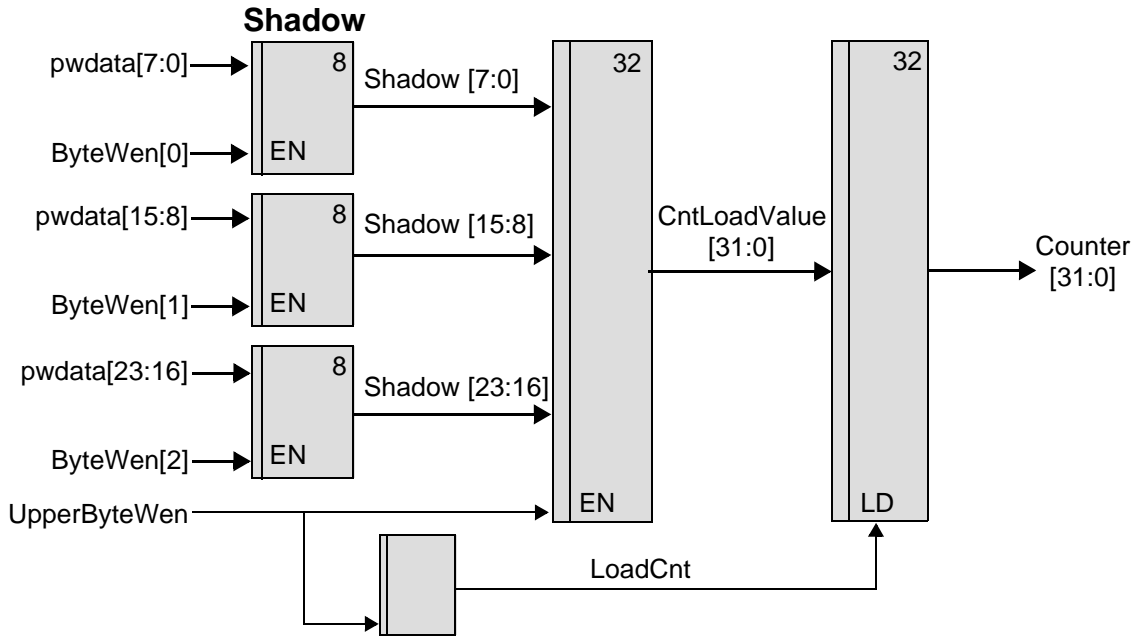
There are three relationship cases to be considered for the processor and peripheral clocks:

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

8.5.1.1 Identical Clocks

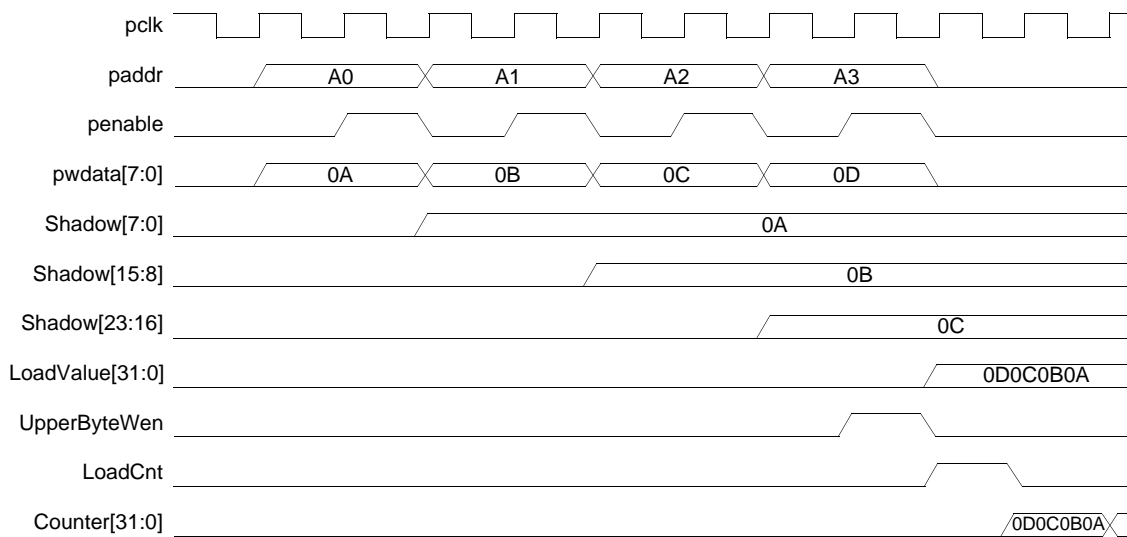
The following figure illustrates an RTL diagram for the circuitry required to implement the coherent write transaction when the APB bus clock and peripheral clocks are identical.

Figure 8-4 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 8-5 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.

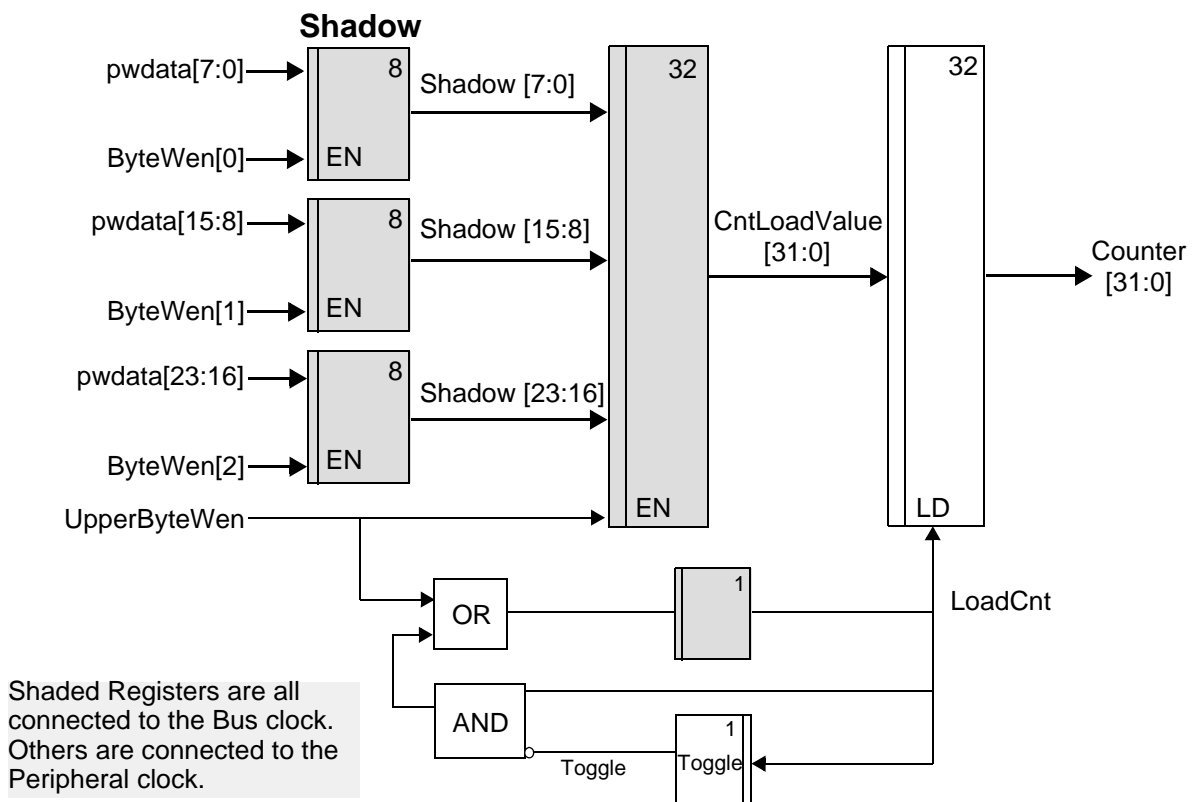
By using the shadow registers, the CntLoadValue is kept stable until it can be changed in one cycle. This allows the counter to be loaded in one access and the state of the counter is not affected by the latency in programming it. When there is a new value to be loaded into the counter initially, this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

8.5.1.2 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.

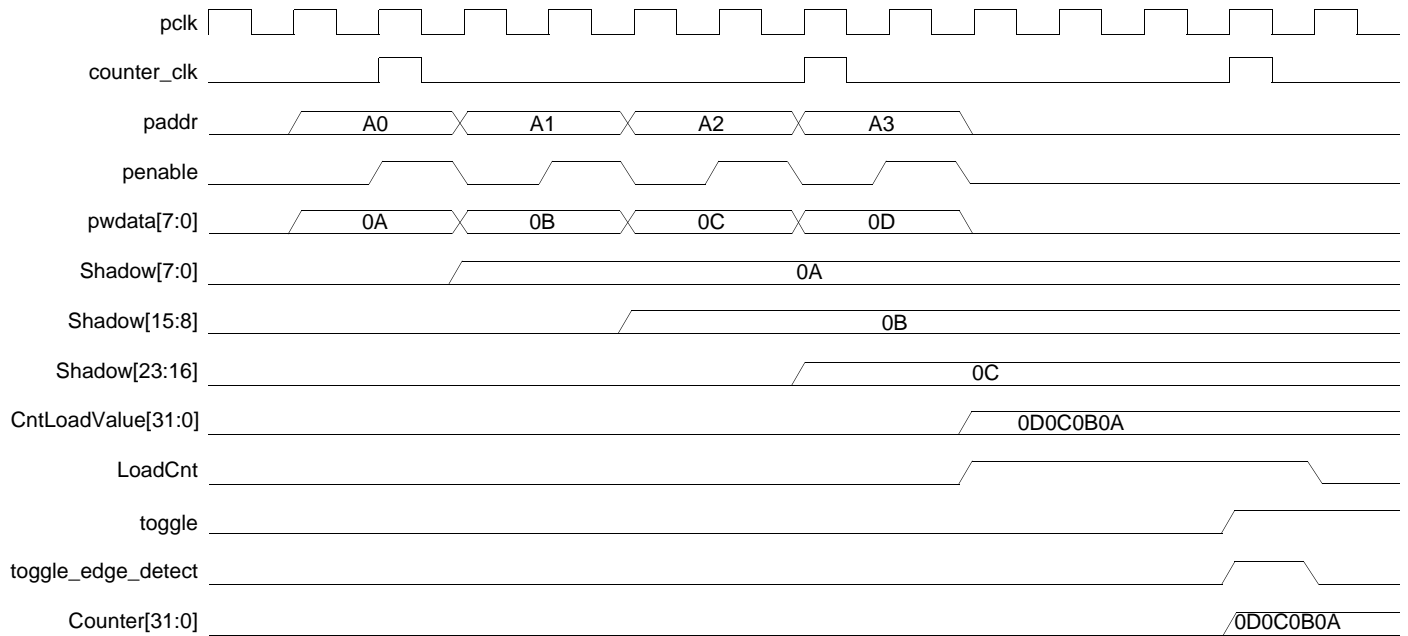
The following figure shows an RTL diagram for the circuitry required to implement the coherent write when the bus and peripheral clocks are synchronous.

Figure 8-6 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.

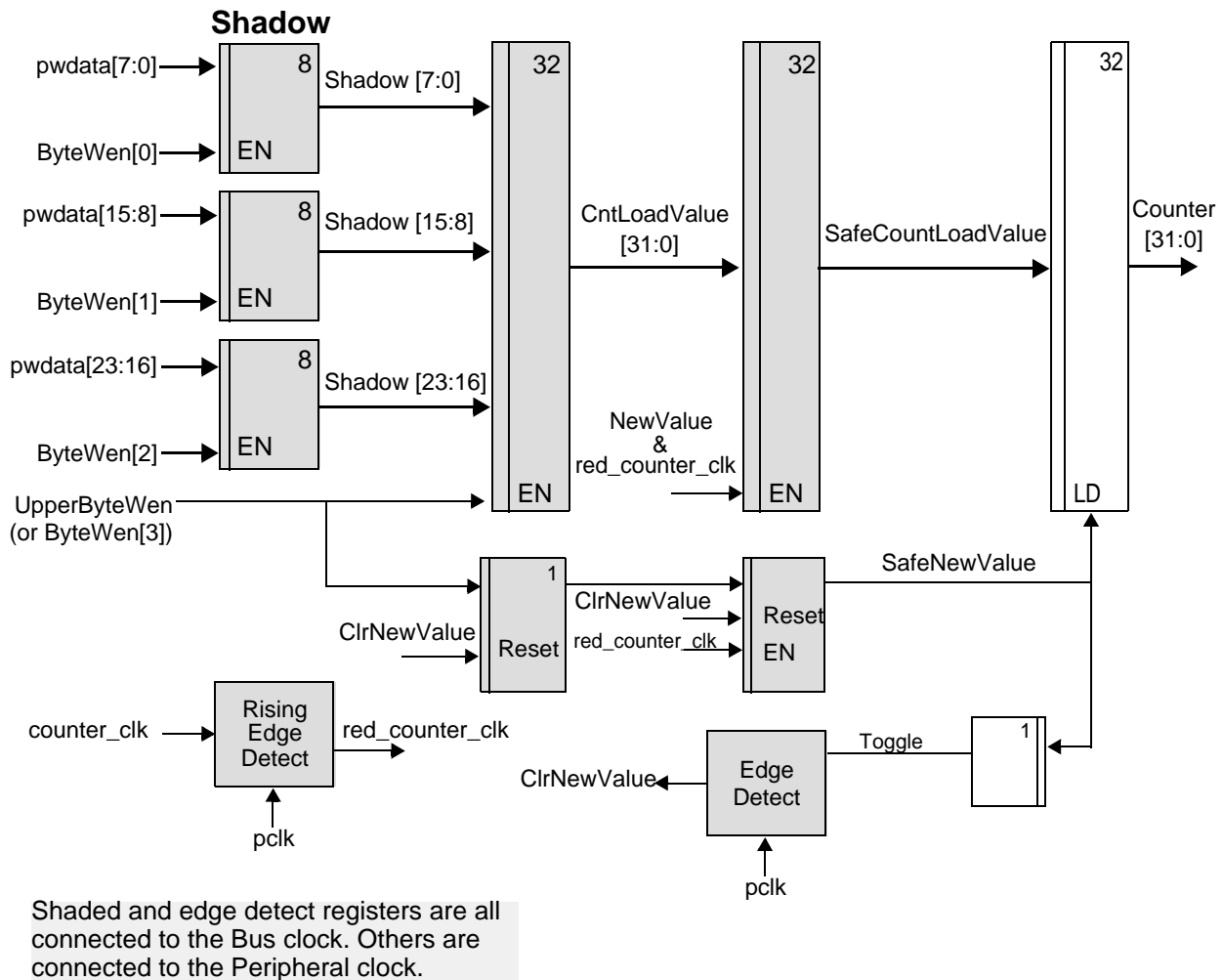
Figure 8-7 Coherent Loading – Synchronous Clocks



8.5.1.3 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 8-8 Coherent Loading – Asynchronous Clocks



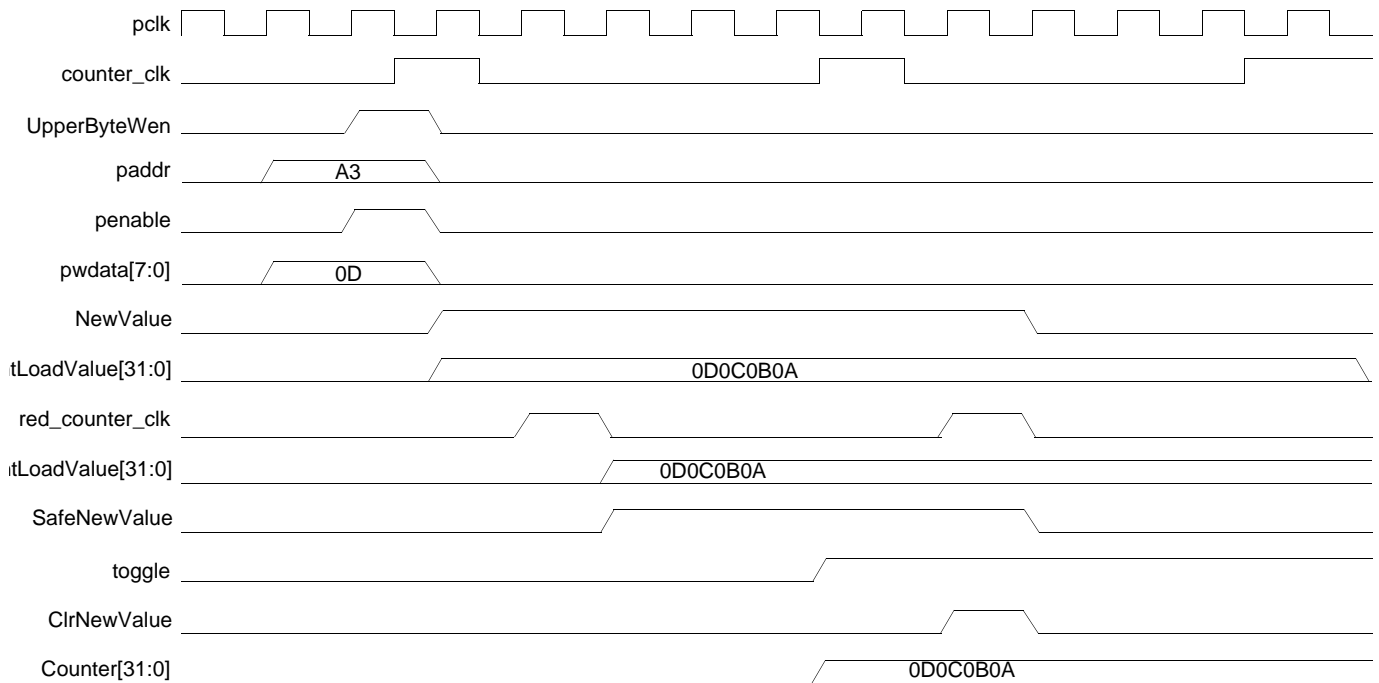
When the clocks are asynchronous, you need to transfer the contents of the register from one clock domain to another. It is not desirable to transfer the entire register through meta-stability registers, as coherency is not guaranteed with this method. The circuitry needed requires the processor clock 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 8-9 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.

8.5.2 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 8-2 Lower Byte Generation

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

Table 8-2 Lower Byte Generation

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

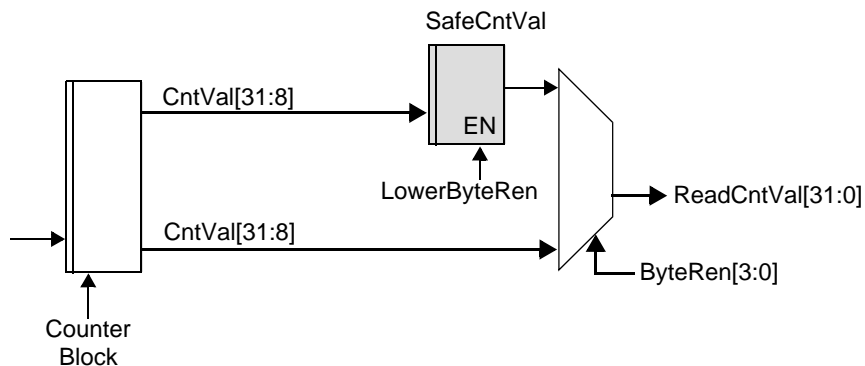
Depending on the bus width and the register width, there may be no need to save the upper bits because the entire register is read in one access, in which case there is no problem with coherency. When the lower byte is read, the remaining upper bytes within the counter register are transferred into a 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:

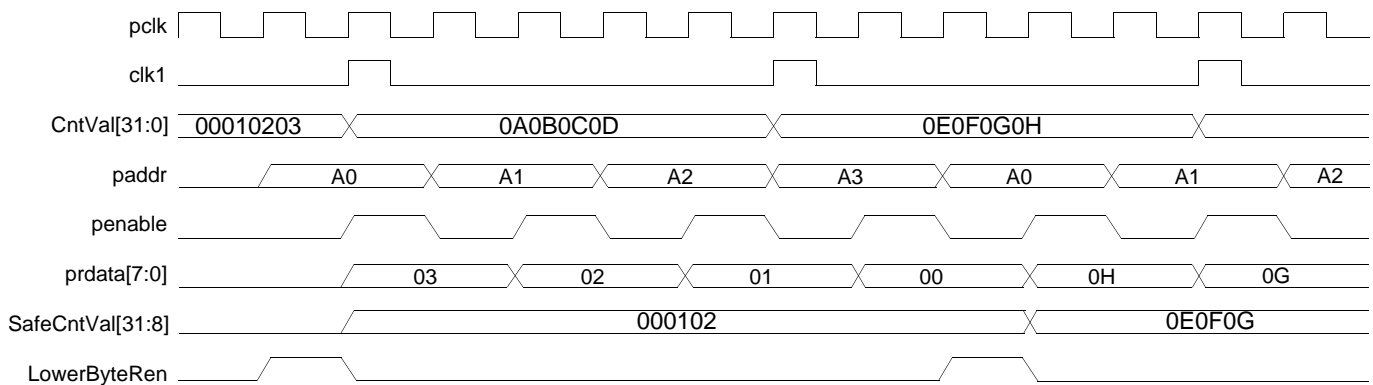
- Identical and/or synchronous
- Asynchronous

8.5.2.1 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 8-10 Coherent Registering – Synchronous Clocks

Shaded registers are clocked with the processor clock.

Figure 8-11 Coherent Registering – Synchronous Clocks

8.5.2.2 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

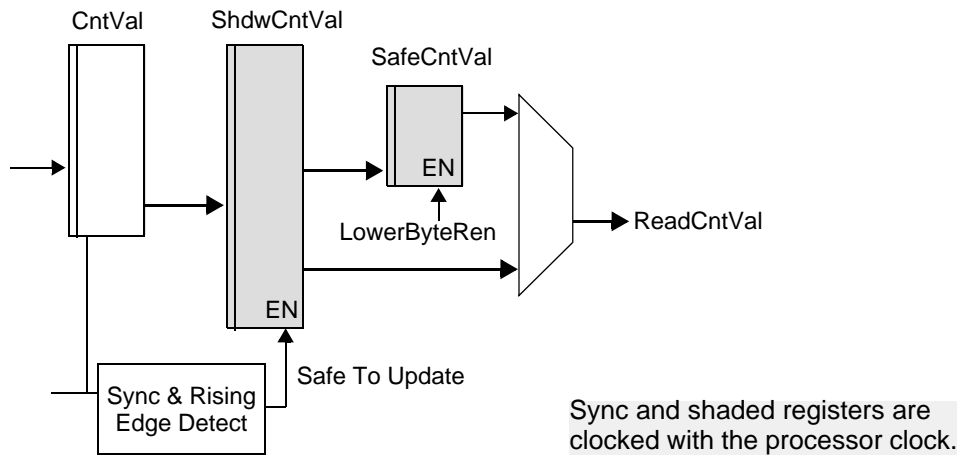
You must read LSB to MSB when the bus width is narrower than the counter width.

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 8-12 Coherency and Shadow Registering – Asynchronous Clocks



8.6 Performance

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

8.6.1 Power Consumption, Frequency, and Area Results

Table 8-3 provides information about the synthesis results (power consumption, frequency, and area) of the DW_apb_gpio using the industry standard 28nm technology library and how it affects performance.

Table 8-3 Power Consumption, Frequency, and Area Results for DW_apb_gpio Using 28nm Technology Library

| Configuration | Operating Frequency | Gate Count | Static Power Consumption | Dynamic Power Consumption |
|---|---------------------|------------|--------------------------|---------------------------|
| Default Configuration | pclk: 200 MHz | 1622 gates | 0.0272 uW | 5.01 uW |
| Minimum Configuration: APB_DATA_WIDTH=8 GPIO_DEBOUNCE=0 GPIO_DFLT_DIR_A=1 GPIO_DFLT_SRC_A=1 GPIO_HW_PORTA=0 GPIO_ID=0 GPIO_INT_POL=1 GPIO_NUM_PORTS=1 GPIO_PA_SYNC_EXT_DATA=0 GPIO_PORTA_INTR=0 GPIO_PWIDTH_A=1 | pclk: 200 MHz | 117 gates | 0.00183uW | 0.343uW |

| Configuration | Operating Frequency | Gate Count | Static Power Consumption | Dynamic Power Consumption |
|---|--|------------|--------------------------|---------------------------|
| Maximum Configuration: APB_DATA_WIDTH=32 GPIO_DEBOUNCE=1 GPIO_DFLT_DIR_A=0 GPIO_DFLT_DIR_B=1 GPIO_DFLT_DIR_C=0 GPIO_DFLT_DIR_D=1 GPIO_DFLT_SRC_A=0 GPIO_DFLT_SRC_B=1 GPIO_DFLT_SRC_C=0 GPIO_DFLT_SRC_D=1 GPIO_HW_PORTA=1 GPIO_HW_PORTB=1 GPIO_HW_PORTC=1 GPIO_HW_PORTD=1 GPIO_ID=1 GPIO_ID_NUM=0x12345678 GPIO_ID_WIDTH=32 GPIO_INT_POL=1 GPIO_NUM_PORTS=4 GPIO_PA_SYNC_EXT_DATA=1 GPIO_PB_SYNC_EXT_DATA=1 GPIO_PC_SYNC_EXT_DATA=1 GPIO_PD_SYNC_EXT_DATA=1 GPIO_PORTA_INTR=1 GPIO_PWIDTH_A=32 GPIO_PWIDTH_B=32 GPIO_PWIDTH_C=32 GPIO_PWIDTH_D=32 | pclk: 200 MHz dbclk: 200 MHz pclk_intr: 200MHz | 8225 gates | 0.136uW | 28.4uW |

A

Synchronizer Methods

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

This appendix contains the following sections:

- “Synchronizers Used in DW_apb_gpio” on page 140
- “Synchronizer 1: Simple Double Register Synchronizer (DW_apb_gpio)” on page 141



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:

<https://www.synopsys.com/dw/buildingblock.php>

A.1 Synchronizers Used in DW_apb_gpio

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_gpio 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_gpio

| Synchronizer Module File | Synchronizer Type and Number |
|--------------------------|---|
| DW_apb_gpio_bcm21.v | Synchronizer 1: Simple Multiple register synchronizer |

**Note**

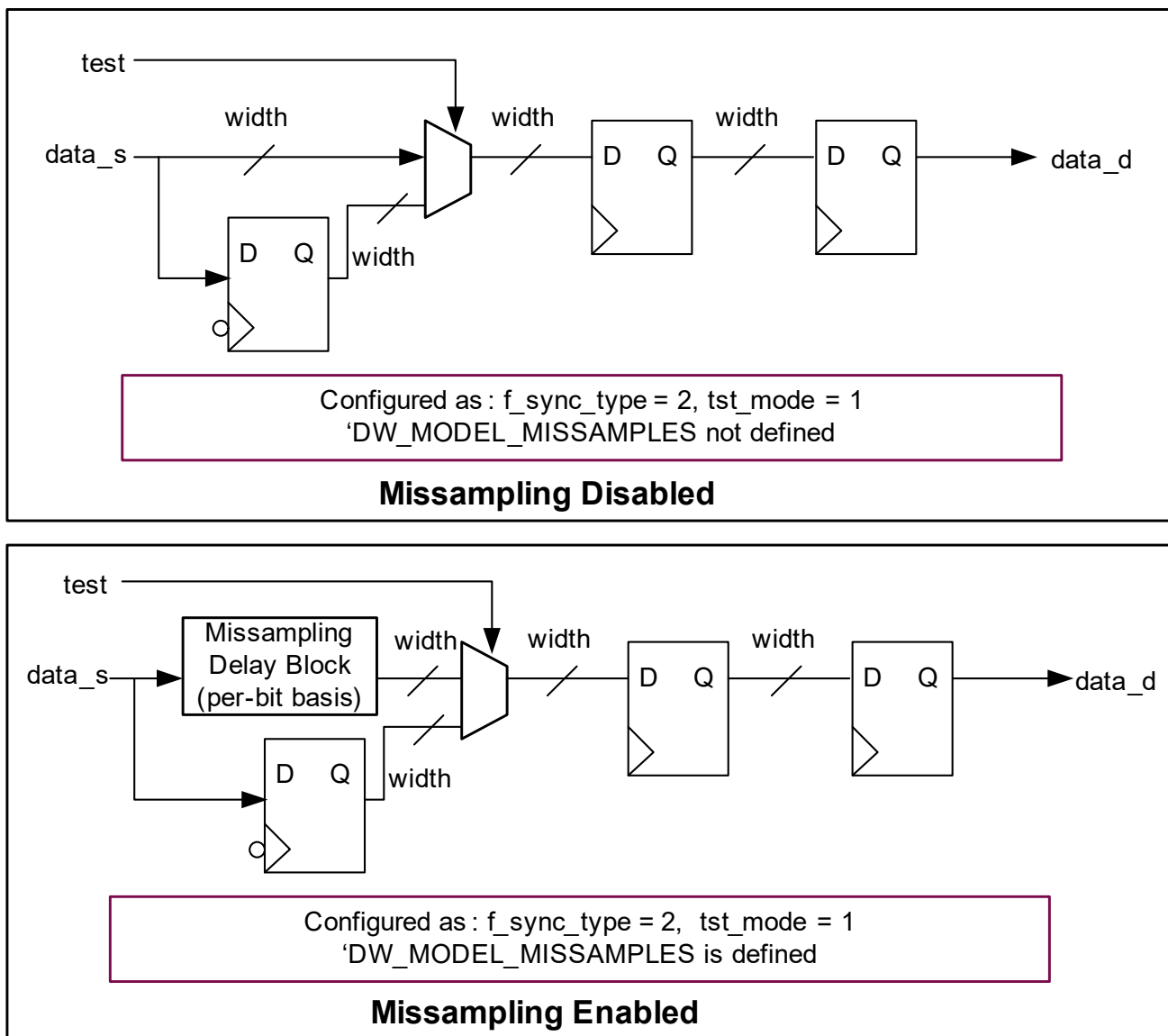
The BCM21 is a basic multiple register based synchronizer module used in the design. It can be replaced with equivalent technology specific synchronizer cell.

A.2 Synchronizer 1: Simple Double Register Synchronizer (DW_apb_gpio)

This is a single clock data bus synchronizer for synchronizing control signals that crosses asynchronous clock boundaries. The synchronization scheme uses two stage synchronization process (Figure A-1) both using positive edge of clock.

This is a single clock data bus synchronizer for synchronizing data that crosses asynchronous clock boundaries. Your core may have parameters that allow you to configure for the number of synchronizing stages (2 or 3), the style of first-stage capturing flip-flop needed (negative or positive edge-triggered), and insertion of 'hold latches' to facilitate scan testing.

Figure A-1 Block Diagram of Synchronizer 1 with Two-Stage Synchronization (Both Positive Edges)



B

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 B-1 Internal Parameters

| Parameter Name | Equals To |
|------------------------|---|
| GPIO_ADDR_SLICE_LHS | 6 |
| GPIO_COMBINED | 1 |
| GPIO_DFLT_SRC_RESET_A | {{function_of: GPIO_PORTA_SINGLE_CTL GPIO_DFLT_SRC_A GPIO_PWIDTH_A}} |
| GPIO_DFLT_SRC_RESET_B | {{function_of: GPIO_PORTB_SINGLE_CTL GPIO_DFLT_SRC_B GPIO_PWIDTH_B}} |
| GPIO_DFLT_SRC_RESET_C | {{function_of: GPIO_PORTC_SINGLE_CTL GPIO_DFLT_SRC_C GPIO_PWIDTH_C}} |
| GPIO_DFLT_SRC_RESET_D | {{function_of: GPIO_PORTD_SINGLE_CTL GPIO_DFLT_SRC_D GPIO_PWIDTH_D}} |
| GPIO_ENCODED_APB_WIDTH | {{function_of: APB_DATA_WIDTH}} |
| GPIO_ENCODED_ID_WIDTH | {{function_of: GPIO_ID_WIDTH}} |
| GPIO_ENCODED_NUM_PORTS | {{function_of: GPIO_NUM_PORTS}} |
| GPIO_ENCODED_PWIDTH_A | {{function_of: GPIO_PWIDTH_A}} |
| GPIO_ENCODED_PWIDTH_B | {{function_of: GPIO_PWIDTH_B}} |

Table B-1 Internal Parameters (Continued)

| Parameter Name | Equals To |
|---------------------------|--|
| GPIO_ENCODED_PWIDTH_C | {{function_of: GPIO_PWIDTH_C}} |
| GPIO_ENCODED_PWIDTH_D | {{function_of: GPIO_PWIDTH_D}} |
| GPIO_INDIVIDUAL | 0 |
| GPIO_L_DEBOUNCE | (GPIO_DEBOUNCE ? 1'b1 : 1'b0) |
| GPIO_L_HW_PORTA | (GPIO_HW_PORTA ? 1'b1 : 1'b0) |
| GPIO_L_HW_PORTB | (GPIO_HW_PORTB ? 1'b1 : 1'b0) |
| GPIO_L_HW_PORTC | (GPIO_HW_PORTC ? 1'b1 : 1'b0) |
| GPIO_L_HW_PORTD | (GPIO_HW_PORTD ? 1'b1 : 1'b0) |
| GPIO_L_ID | (GPIO_ID ? 1'b1 : 1'b0) |
| GPIO_L_PORTA_INTR | (GPIO_PORTA_INTR ? 1'b1 : 1'b0) |
| GPIO_SWPORTA_CTL_REG_SIZE | ((GPIO_PORTA_SINGLE_CTL == 1) ? 1 : GPIO_PWIDTH_A) |
| GPIO_SWPORTB_CTL_REG_SIZE | ((GPIO_PORTB_SINGLE_CTL == 1) ? 1 : GPIO_PWIDTH_B) |
| GPIO_SWPORTC_CTL_REG_SIZE | ((GPIO_PORTC_SINGLE_CTL == 1) ? 1 : GPIO_PWIDTH_C) |
| GPIO_SWPORTD_CTL_REG_SIZE | ((GPIO_PORTD_SINGLE_CTL == 1) ? 1 : GPIO_PWIDTH_D) |
| GPIO_VERSION_ID | 32'h3231332a |
| PORTA_INT_BOTHEDGE | (GPIO_INT_BOTH_EDGE ? 1'b1 : 1'b0) |
| PORTA_SINGLE_CTL | (GPIO_PORTA_SINGLE_CTL ? 1'b1 : 1'b0) |
| PORTB_SINGLE_CTL | (GPIO_PORTB_SINGLE_CTL ? 1'b1 : 1'b0) |
| PORTC_SINGLE_CTL | (GPIO_PORTC_SINGLE_CTL ? 1'b1 : 1'b0) |
| PORTD_SINGLE_CTL | (GPIO_PORTD_SINGLE_CTL ? 1'b1 : 1'b0) |
| POW_2_GPIO_ID_WIDTH | {{function_of: GPIO_ID_WIDTH}} |

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. |
| AHB | Advanced High-performance Bus — high-performance system backbone bus. AHB supports the efficient connection of processors, on-chip memories and off-chip external memory interfaces (Arm® Limited specification). |
| AMBA | Advanced Microcontroller Bus Architecture — a trademarked name by Arm® Limited that defines an on-chip communication standard for high speed microcontrollers. |
| APB | Advanced Peripheral Bus — optimized for minimal power consumption and reduced interface complexity to support peripheral functions (Arm® Limited specification). |
| APB bridge | DW_apb submodule that converts protocol between the AHB bus and APB bus. |
| application design | Overall chip-level design into which a subsystem or subsystems are integrated. |
| arbiter | AMBA bus submodule that arbitrates bus activity between masters and slaves. |
| 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. |

| | |
|-------------------------------------|--|
| bus bridge | Logic that handles the interface and transactions between two bus standards, such as AHB and APB. See APB bridge. |
| 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 Synthesizable Components | The Synopsys name for the collection of AMBA-compliant coreKits and verification models delivered with DesignWare and used with coreConsultant or coreAssembler to quickly build DesignWare Synthesizable Component designs. |

| | |
|----------------------|--|
| 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

A

active command queue

definition [145](#)

activity

definition [145](#)

AHB

definition [145](#)

AMBA

definition [145](#)

APB

definition [145](#)

APB bridge

definition [145](#)

application design

definition [145](#)

arbiter

definition [145](#)

B

BFM

definition [145](#)

big-endian

definition [145](#)

Block diagram, of DW_apb_gpio [17](#)

blocked command stream

definition [145](#)

blocking command

definition [145](#)

bus bridge

definition [146](#)

C

Coherency

about [128](#)

read [134](#)

write [128](#)

command channel

- definition [146](#)
- command stream
 - definition [146](#)
- component
 - definition [146](#)
- configuration
 - definition [146](#)
- configuration intent
 - definition [146](#)
- Control flow, of DW_apb_gpio [19](#)
- core
 - definition [146](#)
- core developer
 - definition [146](#)
- core integrator
 - definition [146](#)
- coreAssembler
 - definition [146](#)
- coreConsultant
 - definition [146](#)
- coreKit
 - definition [146](#)
- Customer Support [12](#)
- cycle command
 - definition [146](#)

D

- Data flow, of DW_apb_gpio [19](#)
- De-bounce operation
 - about [24](#)
 - RTL diagram of [24](#)
- decoder
 - definition [146](#)
- design context
 - definition [146](#)
- design creation
 - definition [146](#)
- Design View
 - definition [146](#)
- DesignWare cores
 - definition [147](#)
- DesignWare Library
 - definition [147](#)
- DesignWare Synthesizable Components
 - definition [146](#)
- dual role device

definition [147](#)

DW_apb

slaves

read timing operation [127](#)

write timing operation [126](#)

DW_apb_gpio [122](#)

block diagram of [17](#)

control flow of [19](#)

data flow of [19](#)

description [19](#)

functional description of [17](#)

testbench

overview of [120](#)

E

Edge detection, of interrupts [27](#)

endian

definition [147](#)

Environment, licenses [18](#)

External signals, reading [21](#)

F

Full-Functional Mode

definition [147](#)

Functional description [17](#), [19](#)

G

GPIO

definition [147](#)

GTECH

definition [147](#)

H

hard IP

definition [147](#)

Hardware control, about [21](#)

HDL

definition [147](#)

I

IIP

definition [147](#)

implementation view

definition [147](#)

instantiate

definition [147](#)

interface

definition [147](#)

Interrupts

- about [22](#)
- block diagram of [23](#)
- edge detection [27](#)
- level-sensitive [34](#)
- synchronization to system clock [26](#)

IP

- definition [147](#)

L

Level-sensitive interrupts [34](#)

Licenses [18](#)

little-endian

- definition [147](#)

M

MacroCell

- definition [147](#)

master

- definition [147](#)

model

- definition [147](#)

monitor

- definition [147](#)

N

non-blocking command

- definition [147](#)

O

Operation modes

- hardware control, about [21](#)

- software control, about [20](#)

P

peripheral

- definition [147](#)

R

Read coherency

- about [134](#)

- and asynchronous clocks [136](#)

- and synchronous clocks [135](#)

Reading, external signals [21](#)

Reading, from unused locations [123](#)

RTL

- definition [148](#)

runtest.log [122](#)

S

SDRAM

- definition [148](#)
- SDRAM controller
 - definition [148](#)
- Simple double register synchronizer [141](#)
- Simulation
 - command line output files [122](#)
 - from command line [122](#)
 - of DW_apb_gpio coreKit [120](#)
 - results [122](#)
- slave
 - definition [148](#)
- SoC
 - definition [148](#)
- SoC Platform
 - AHB contained in [15](#)
 - APB, contained in [15](#)
 - defined [15](#)
- soft IP
 - definition [148](#)
- Software control, about [20](#)
- static controller
 - definition [148](#)
- subsystem
 - definition [148](#)
- Synchronization
 - of interrupt signals to system clock [26](#)
- Synchronizer
 - simple double register [141](#)
- synthesis intent
 - definition [148](#)
- synthesizable IP
 - definition [148](#)
- T**
- technology-independent
 - definition [148](#)
- test.log [122](#)
- testbench
 - output files [122](#)
- Testsuite Regression Environment (TRE)
 - definition [148](#)
- Timing
 - read operation of DW_apb slave [127](#)
 - write operation of DW_apb slave [126](#)
- TRE
 - definition [148](#)

V

Verification

- of DW_apb_gpio coreKit [120](#)
- output file [122](#)
- results [122](#)

VIP

- definition [148](#)

W

workspace

- definition [148](#)

wrap

- definition [148](#)

wrapper

- definition [148](#)

Write coherency

- about [128](#)
- and asynchronous clocks [133](#)
- and identical clocks [130](#)
- and synchronous clocks [131](#)

Z

zero-cycle command

- definition [148](#)