General Purpose I/O Controller Integration Guide

Document Version 1.0.1
December 20, 2010

Printed in the USA

IPextreme, Inc.
54 North Central Ave.
Suite 204
Campbell, CA 95008
www.ip-extreme.com

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

This integration guide provides the information you need to integrate the General Purpose I/O Controller (GPIO) into an ASIC or FPGA device. The GPIO is the General Purpose I/O Controller synthesizable IP core from National Semiconductor.

Intended Audience

This manual is intended for chip designers who are integrating the GPIO into an ASIC or FPGA design. Readers of this manual are expected to be familiar with chip design technology and electronic design automation (EDA) tools.

Document Organization

This integration guide is organized as follows:

- Chapter 1: Introduction provides an overview of the GPIO and its role in a system-on-chip (SoC).
- Chapter 2: Top-Level Signal Summary defines the top-level interface signals of the GPIO.
- Chapter 3: Configuration Parameters describes the GPIO configuration parameters. You configure the GPIO on the IPextreme IP Distribution and Support Portal as described in the *XPack Guide for AMBA IP.*
- Chapter 4: Clock and Reset describes the GPIO clock and reset structure, including the (optional) local clock gating implementation.
- Chapter 5: Interface Connections and Timing describes the connections, protocol, and timing of the top-level interfaces.
- Chapter 6: Integration Testbench describes the testbench that is included with the GPIO product.
- Chapter 7: Testability describes the use of the GPIO design-for-test (DFT) signals.

Related Documentation

For the complete list of GPIO documentation, refer to the *General Purpose I/O Controller Documentation Guide*.

Getting Help

To get help with the GPIO product, send e-mail to support@ip-extreme.com.

If you are an IPextreme customer with a current support contract, log on to the IPextreme IP Distribution and Support Portal at https://xpack.ip-extreme.com.

For information about IPextreme and IPextreme products, go to www.ip-extreme.com.

CHAPTER 1 INTRODUCTION

The General Purpose I/O Controller (GPIO) provides a bidirectional port with alternate function capability to an external off-chip interface. As an external interface, the GPIO is designed for direct connection to I/O pads.

Features

GPIO features include:

- 16 or 32 I/O channels; each channel corresponds to one off-chip interface pin connected to the GPIO through an I/O pad
- Programmable pin direction
- Internal weak pull-up, pull-down
- Direct, low-impedance analog input
- Read-back on all registers
- Each pin may be controlled by other modules through its software-selectable alternate function and alternate source
- Selectable high drive current option
- Functional test mode to directly control the pad interface signals from user-implemented test logic

In summary, each GPIO channel connects one off-chip I/O pad to one of the following sources:

- In general-purpose I/O mode, a channel's direction is controlled and its input/output data is read and written directly by the host system through software-accessible registers within the GPIO.
- If alternate function is programmed for a channel, that channel's direction is controlled and its input/output data is read and written from a peripheral connected to one of two alternate sources (A or B). Each channel supports two alternate sources; software selects the currently active source (A or B) for each channel.

- In functional test mode, test logic controls the direction and drives or receives the I/O data for each channel.

System Overview

Figure 1 shows an example GPIO implementation in an AMBA system environment. The GPIO connects to surrounding logic through the following top-level interfaces:

- AMBA 2 APB – The host interface for transferring control/status information and data. In the example system shown in Figure 1, the GPIO APB interface connects to the host CPU's AMBA AHB interface through an AMBA AHB-to-APB Bridge. However, the GPIO can connect to the host CPU through any AMBA 2 APB compliant interface.
- Pad interface – Input/output data and control signals to the I/O pad associated with each channel
- Peripheral interface – Provides two software-selectable alternate sources (A and B) for a channel's input/output data and control
- Interrupt signal to host CPU or optional interrupt controller
- Clock and reset signals
- DFT signals **–** Control testability of GPIO logic

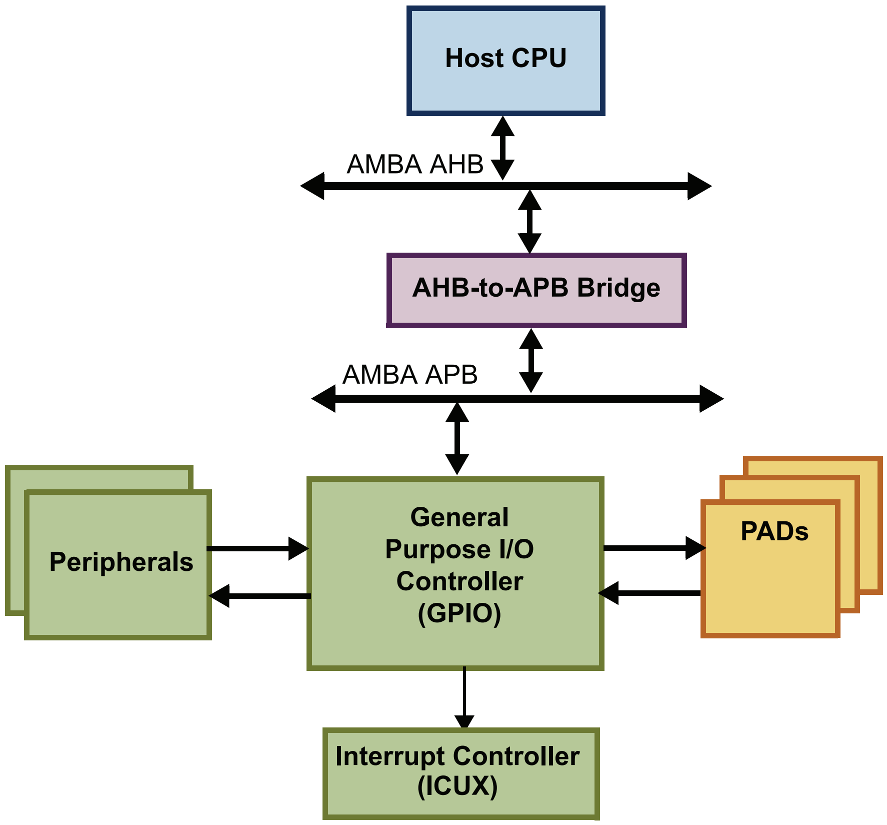

Host CPU
AMBA AHB
AHB-to-APB Bridge
AMBA APB
Peripherals
General Purpose I/O Controller (GPIO)
PADs
Interrupt Controller (ICUX)

FIGURE 1: GPIO IN EXAMPLE APPLICATION SYSTEM

NOTE: The following blocks, shown in Figure 1, are also available within the National Semiconductor IP Library for AMBA Interconnect from IPextreme: AHB-to-APB Bridge and Interrupt Controller.

RTL Structure

Table 1 lists the Verilog RTL source files associated with each of the top-level submodules of the GPIO.

NOTE: The ck_gpio_xt_geh0 block is the only block instantiated at top level of gpio.

TABLE 1: RTL BLOCK STRUCTURE

Block Name	Description	RTL Instance Name	RTL Source Files
gpio	GPIO top level and parameter file	—	gpio.v ck_gpio_xt_geh0_params.v ck_gpio_xt_geh0.v

The RTL code references cells such as inverter, nand2, multiplexer, and buffer. However, these cells are described in a technology-independent format. The instantiated cells are:

- buf_gea1
- inv_gea1
- nand2_gea1

CHAPTER 2
TOP-LEVEL SIGNAL SUMMARY

Figure 2 shows the GPIO top-level signals. Table 2 defines the top-level signals.

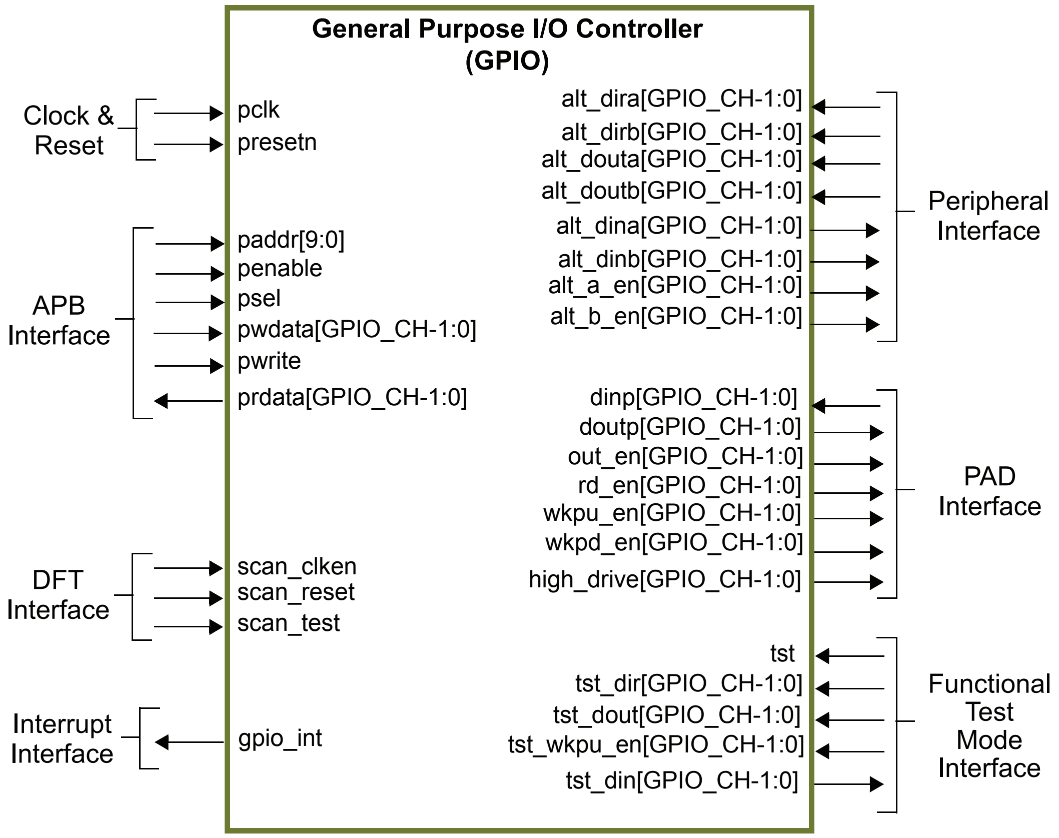


FIGURE 2: GPIO INTERFACE DIAGRAM

TABLE 2: TOP-LEVEL SIGNAL SUMMARY

Signal	I/O	Width	Function
Clock Signals (see "Clocking" on page 21)			
pclk	In	1	Main GPIO clock and AMBA APB clock
Reset Signals (see "Reset" on page 24)			
presetn	In	1	Asynchronous reset, active low
AMBA APB Interface (see "AMBA APB Interface" on page 25)			
paddr[9:0]	In	10	APB address bus
penable	In	1	APB enable – Indicates the second cycle of an AMBA APB transfer
psel	In	1	APB select – Indicates that the slave device is selected for a data transfer
pwdata [GPIO_CH-1:0]	In	GPIO_CH (16 or 32)	APB write data bus **Dependency**: Width of *pwdata* depends on parameter GPIO_CH.
pwrite	In	1	APB write enable: • 0: Read access • 1: Write access
prdata [GPIO_CH-1:0]	Out	GPIO_CH (16 or 32)	APB read data bus **Dependency**: Width of *prdata* depends on parameter GPIO_CH.
Interrupt Interface (see "Interrupt Interface" on page 31)			
gpio_int	Out	1	Interrupt, active high, level-sensitive
Peripheral Interface (see "Peripheral Interface" on page 28)			
alt_dira [GPIO_CH-1:0]	In	GPIO_CH (16 or 32)	Alternate function A direction control: • 0: Input • 1: Output **Note**: Assign to '0' if unused. **Dependency**: Width of *alt_dira* depends on parameter GPIO_CH.
alt_dirb [GPIO_CH-1:0]	In	GPIO_CH (16 or 32)	Alternate function B direction control: • 0: Input • 1: Output **Note**: Assign to '0' if unused. **Dependency**: Width of *alt_dirb* depends on parameter GPIO_CH.
alt_douta [GPIO_CH-1:0]	In	GPIO_CH (16 or 32)	Alternate function A data output **Note**: Assign to '0' if unused. **Dependency**: Width of *alt_douta* depends on parameter GPIO_CH.
alt_doutb [GPIO_CH-1:0]	In	GPIO_CH (16 or 32)	Alternate function B data output **Note**: Assign to '0' if unused. **Dependency**: Width of *alt_doutb* depends on parameter GPIO_CH.
alt_dina [GPIO_CH-1:0]	Out	GPIO_CH (16 or 32)	Alternate function A data input **Dependency**: Width of *alt_dina* depends on parameter GPIO_CH.

TABLE 2: TOP-LEVEL SIGNAL SUMMARY

Signal	I/O	Width	Function
alt_dinb [GPIO_CH-1:0]	Out	GPIO_CH (16 or 32)	Alternate function B data input **Dependency**: Width of *alt_dinb* depends on parameter GPIO_CH.
alt_a_en [GPIO_CH-1:0]	Out	GPIO_CH (16 or 32)	Alternate function source A enable status: • 0: Alternate function source A is disabled • 1: Alternate function source A is enabled **Dependency**: Width of *alt_a_en* depends on parameter GPIO_CH.
alt_b_en [GPIO_CH-1:0]	Out	GPIO_CH (16 or 32)	Alternate function source B enable status: • 0: Alternate function source B is disabled • 1: Alternate function source B is enabled **Dependency**: Width of *alt_b_en* depends on parameter GPIO_CH.
Pad Interface (see “Pad Interface” on page 27)			
dinp [GPIO_CH-1:0]	In	GPIO_CH (16 or 32)	Pad data input **Dependency**: Width of *dinp* depends on parameter GPIO_CH.
doutp [GPIO_CH-1:0]	Out	GPIO_CH (16 or 32)	Pad data output **Dependency**: Width of *doutp* depends on parameter GPIO_CH.
out_en [GPIO_CH-1:0]	Out	GPIO_CH (16 or 32)	Pad output buffer enable: • 0: Pad output buffer disabled/tristate • 1: Pad output buffer enabled **Dependency**: Width of *out_en* depends on parameter GPIO_CH.
rd_en [GPIO_CH-1:0]	Out	GPIO_CH (16 or 32)	Pad read enable: • 0: Input read buffer disabled • 1: Input read buffer enabled **Dependency**: Width of *rd_en* depends on parameter GPIO_CH.
wkpu_en [GPIO_CH-1:0]	Out	GPIO_CH (16 or 32)	Pad weak pull-up enable: • 0: Pad pull-up disabled • 1: Pad pull-up enabled **Dependency**: Width of *wkpu_en* depends on parameter GPIO_CH.
wkpd_en [GPIO_CH-1:0]	Out	GPIO_CH (16 or 32)	Pad weak pull-down enable: • 0: Pad pull-down disabled • 1: Pad pull-down enabled **Dependency**: Width of *wkpd_en* depends on parameter GPIO_CH.
high_drive [GPIO_CH-1:0]	Out	GPIO_CH (16 or 32)	Pad high drive enable: • 0: Pad high drive disabled • 1: Pad high drive enabled **Dependency**: Width of *high_drive* depends on parameter GPIO_CH.

TABLE 2: TOP-LEVEL SIGNAL SUMMARY

Signal	I/O	Width	Function
Functional Test Mode Interface (see "Functional Test Mode Interface" on page 30)			
tst	In	1	Enable or disable functional test mode: • 0: Normal operation • 1: Enable functional test mode
tst_dir [GPIO_CH-1:0]	In	GPIO_CH (16 or 32)	Functional test mode direction: • 0: Input • 1: Output **Dependency**: Width of *tst_dir* depends on parameter GPIO_CH.
tst_dout [GPIO_CH-1:0]	In	GPIO_CH (16 or 32)	Functional test mode data output **Dependency**: Width of *tst_dout* depends on parameter GPIO_CH.
tst_wkpu_en [GPIO_CH-1:0]	In	GPIO_CH (16 or 32)	Functional test mode pull-up enable: • 0: Pad pull-up disabled (in functional test mode) • 1: Pad pull-up enabled (in functional test mode) **Dependency**: Width of *tst_wkpu_en* depends on parameter GPIO_CH.
tst_din [GPIO_CH-1:0]	Out	GPIO_CH (16 or 32)	Functional test mode data input **Dependency**: Width of *tst_din* depends on parameter GPIO_CH.
Scan Test and Test Mode Signals (see "Testability" on page 37)			
scan_clken	In	1	Scan test NAND clock enable, used only if clock gating is enabled in the GPIO design: • If clock gating is enabled (GATED_CLOCK = true), connect *scan_clken* to the chip's scan_shift/scan_enable signal. • If GATED_CLOCK = false, *scan_clken* is tied to '0' internally.
scan_test	In	1	Scan test enable. Assign to '1' during test.
scan_reset	In	1	Scan reset

CHAPTER 3
CONFIGURATION PARAMETERS

Table 3 lists the GPIO configuration parameters. You select values for the parameters on the IPextreme IP Distribution and Support Portal as described in the *XPack Guide for AMBA IP*. The parameters are defined in the file ck_gpio_xt_geh0_params.v.

NOTE: When GATED_CLOCK = true, there are additional setup and hold check related parameters enabled in the PARAMETERS view on the IPextreme IP Distribution and Support Portal. The setup and hold check parameters are used for synthesis script generation and are not actual hardware configuration parameters of the GPIO. See "Local Clock Gating" on page 21 for additional details.

TABLE 3: TOP-LEVEL PARAMETERS

Parameter	Description
GATED_CLOCK	Enables local clock gating within the GPIO. When GATED_CLOCK = true, the GPIO includes local clock gating structures to reduce power consumption in an ASIC implementation. For details about the local clock gating implementation, see "Local Clock Gating" on page 21. For FPGA implementation, IPextreme recommends setting GATED_CLOCK to false. **Range**: True (clock gating enabled) or false (clock gating disabled) **Default**: False **Dependency**: None
GPIO_CH	Number of I/O channels **Range**: 16 or 32 **Default**: 32 **Dependency**: None
PxALT_RESET_VALUE	Reset value of the Port Alternate Function (PxALT) register **Range**: • 0x0000–0xFFFF (16 channels) • 0x0000_0000–0xFFFF_FFFF (32 channels) **Default**: 0x0000_0000 **Dependency**: Width of parameter depends on GPIO_CH.

TABLE 3: TOP-LEVEL PARAMETERS

Parameter	Description
PxALTS_RESET_VALUE	Reset value of the Port Alternate Source (PxALTS) register **Range**: • 0x0000–0xFFFF (16 channels) • 0x0000_0000–0xFFFF_FFFF (32 channels) **Default**: 0x0000_0000 **Dependency**: Width of parameter depends on GPIO_CH.
PxWKPU_RESET_VALUE	Reset value of the Port Weak Pull-up/Pull-down Enable (PxWKPU) register **Range**: • 0x0000–0xFFFF (16 channels) • 0x0000_0000–0xFFFF_FFFF (32 channels) **Default**: 0x0000_0000 **Dependency**: Width of parameter depends on GPIO_CH.
PxPDR_RESET_VALUE	Reset value of the Weak Pull-up/Pull-down Direction (PxPDR) register **Range**: • 0x0000–0xFFFF (16 channels) • 0x0000_0000–0xFFFF_FFFF (32 channels) **Default**: 0x0000_0000 **Dependency**: Width of parameter depends on GPIO_CH.

CHAPTER 4
CLOCK AND RESET

Clocking

The GPIO uses a single input clock, *pclk*, which can optionally be gated for power savings.

Local Clock Gating

NOTE: Local clock gating is a configuration option only if you have a GPIO-SRC license.

If the GPIO is configured to use gated clocks (GATED_CLOCK = true), clock gates are used wherever possible to reduce power consumption (local clock gating) and the *scan_clken* input enables all clocks during scan shift. Therefore, if GATED_CLOCK = true, *scan_clken* must be connected to the scan test scan_shift/scan_enable signal when the GPIO is instantiated in a higher-level design. If GATED_CLOCK = false, *scan_clken* is connected to ‘0’ internally.

The local clock gating uses an inverted clock (*clk_phi2_c*) and a NAND gate for glitch-free gating; the gating signal is driven from the non-inverted clock (*pclk*) as shown in Figure 3. Figure 4 shows the resulting signal timing.

The GPIO Verilog source code uses a consistent naming style for clock nets and clock gating elements:

- Internal clocks nets are named <net_name>_c.
- Instance of clock gating elements are named phmul_<name>.

If you want to keep the instance names within the netlist to find the generic cells during layout, ensure that the cells are marked “dont_touch” or not ungrouped during synthesis. Refer to the Synthesis chapter of the *XPack Guide for AMBA IP* for more information on this topic.

To ensure that there are no glitches on the gated clock signal (*clk1_c*), the clock gating enable signal (*gate_o*) must be stable and meet setup and hold time requirements with respect to the rising and falling edges of the input clock signal (*clk_phi2_c*) at the input of the clock gating NAND2 cell. Figure 5 shows the required timing.

NOTE: See also the *Design Compiler Reference Manual*: Constraints and Timing "Performing Clock Gating Signal Timing Checks" by Synopsys, Inc.

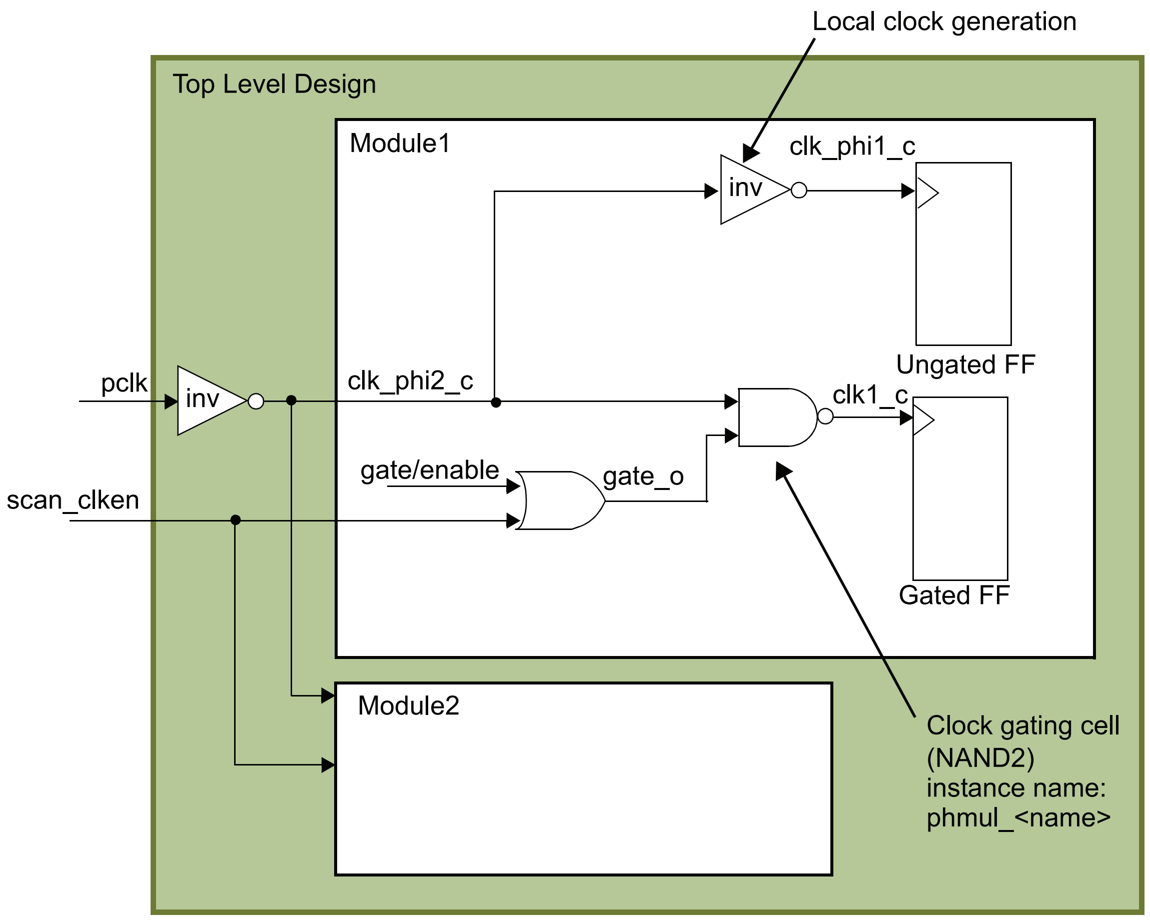


FIGURE 3: CLOCK DISTRIBUTION (WITH CLOCK GATING ENABLED)

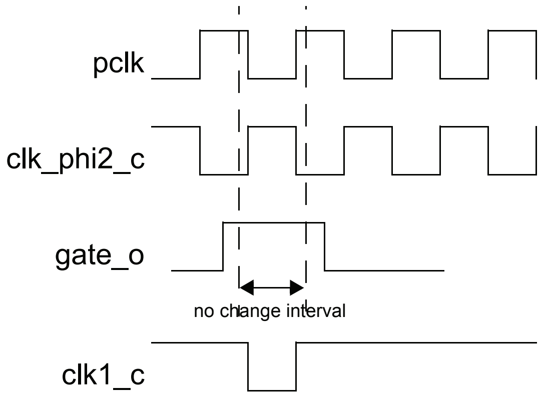


FIGURE 4: CLOCK GATING SIGNALS TIMING

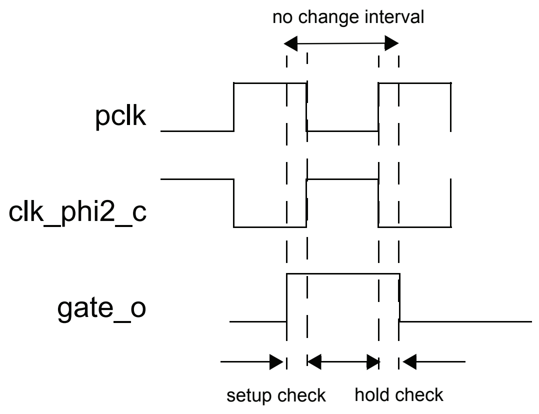


FIGURE 5: CLOCK GATING TIMING CHECKS

The constraint scripts for the GPIO include appropriate commands for the EDA tools (Design Compiler, RTL Compiler) to check the required setup and hold times:

- The setup check ensures that the clock-gating signal is stable for a given time interval before the clock input of the gating cell changes (rising clock edge for NAND clock gating).
- The hold check ensures that the clock-gating signal remains stable for a given time interval after the clock input has changed (falling clock edge for NAND clock gating).

The timing specifications for the setup/hold checks are configurable through parameters that you specify when you configure the GPIO on the IPextreme IP Distribution and Support Portal (see Table 4).

Note: The values for the clock gating setup and hold check parameters are technology-dependent.

TABLE 4: CLOCK GATING TIMING CHECK CONSTRAINT PARAMETERS

Parameter	Default	Range	Description
setup_check	0.3	Float value in ns	Clock gating setup check value
hold_check	0.3	Float value in ns	Clock gating hold check value

If you do not want to use local clock gating (for example, for an FPGA implementation), set GATED_CLOCK to false. If GATED_CLOCK = false:

- There is no local clock gating.
- The synthesis tool removes the two inverters from the clock tree as shown in Figure 6.
- The setup_check and hold_check parameters are not used.

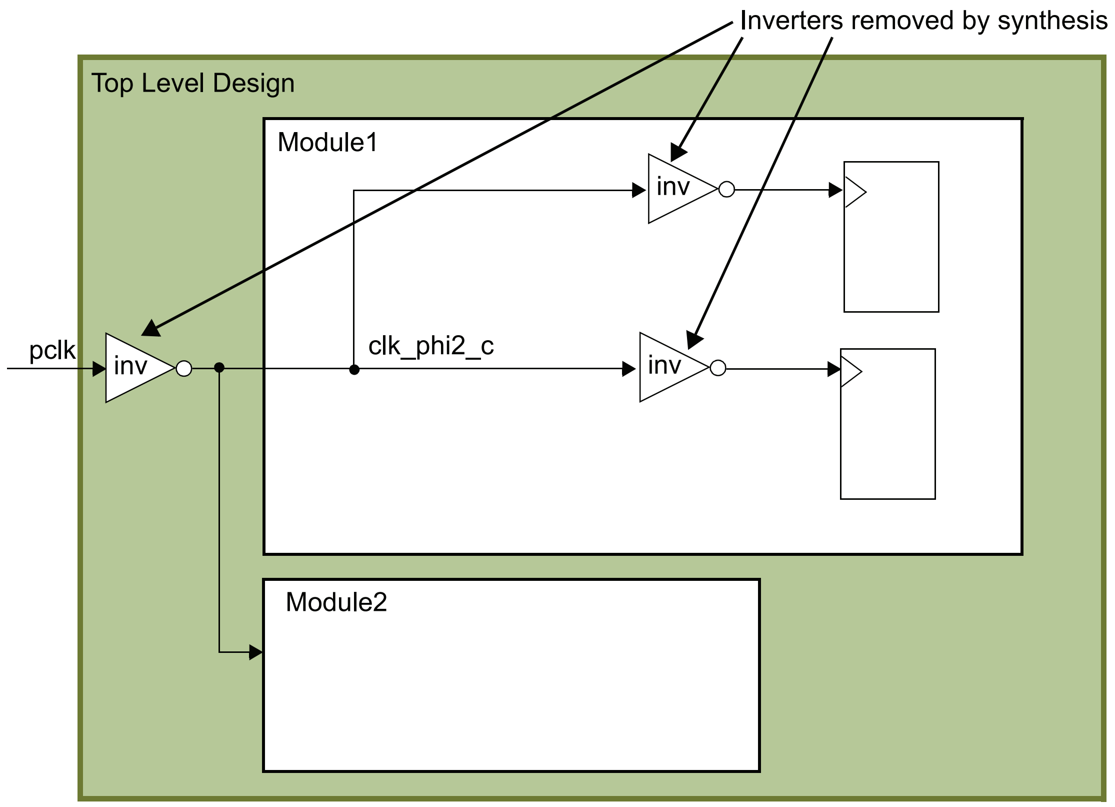


FIGURE 6: CLOCK DISTRIBUTION (NO CLOCK GATING)

Viewing and Adjusting Clock Constraints

As described in the *XPack Guide for AMBA IP,* you specify the clock period constraints in the CONSTRAINTS view of your GPIO workspace on the IPextreme IP Distribution and Support Portal. You can modify the clock period constraints for the following input clock signal:

- *pclk* – APB clock and the main GPIO system clock; used for registers and interrupt functionality

Reset

The GPIO uses a single asynchronous, active-low reset signal, *presetn*. Connect *presetn* as needed to your reset generation logic.

Assertion of *presetn* asynchronously resets most, but not all, flip-flops in the GPIO. The Port Data Output (PxDOUT) register does not have an asynchronous reset (from any reset source) and must be initialized by software. Refer to the *General Purpose I/O Controller User Guide* for additional register-related information.

Chapter 5
Interface Connections and Timing

This chapter describes the connections and timing of the GPIO top-level interfaces:

AMBA APB Interface

The GPIO is an AMBA 2 APB slave with 16 or 32-bit read and write data busses (depending on configuration parameter GPIO_CH) and a 10-bit address bus. The GPIO APB slave is little endian.

The SFR addresses of the GPIO are always 32-bit aligned. The size of the GPIO SFRs are 16 or 32 bits. As all SFRs are 32-bit aligned, *paddr[1:0]* are not used within the GPIO. Upon access to a non-32-bit-aligned boundary, the GPIO ignores *paddr[1:0]* and uses a value of ‘00’. Software must be aware of this behavior to ensure that the correct byte lanes are used in the event that a non-32-bit-aligned address is used to access a 16-bit or 8-bit SFR.

Byte accesses are not supported, as the AMBA 2 APB does not support byte selects.

GPIO_CH = 32

As a 32-bit SFR access example, a host access to any offset address from 0x00–0x04 results in an access to offset 0x00 (the 32-bit PxALT register if GPIO_CH = 32) and the GPIO accepts or returns the PxALT register data on *pwdata[31:0]* or *prdata[31:0]*.

Byte and halfword (8-bit and 16-bit) accesses are not supported. That is, byte accesses to the upper bytes (*paddr[0]* = ‘1’) and halfword accesses to the upper halfword (*paddr[1:0]* = ‘10’) are not supported, as the AMBA 2 APB does not support byte selects.

GPIO_CH = 16

As a 16-bit SFR access example, a host access to any offset address from 0x00–0x04 results in an access to offset 0x00 (the 16-bit PxALT register if GPIO_CH = 16) and the GPIO accepts or returns the PxALT register data on *pwdata[15:0]* or *prdata[15:0]*.

If the National Semiconductor AHB-to-APB Bridge is used to connect the GPIO to a 32-bit AHB data bus, a 32-bit access will return zeros in the unused upper 16 bits. A 32-bit write access to the upper 16 bits is ignored; only the lower 16 bits are written to the GPIO register.

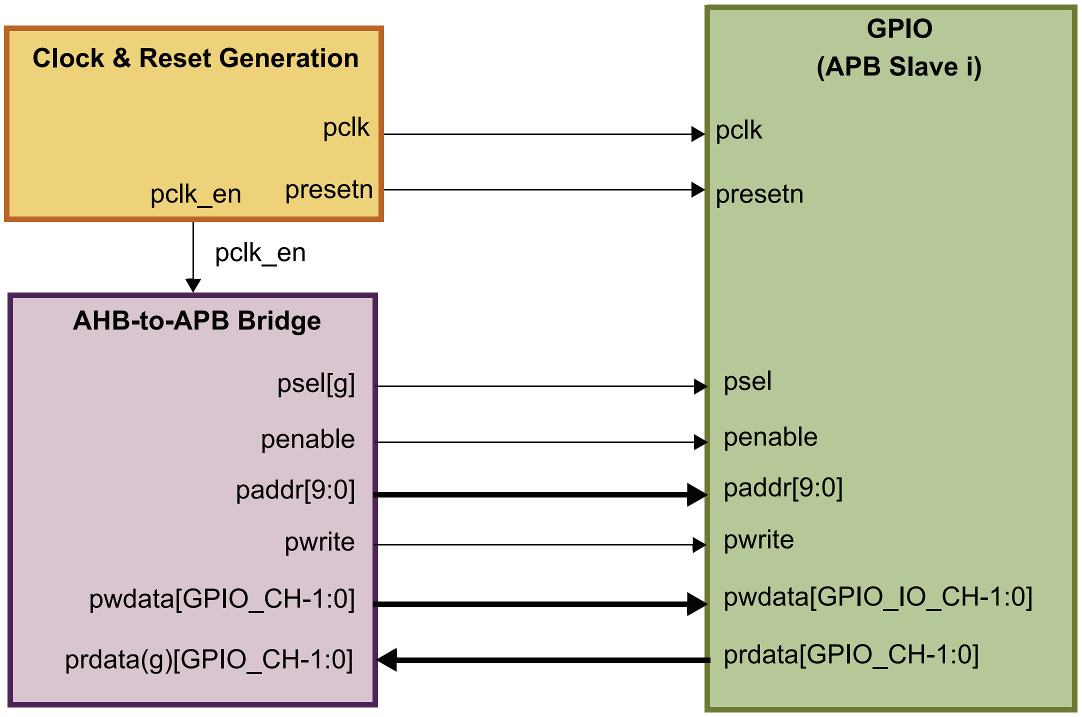
Example APB Connections

Table 5 defines the APB interface signals. Figure 7 shows example connections to an AMBA AHB-to-APB bridge.

For more details on the complete GPIO SFR memory map, refer to the *General Purpose I/O Controller User Guide*. For more details on AMBA APB protocol and timing diagrams, refer to Chapter 4 of the *AMBA Specification* (Rev 2) from ARM, Ltd.

TABLE 5: AMBA APB INTERFACE SIGNALS

Signal	I/O	Width	Function
paddr[9:0]	In	10	APB address bus
penable	In	1	APB enable – Indicates the second cycle of an AMBA APB transfer
psel	In	1	APB select – Indicates that the slave device is selected for a data transfer
pwdata [GPIO_CH-1:0]	In	GPIO_CH (16 or 32)	APB write data bus **Dependency**: Width of *pwdata* depends on parameter GPIO_CH.
pwrite	In	1	APB write enable: • 0: Read access • 1: Write access
prdata [GPIO_CH-1:0]	Out	GPIO_CH (16 or 32)	APB read data bus **Dependency**: Width of *prdata* depends on parameter GPIO_CH.



Note: In this example, the AHB-to-APB Bridge supports up to 16 APB slave ports. The General Purpose I/O Controller is connected to port g.

FIGURE 7: EXAMPLE APB INTERFACE CONNECTIONS

Pad Interface

The GPIO provides a bidirectional port with alternate function capability to an external off-chip interface. As an external interface, the GPIO is designed for direct connection to I/O pads. Figure 8 shows example connections from the GPIO to an I/O pad for a single I/O channel (channel i).

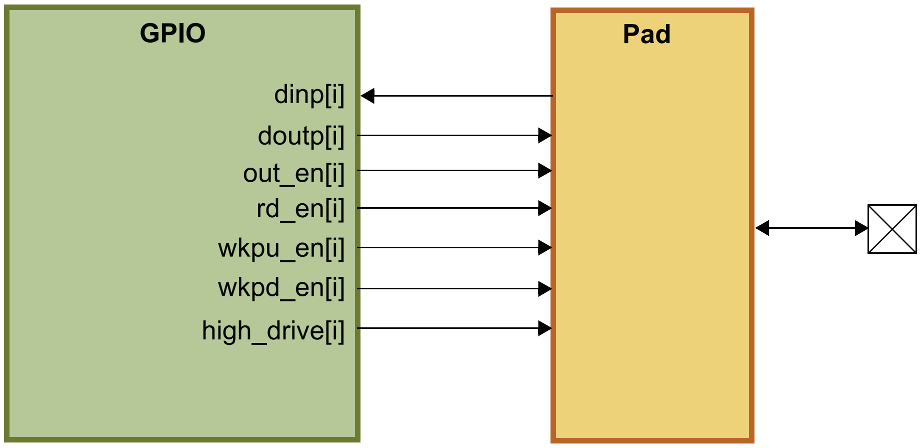


FIGURE 8: EXAMPLE CONNECTIONS TO I/O PAD FOR ONE CHANNEL

Peripheral Interface

The peripherals interface provides the capability to connect on-chip peripherals to the GPIO for use as alternate sources A and B. Control registers within the GPIO determine whether each I/O channel is used for general-purpose I/O or for an alternate function. When alternate function is selected for a channel, control registers also select the alternate source (A or B) for that channel.

Figure 9 shows example alternate source connections for I/O channel i. When alternate source A is selected, peripheral A controls the direction of and drives/receives data for channel i. When alternate source B is selected, peripheral B controls the direction of and drives/receives data for channel i.

In the case of Figure 9, each alternate source requires bi-directional functionality. The enable signals (*alt_a_en[i]* and *alt_a_en[i]*) are optional and available for peripherals that require an indication that their respective alternate source is selected.

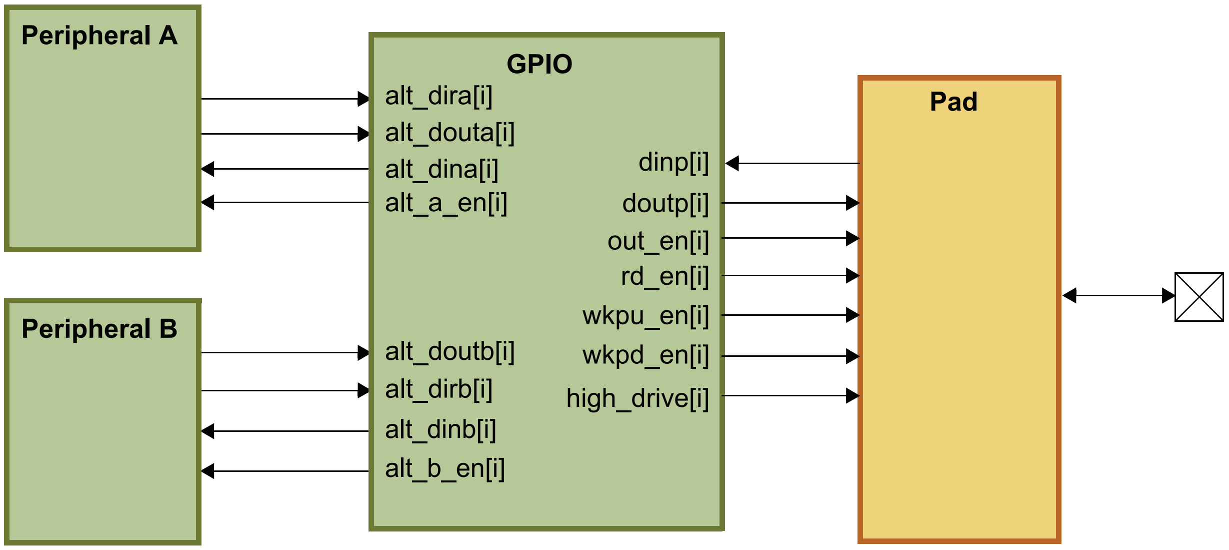


FIGURE 9: EXAMPLE ALTERNATE SOURCE CONNECTIONS FOR ONE CHANNEL

Figure 10 shows a similar set of connections for the case where peripheral A requires only input signal functionality and does not require an enable signal.

Figure 11 shows a similar set of connections for the case where peripheral A requires only output signal functionality and does not require an enable signal.

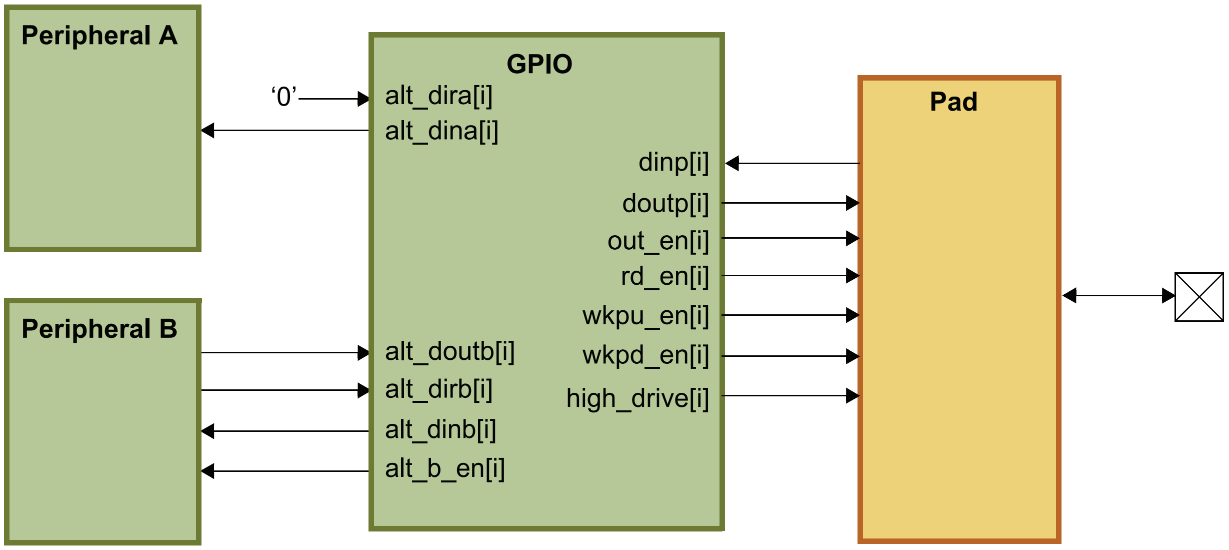


FIGURE 10: EXAMPLE ALTERNATE SOURCE CONNECTIONS FOR INPUT-ONLY PERIPHERAL

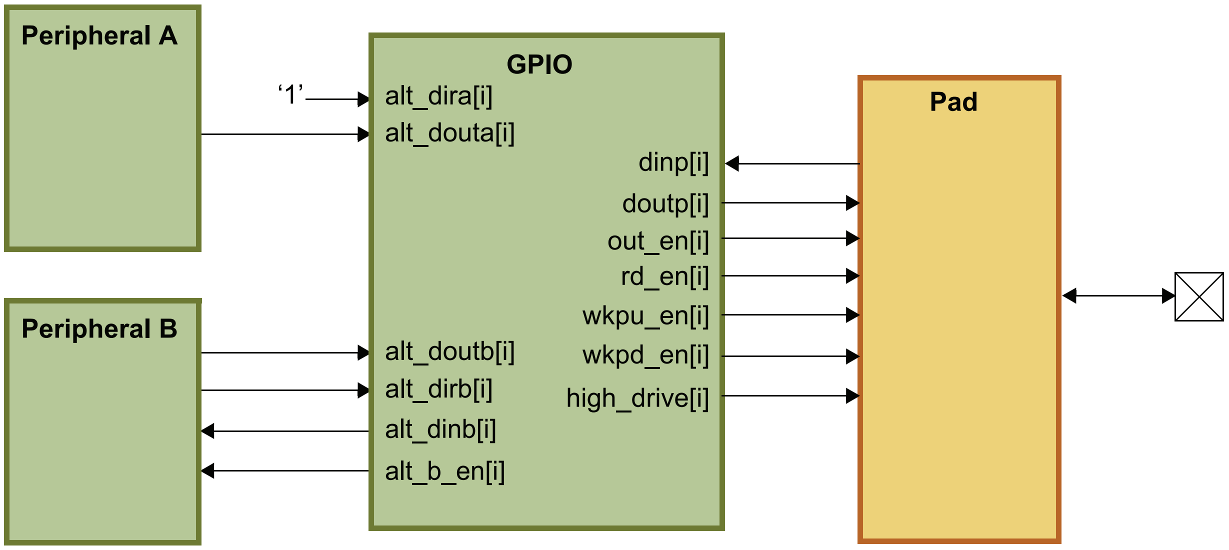


FIGURE 11: EXAMPLE ALTERNATE SOURCE CONNECTIONS FOR OUTPUT-C PERIPHERAL

Functional Test Mode Interface

Functional test mode enables control of the pad interface directly from user-implemented test logic. When input signal *tst* is asserted, pad interface signals for all I/O channels are controlled by the test logic, independent of the programming of the GPIO and the state of the alternate function peripherals. Figure 12 shows example functional test mode connections for channel i.

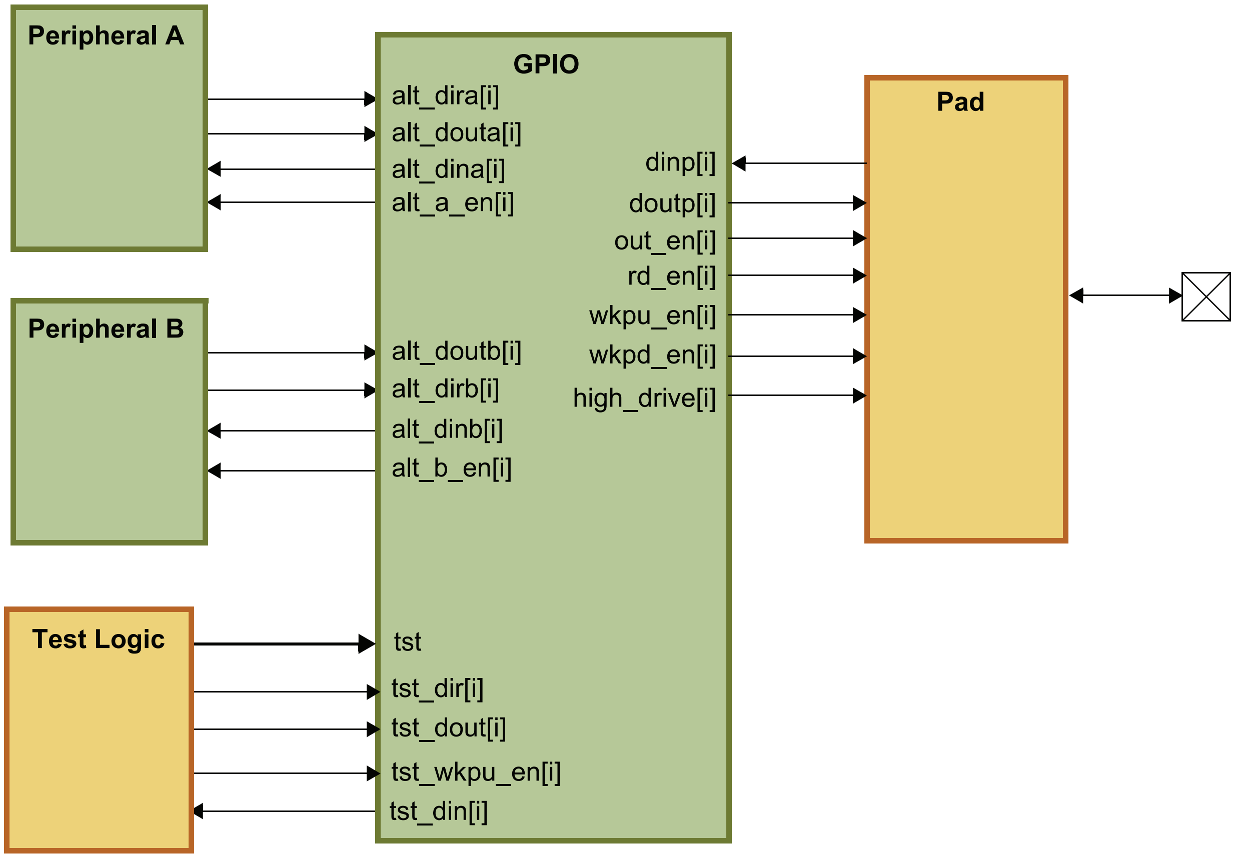


FIGURE 12: EXAMPLE CONNECTIONS FOR FUNCTIONAL TEST LOGIC

Interrupt Interface

The GPIO generates one interrupt signal, *gpio_int*, which is asserted (active high, level-sensitive) if one of the input pins *dinp* (enabled with register PxIEN) has the selected interrupt polarity (programmed in register PxDOUT) and the interrupt is enabled (PxIEN[i] set to '1').

Once asserted, *gpio_int* remains asserted until software clears the interrupt, as described as described in the *General Purpose I/O Controller User Guide*.

Figure 13 shows an example connection diagram of *gpio_int* to an interrupt controller. Figure 14 shows the *gpio_int* timing (level-sensitive interrupt).

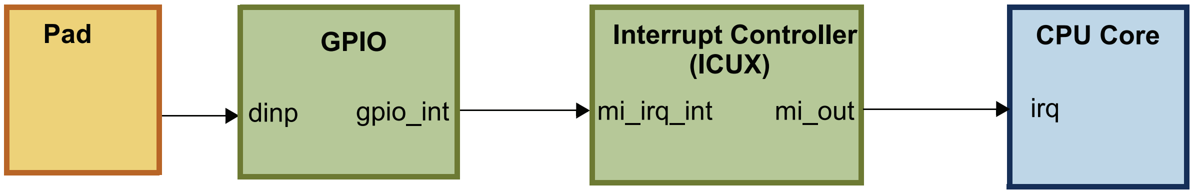


FIGURE 13: EXAMPLE INTERRUPT CONNECTION

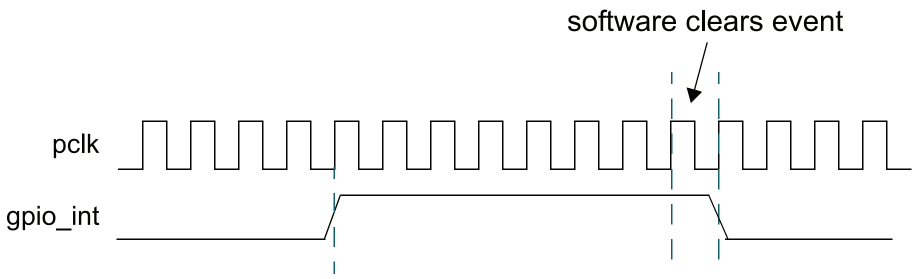


FIGURE 14: INTERRUPT TIMING

Wakeup Interface

The GPIO *dinp* input can be used as a wakeup input source to the Multi-Input Wakeup Input.

Figure 15 shows an example connection diagram of signal *dinp* from the pad logic to the Multi-Input Wakeup Unit. The GPIO is used to control the direction of the pad of the wakeup functionality. To set the pad to input, use either of the following methods:

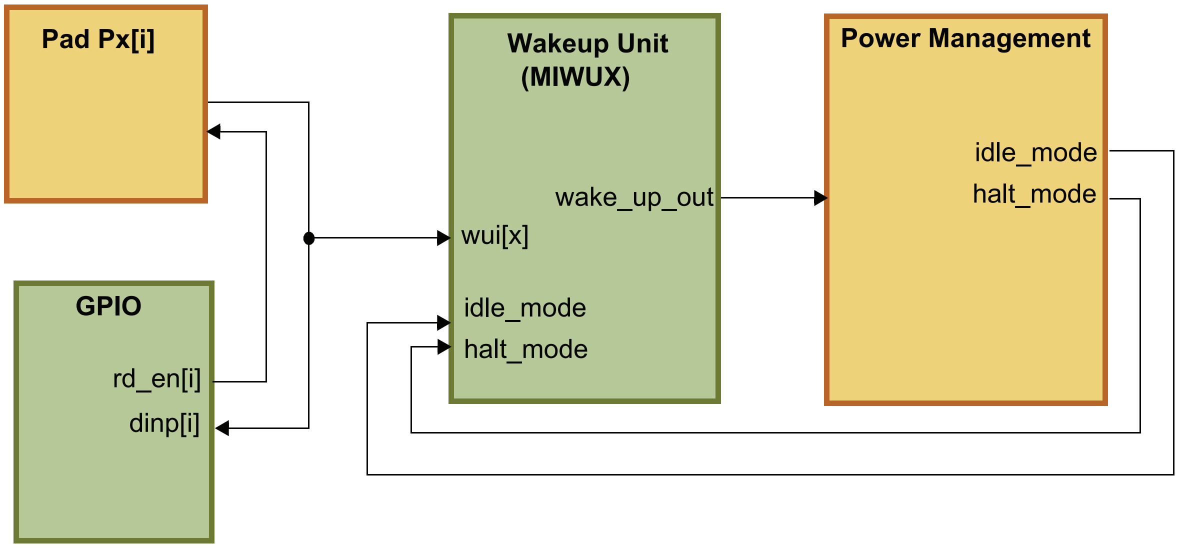
- Set PxALT[i] to ‘0’ (disable alternate function), set PXDIR[i] to ‘0’ (select input direction), and set PxIEN[i] to ‘1’ (enable read-path of GPIO)

  i is number of the GPIO channel that is used as wakeup input. Refer to the description of register PxIEN in the *General Purpose I/O Controller User Guide*.

  or

- Select an alternate function (by programming PxALT[i]) that is configured as an input. An alternate function that is configured as an output will not enable the read-path.

NOTE: In this case (wakeup interface required), the interrupt output *gpio_int* of the GPIO should be left unconnected.



In this example, GPIO channel i from PAD Px[i] is connected to wakeup channel x.

FIGURE 15: EXAMPLE WAKEUP CONNECTION

Chapter 6

Integration Testbench

The GPIO product includes an integration testbench that demonstrates how to integrate the GPIO into a system testbench, check the top-level connections, and stimulate the top-level interfaces. A set of test programs exercise various functions and interfaces.

The integration testbench is not intended to serve as a full functional verification environment. Rather, it is intended to demonstrate the functional behavior of the GPIO and its top-level interfaces. You can use the integration testbench as an example to integrate the GPIO into your system functional verification environment.

This chapter describes the integration testbench. The topics are:

- Integration Testbench Architecture
- File List
- Test List
- Testbench Configuration
- Checking Simulation Results

For information about running simulations with the GPIO integration testbench, or any other component in the National Semiconductor IP Library for AMBA Interconnect, refer to the *XPack Guide for AMBA IP*.

Integration Testbench Architecture

Figure 16 shows the structure of the integration testbench. The purpose of the integration testbench is to provide some application examples on how to integrate the product into an SoC. Therefore, it contains clock and reset generators, verification components, and models to stimulate all interfaces. Test programs to initialize, configure, and demonstrate some of the key functionality of the product are provided to exercise the integration testbench.

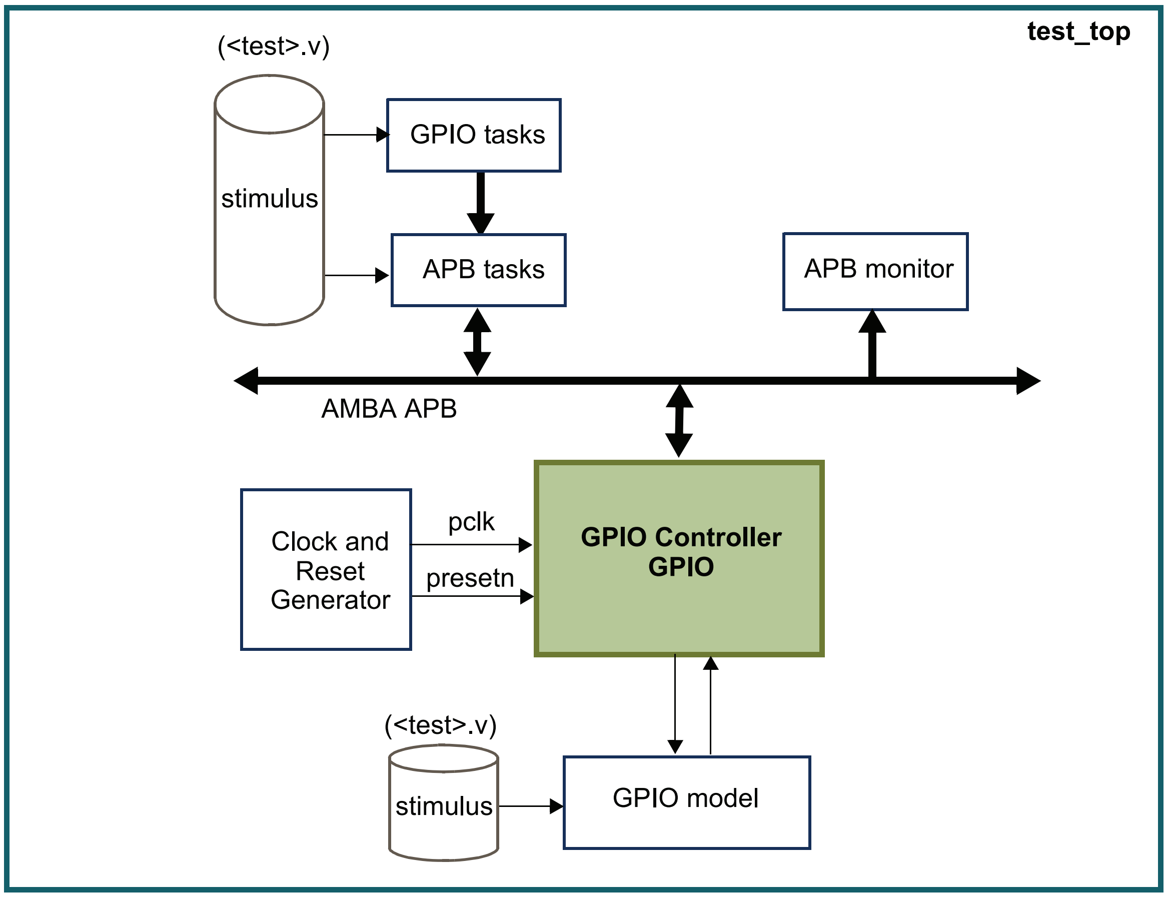


FIGURE 16: INTEGRATION TESTBENCH BLOCK DIAGRAM

File List

Table 6 lists the integration testbench models; filenames are relative to <install_dir>/sim.

TABLE 6: TESTBENCH MODELS

File Name	Description
tb/tb_ck_gpio_xt_geh0.v	Top-level integration testbench named test_top
tb/apb_tasks.v	Low-level APB write/read task; included by tb_ck_gpio_xt_geh0
tb/ck_gpio_xt_tasks.v	Special tasks for the GPIO; included by tb_ck_gpio_xt_geh0
test/tp-<test>/<test>.v	Testcase-specific Verilog file containing the stimulus for the APB using the tasks from ck_gpio_xt_task and apb_task; included by tb_ck_gpio_xt_geh0
tb/apb_slave_timing.inc	APB testbench timing information; included by tb_ck_gpio_xt_geh0
tb/apb_slave_monitor.v	AMBA APB monitor; instantiated in tb_ck_gpio_xt_geh0

Test List

Table 7 describes the available test programs. There is one test-specific file for each test (<install_dir>/sim/test/tp-<test>/<test>.v), which contains the stimulus for the APB.

TABLE 7: GPIO INTEGRATION TESTS

Test	Description
tp-gpio_func	Demonstrates GPIO functions: • Path *dinp* to APB • Path *dinp* to *tst_din* • Path *dinp* to *alt_dina* • Path *dinp* to *alt_dinb* • Path *dinp* to *gpio_int* • Data direction for all combinations • Data output for all combinations • Weak pull-up in normal and test mode • Read enable inactive • Read enable active for all input combinations • High drive • *alt_a_en*, *alt_b_en* • *gpio_int*
tp-gpio_rdwr_regs	Demonstrates register access: • Reset values • apb_write and apb_read 8’h55 to all registers • apb_write and apb_read 8’hAA to all registers • apb_write and apb_read unique pattern • Set all registers to opposite reset values and perform system reset • Recheck reset values

Testbench Configuration

The GPIO configuration used for simulation is independent of the GPIO configuration that you choose on the IPextreme IP Distribution and Support Portal for synthesis. Your synthesis configuration is defined in the file:

<install_dir>/impl/cfg-<myconfig/src/ck_gpio_xt_geh0_params.v

whereas the configuration used for simulation is determined by the parameter settings in the file:

<install_dir>/src/gpio/ck_gpio_xt_geh0_params.v

Checking Simulation Results

Running a simulation creates a simulation log file (sim.log) in your <install_dir>/sim/log/tp-<test> directory. sim.log contains:

- Simulation startup and execution messages
- Pass/fail status
- Testbench tasks and time span in which the testbench tasks are executed

If the simulation fails, check sim.log for error messages.

In addition, if you create a dump file from simulation at the testbench level, you can observe the signalling in the testbench through your waveform viewer. The signals of interest are at the GPIO top level (instance name ‘dut’ within in the top level testbench):

- APB interface
- Interrupt interface
- Pad interface
- Peripheral interface

CHAPTER 7
TESTABILITY

DFT Signals

The *scan_clken, scan_test,* and *scan_reset* signals are for test mode and control testability of the GPIO logic. During scan test mode (when *scan_test* is asserted):

- The local reset is controlled by *scan_reset* (not by *presetn*).
- All flips-flops are clocked by system clock *pclk*.

TABLE 8: DFT SIGNALS

Signal	I/O	Width	Comment
scan_clken	In	1	Scan test NAND clock enable, used only if clock gating is enabled in the GPIO design: • If clock gating is enabled (GATED_CLOCK = true), connect *scan_clken* to the chip's scan_shift/scan_enable signal. • If GATED_CLOCK = false, *scan_clken* is tied to '0' internally.
scan_test	In	1	Scan test enable. Assign to '1' during test.
scan_reset	In	1	Scan reset