RTL Design Sherpa

APB GPIO Micro-Architecture Specification 1.0

January 4, 2026

Table of Contents

List of Figures

List of Tables

List of Waveforms

No waveforms in this document.

1 Gpio Mas Index

Generated: 2026-01-04

RTL Design Sherpa · Learning Hardware Design Through Practice GitHub · Documentation Index · MIT License

---

2 APB GPIO - Overview

2.1 Introduction

The APB GPIO controller provides a 32-bit general-purpose I/O interface with APB bus connectivity. It enables software-controlled digital I/O with flexible interrupt generation capabilities.

2.2 Features

2.2.1 Core Functionality

- 32-bit bidirectional GPIO port
- Per-bit direction control (input/output)
- Per-bit output enable control
- Input synchronization for metastability protection

2.2.2 Interrupt Capabilities

- Per-bit interrupt enable
- Edge-triggered interrupts (rising, falling, or both)
- Level-triggered interrupts (high or low)
- Combined interrupt output (OR of all enabled sources)
- Write-1-to-clear interrupt status

2.2.3 Atomic Operations

- Atomic set (OR with mask)
- Atomic clear (AND with inverted mask)
- Atomic toggle (XOR with mask)
- No read-modify-write race conditions

2.2.4 Clock Domain Crossing

- Optional CDC support via `CDC_ENABLE` parameter
- Separate GPIO clock domain for async I/O
- Multi-stage input synchronization

2.3 Applications

2.3.1 Typical Use Cases

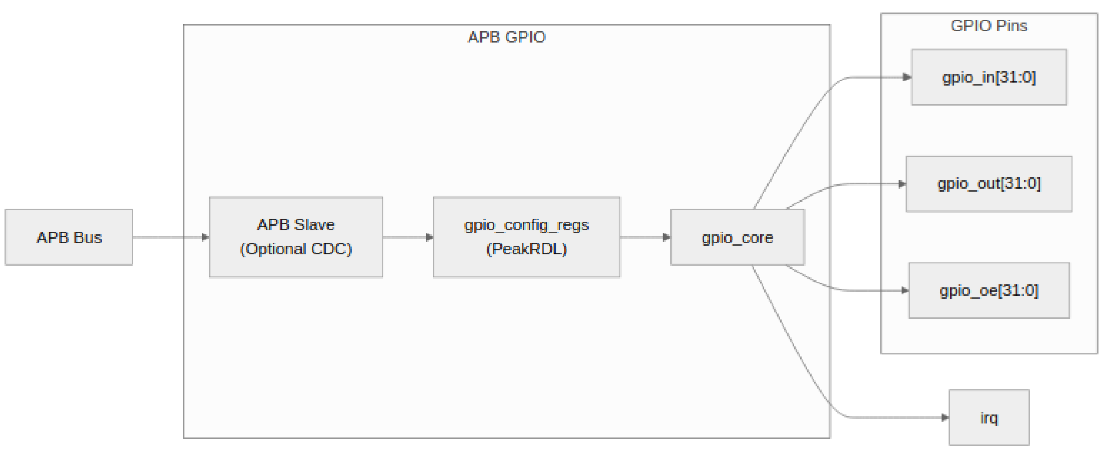
- LED control and status indication
- Push-button and switch inputs
- External device reset control
- Interrupt generation from external events
- Bit-banged serial protocols (I2C, SPI fallback)
- Debug signals and test points

2.3.2 System Integration

- Memory-mapped APB peripheral
- Single interrupt line to CPU/interrupt controller
- Direct connection to FPGA I/O pads via IOBUFs
- Compatible with standard GPIO software drivers

2.4 Block Diagram

2.4.1 Figure 1.1: APB GPIO Block Diagram



APB GPIO Block Diagram

2.5 Key Specifications

Parameter	Value
GPIO Width	32 bits (configurable)
APB Data Width	32 bits
APB Address Width	12 bits (4KB)
Sync Stages	2 (configurable)
CDC Support	Optional

2.6 Register Summary

Address	Register	Description
0x000	GPIO_CONTROL	Global enable and interrupt enable
0x004	GPIO_DIRECTION	Per-bit direction (1=output)
0x008	GPIO_OUTPUT	Output data value
0x00C	GPIO_INPUT	Input data (read-only)
0x010	GPIO_INT_ENABLE	Per-bit interrupt enable
0x014	GPIO_INT_TYPE	Interrupt type (1=level, 0=edge)
0x018	GPIO_INT_POLARITY	Polarity (1=high/rising)
0x01C	GPIO_INT_BOTH	Both-edge enable
0x020	GPIO_INT_STATUS	Interrupt status (W1C)
0x024	GPIO_RAW_INT	Raw interrupt (pre-mask)
0x028	GPIO_OUTPUT_SET	Atomic set
0x02C	GPIO_OUTPUT_CLR	Atomic clear
0x030	GPIO_OUTPUT_TGL	Atomic toggle

Next: 02_architecture.md - High-level architecture

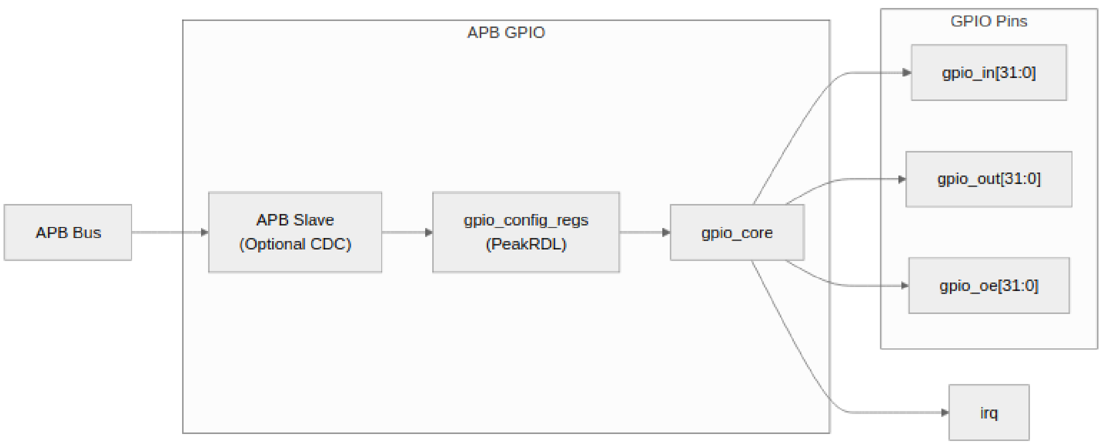
RTL Design Sherpa · Learning Hardware Design Through Practice GitHub · Documentation Index · MIT License

---

3 APB GPIO - Architecture

3.1 High-Level Block Diagram

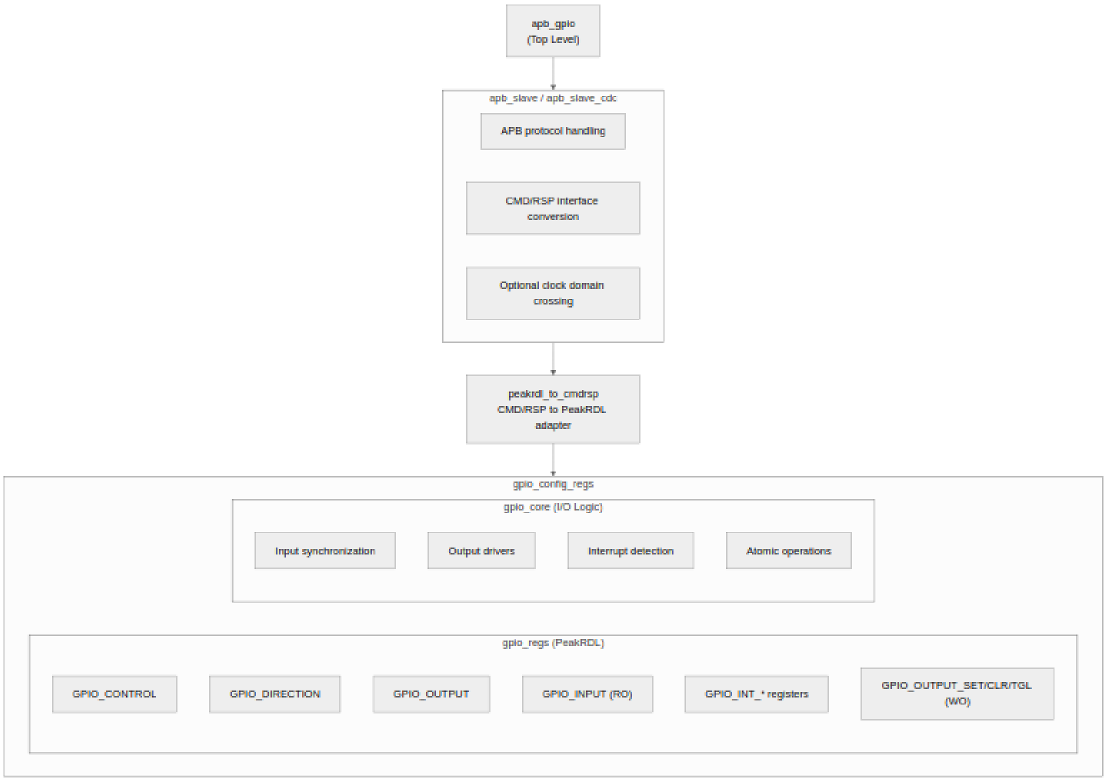
3.1.1 Figure 1.2: APB GPIO Top-Level Architecture



APB GPIO Architecture

3.2 Module Hierarchy

3.2.1 Figure 1.3: APB GPIO Module Hierarchy

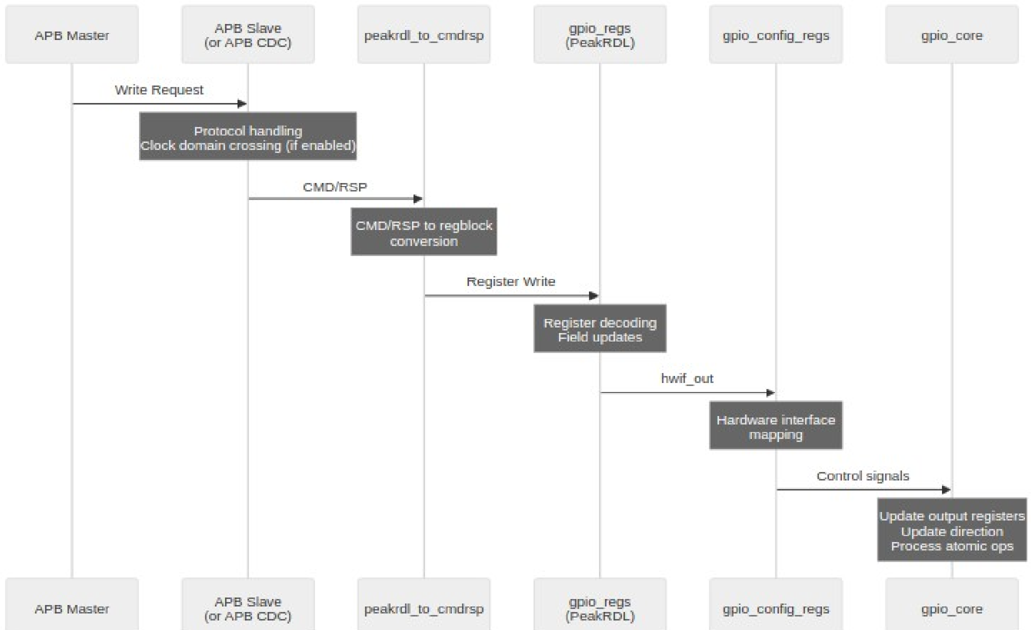


APB GPIO Module Hierarchy

3.3 Data Flow

3.3.1 Write Transaction Flow

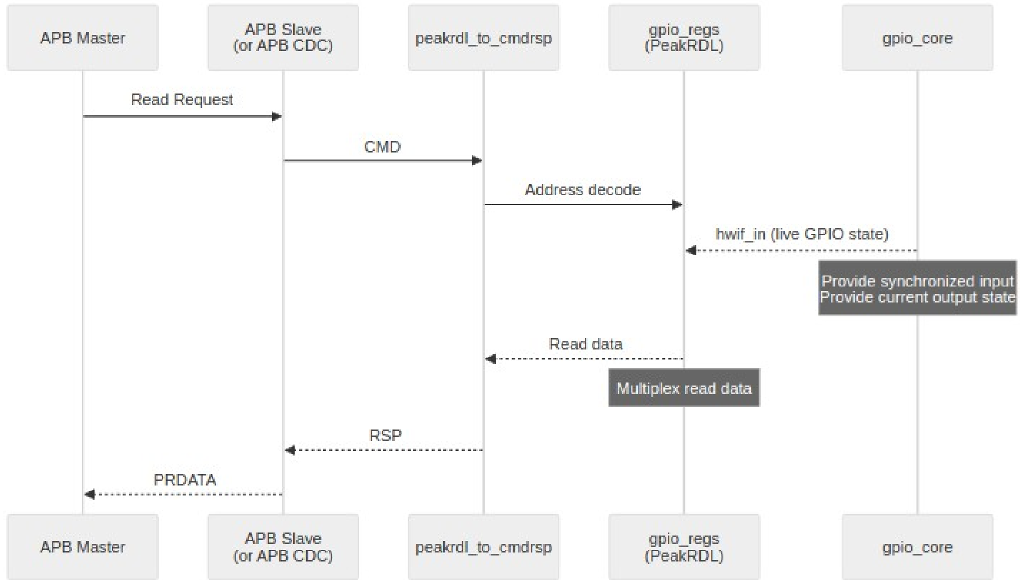
3.3.2 Figure 1.4: Write Transaction Flow



Write Transaction Flow

3.3.3 Read Transaction Flow

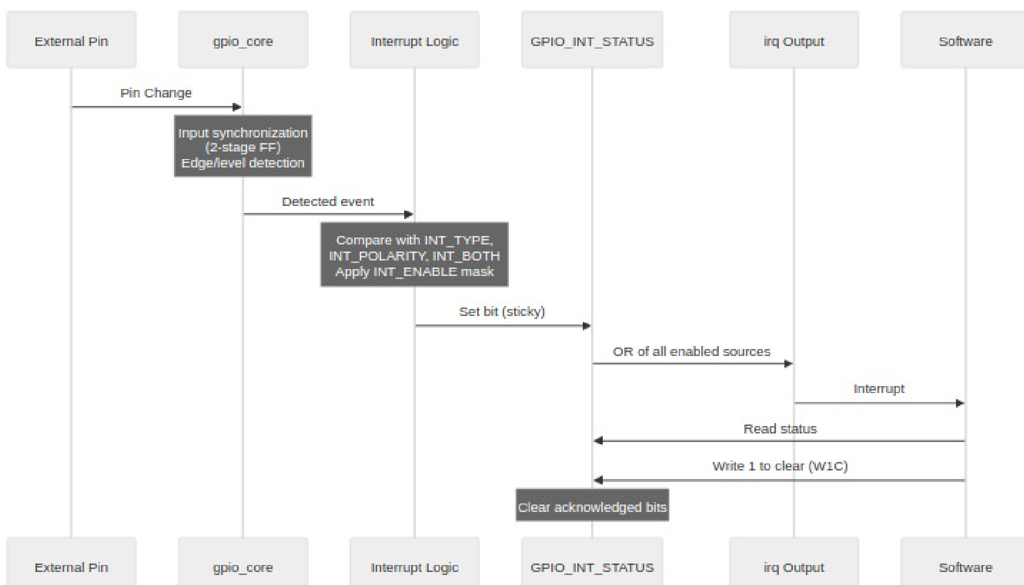
3.3.4 Figure 1.5: Read Transaction Flow



Read Transaction Flow

3.3.5 Interrupt Flow

3.3.6 Figure 1.6: Interrupt Flow

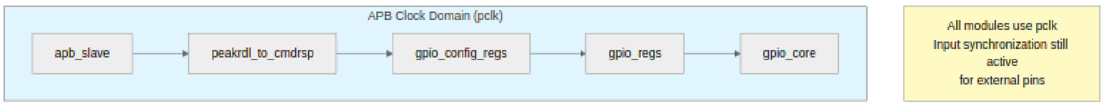


Interrupt Flow

3.4 Clock Domains

3.4.1 Synchronous Mode (CDC_ENABLE = 0)

3.4.2 Figure 1.7: Synchronous Mode Clock Domains

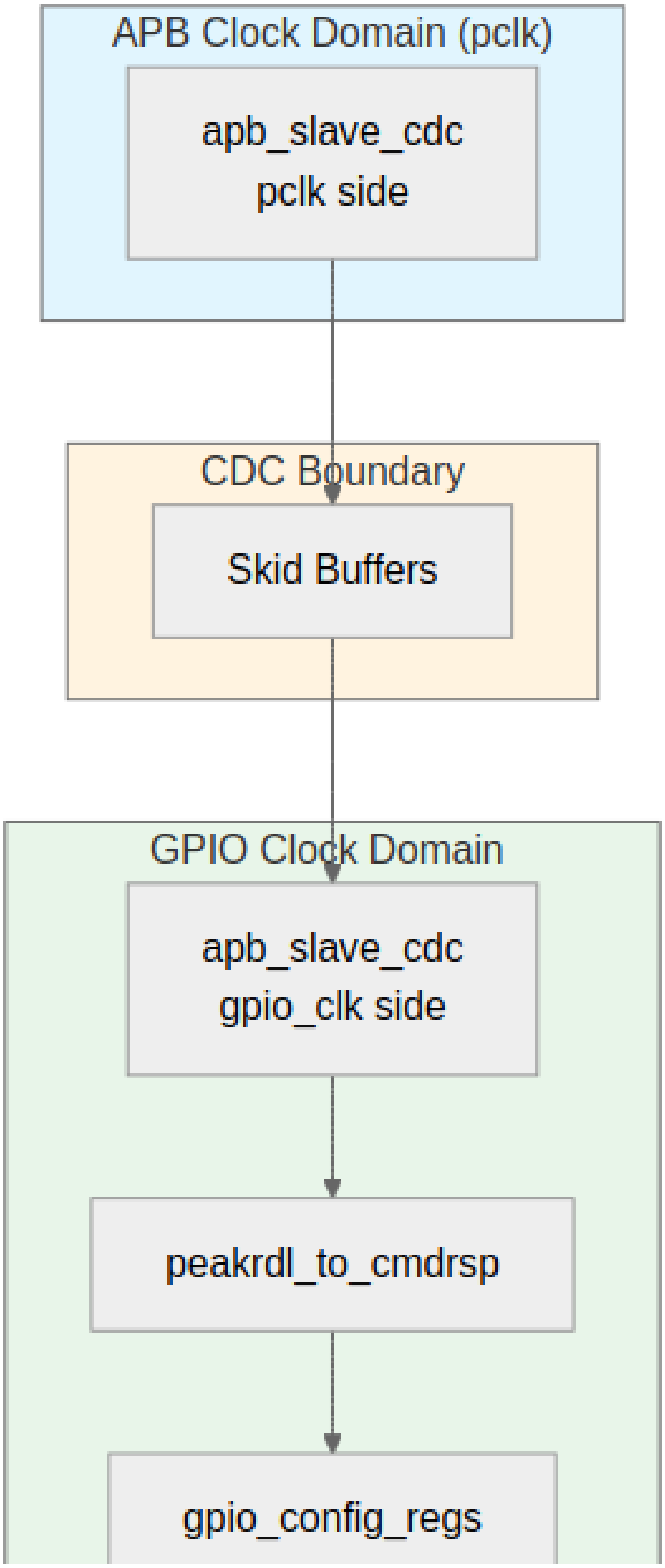


Synchronous Mode

In synchronous mode, all modules operate on the APB clock (pclk). Input synchronization remains active for external GPIO pins to prevent metastability.

3.4.3 Asynchronous Mode (CDC_ENABLE = 1)

3.4.4 Figure 1.8: Asynchronous Mode Clock Domains


APB Clock Domain (pclk)
apb_slave_cdc
pclk side
CDC Boundary
Skid Buffers
GPIO Clock Domain
apb_slave_cdc
gpio_clk side
peakrdl_to_cmdrsp
gpio_config_regs

Asynchronous Mode

In asynchronous mode, the APB clock domain handles protocol conversion while the GPIO clock domain handles all register and I/O operations. Skid buffers provide safe clock domain crossing.

3.5 Parameterization

Table 1.1: GPIO Parameters

Parameter	Type	Default	Description
`GPIO_WIDTH`	int	32	Number of GPIO pins
`SYNC_STAGES`	int	2	Input synchronizer stages
`CDC_ENABLE`	int	0	Enable clock domain crossing
`SKID_DEPTH`	int	2	CDC skid buffer depth

3.6 Resource Estimates

Table 1.2: Resource Estimates

Component	Flip-Flops	LUTs
gpio_core	~200	~300
gpio_regs	~400	~200
gpio_config_regs	~50	~100
apb_slave (no CDC)	~20	~50
apb_slave_cdc	~100	~150
Total (no CDC)	~670	~650
Total (with CDC)	~750	~750

Next: 03_clocks_and_reset.md - Clock and reset behavior

4 APB GPIO - Clocks and Reset

4.1 Clock Signals

4.1.1 pclk (APB Clock)

- **Purpose:** Primary APB bus clock
- **Usage:** APB protocol, register access
- **Typical Frequency:** 50-200 MHz

4.1.2 gpio_clk (GPIO Clock)

- **Purpose:** Optional separate GPIO clock domain
- **Usage:** Only when `CDC_ENABLE=1`
- **Relationship:** Can be asynchronous to pclk

4.2 Reset Signals

4.2.1 presetn (APB Reset)

- **Type:** Active-low asynchronous reset
- **Scope:** APB interface logic
- **Behavior:** Resets APB state machine, clears pending transactions

4.2.2 gpio_rstn (GPIO Reset)

- **Type:** Active-low asynchronous reset
- **Scope:** GPIO core logic
- **Usage:** Only when `CDC_ENABLE=1`
- **Behavior:** Resets GPIO outputs, interrupt state

4.3 Reset Behavior

4.3.1 Register Reset Values

Register	Reset Value	Notes
GPIO_CONTROL	0x00000000	GPIO disabled

Register	Reset Value	Notes
GPIO_DIRECTION	0x00000000	All inputs
GPIO_OUTPUT	0x00000000	Outputs low
GPIO_INT_ENABLE	0x00000000	No interrupts
GPIO_INT_TYPE	0x00000000	Edge mode
GPIO_INT_POLARITY	0x00000000	Falling/low
GPIO_INT_BOTH	0x00000000	Single edge
GPIO_INT_STATUS	0x00000000	No pending

4.3.2 Output Pin Behavior During Reset

During reset: - `gpio_out[31:0]` = 0 - `gpio_oe[31:0]` = 0 (all high-Z) - `irq` = 0

4.4 Clock Domain Crossing

4.4.1 When CDC_ENABLE = 0

- All logic runs on `pclk`
- `gpio_clk` input is ignored
- Connect `gpio_clk = pclk` for clean design

4.4.2 When CDC_ENABLE = 1

- APB interface uses `pclk`
- GPIO core uses `gpio_clk`
- Skid buffers handle CDC
- Both resets must be asserted together at power-on

4.5 Input Synchronization

GPIO inputs are always synchronized regardless of CDC setting:

```
gpio_in[i] --> FF1 --> FF2 --> synchronized_input[i]
                (clk)   (clk)
```

- SYNC_STAGES parameter controls depth (default: 2)
- Prevents metastability from external signal transitions
- Adds latency equal to SYNC_STAGES clock cycles

4.6 Timing Constraints

4.6.1 Synchronous Mode

- Standard single-clock timing
- All paths constrained to pclk

4.6.2 Asynchronous Mode

- Set false_path between pclk and gpio_clk domains
- Set max_delay for CDC paths
- Synchronizer FFs should have ASYNC_REG attribute

Next: 04_acronyms.md - Acronyms and terminology

---

5 APB GPIO - Acronyms and Terminology

5.1 Protocol Acronyms

Acronym	Full Name	Description
APB	Advanced Peripheral Bus	Low-power AMBA bus protocol
AMBA	Advanced Microcontroller Bus Architecture	ARM standard bus protocols
CDC	Clock Domain Crossing	Synchronization between clock domains

5.2 Signal Acronyms

Acronym	Full Name	Description
CLK	Clock	Timing reference signal

Acronym	Full Name	Description
RST	Reset	System initialization signal
OE	Output Enable	Tri-state buffer control
IRQ	Interrupt Request	Hardware interrupt signal

5.3 GPIO-Specific Terms

Term	Description
GPIO	General Purpose Input/Output - Configurable digital I/O pins
Pin	Individual GPIO signal (input or output)
Port	Group of 32 GPIO pins managed together
Direction	Input (0) or Output (1) configuration per pin
Polarity	Active-high or active-low signal interpretation

5.4 Interrupt Terms

Term	Description
Edge-triggered	Interrupt on signal transition
Level-sensitive	Interrupt while signal is at specified level
Rising edge	Low-to-high transition
Falling edge	High-to-low transition
Both edges	Either transition direction

5.5 Register Terms

Term	Description
RW	Read-Write register

Term	Description
RO	Read-Only register
W1C	Write-1-to-Clear register
HWIF	Hardware Interface (PeakRDL generated)

Next: 05_references.md - Reference documents

6 APB GPIO - References

6.1 Internal Documentation

6.1.1 RTL Source Files

- `rtl/gpio/apb_gpio.sv` - Main GPIO module
- `rtl/gpio/apb_gpio_regs.sv` - PeakRDL-generated register file
- `rtl/gpio/apb_gpio.rdl` - Register description source

6.1.2 Related Specifications

- APB Protocol Specification (AMBA 3)
- RLB Integration Guide

6.2 External References

6.2.1 ARM AMBA Specifications

- **AMBA 3 APB Protocol Specification**
  - ARM IHI 0024E
  - Defines APB interface timing and protocol

6.2.2 Industry Standards

- **GPIO Best Practices**

- Synchronization for external inputs
- Interrupt handling patterns
- Tri-state buffer management

6.3 Design References

6.3.1 Clock Domain Crossing

- Dual flip-flop synchronizer methodology
- Skid buffer for data path CDC
- Reset synchronization techniques

6.3.2 Interrupt Handling

- Edge detection circuits
- Interrupt aggregation patterns
- Software interrupt acknowledge flows

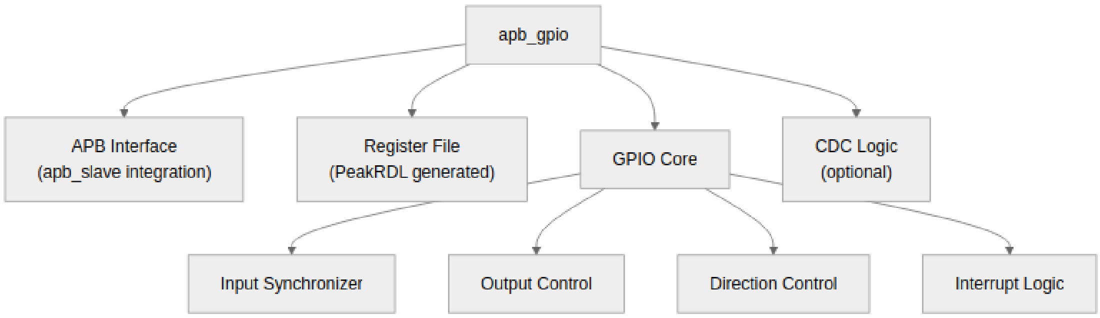
Next: Chapter 2: Block Descriptions

---

7 APB GPIO - Block Descriptions Overview

7.1 Module Hierarchy

7.1.1 Figure 2.1: APB GPIO Module Hierarchy



Module Hierarchy

7.2 Block Summary

Table 2.1: Block Summary

Block	File	Description
APB GPIO Top	apb_gpio.sv	Top-level module with all GPIO functionality
Register File	apb_gpio_regs.sv	PeakRDL-generated control/status registers

7.3 Detailed Block Descriptions

7.3.1 1. APB Interface

Handles APB protocol conversion and register access.

See: 01_apb_interface.md

7.3.2 2. Register File

PeakRDL-generated registers for configuration and status.

See: 02_register_file.md

7.3.3 3. GPIO Core

Main GPIO functionality including I/O control and interrupts.

See: 03_gpio_core.md

7.3.4 4. Interrupt Controller

Edge detection, level sensing, and interrupt aggregation.

See: 04_interrupt_controller.md

7.3.5 5. CDC Logic

Optional clock domain crossing for asynchronous GPIO clock.

See: 05_cdc_logic.md

Next: 01_apb_interface.md - APB Interface details

RTL Design Sherpa · Learning Hardware Design Through Practice GitHub · Documentation Index · MIT License

---

8 APB GPIO - APB Interface Block

8.1 Overview

The APB interface provides the connection between the system APB bus and the GPIO register file.

8.2 Block Diagram

APB Interface Block

APB Interface Block

8.3 Interface Signals

8.3.1 APB Slave Interface

Signal	Width	Direction	Description
s_apb_psel	1	Input	Slave select
s_apb_penable	1	Input	Enable phase
s_apb_pwrite	1	Input	Write

Signal	Width	Direction	Description
			operation
s_apb_paddr	12	Input	Address bus
s_apb_pwdata	32	Input	Write data
s_apb_pstrb	4	Input	Byte strobes
s_apb_prdata	32	Output	Read data
s_apb_pready	1	Output	Ready response
s_apb_pslverr	1	Output	Error response

8.4 Operation

8.4.1 Read Transaction

1. Master asserts `psel` and `paddr`
2. Master asserts `penable` on next cycle
3. Slave returns `prdata` with `pready`

8.4.2 Write Transaction

1. Master asserts `psel`, `paddr`, `pwdata`, `pwrite`
2. Master asserts `penable` on next cycle
3. Slave samples data with `pready`

8.5 Timing Diagram

```
         _____       _____       _____
pclk  __/     \_____/     \_____/     \_____
         _________________
psel  __/                 \_________________
                   _______
penable __________|       |_________________

paddr   XXXXXXXXXX|  ADDR |XXXXXXXXXXXXXXXXX

prdata  XXXXXXXXXX|  DATA |XXXXXXXXXXXXXXXXX
                   _______
pready  __________|       |_________________
```

8.6 Implementation Notes

- Zero wait-state operation for all registers
- No error responses (pslverr always 0)
- 32-bit aligned access only

Next: 02_register_file.md - Register File

---

9 APB GPIO - Register File Block

9.1 Overview

The register file is generated by PeakRDL from `apb_gpio.rdl` and provides hardware/software interface for GPIO control.

9.2 Block Diagram

Register File Block

Register File Block

9.3 Generated Interface (HWIF)

9.3.1 Hardware-to-Software (hw2reg)

Signal	Width	Description
gpio_input.data	32	Current GPIO input values
gpio_int_status.data	32	Interrupt status bits

9.3.2 Software-to-Hardware (reg2hw)

Signal	Width	Description
gpio_control.enable	1	GPIO enable
gpio_direction.data	32	Pin direction (0=in,

Signal	Width	Description
		1=out)
gpio_output.data	32	Output values
gpio_int_enable.data	32	Interrupt enable per pin
gpio_int_type.data	32	Interrupt type (0=edge, 1=level)
gpio_int_polarity.data	32	Polarity (0=fall/low, 1=rise/high)
gpio_int_both.data	32	Both edges enable

9.4 Register Access

9.4.1 Byte Enable Support

All registers support byte-granular writes via pstrb: - `pstrb[0]` enables bits [7:0] - `pstrb[1]` enables bits [15:8] - `pstrb[2]` enables bits [23:16] - `pstrb[3]` enables bits [31:24]

9.4.2 Access Types

Type	Read Behavior	Write Behavior
RW	Returns current value	Updates register
RO	Returns hardware value	No effect
W1C	Returns current value	Clears bits where 1 written

9.5 Implementation Notes

- Generated by PeakRDL regblock
- Synchronous to pclk domain
- All registers reset to 0

Next: 03_gpio_core.md - GPIO Core

10 APB GPIO - GPIO Core Block

10.1 Overview

The GPIO core handles input synchronization, output driving, and direction control for all 32 GPIO pins.

10.2 Block Diagram

GPIO Core Block

GPIO Core Block

10.3 Input Path

10.3.1 Synchronization

External inputs pass through a dual flip-flop synchronizer:

```
gpio_in[i] --> FF1 --> FF2 --> synced_input[i]
                (clk)   (clk)
```

- Prevents metastability from asynchronous inputs
- Configurable depth via `SYNC_STAGES` parameter
- Adds SYNC_STAGES clock cycles of latency

10.3.2 Input Register

Synchronized inputs are presented to software via `GPIO_INPUT` register.

10.4 Output Path

10.4.1 Output Register

Software writes to `GPIO_OUTPUT` register to set output values.

10.4.2 Output Enable

Direction register controls tri-state buffers: - `direction[i] = 0`: Pin is input (high-Z output) - `direction[i] = 1`: Pin is output (driven)

10.4.3 External Signals

Signal	Width	Description
gpio_out	32	Output data values
gpio_oe	32	Output enables (active high)
gpio_in	32	Input data values

10.5 Direction Control

10.5.1 Per-Pin Configuration

Each pin independently configured:

```
if (direction[i]) begin
    // Output mode
    gpio_oe[i] = 1'b1;
    gpio_out[i] = output_reg[i];
end else begin
    // Input mode
    gpio_oe[i] = 1'b0;
    // gpio_out[i] = don't care
end
```

10.5.2 Read-Back Behavior

Reading `GPIO_INPUT` returns: - For input pins: External signal value (synchronized) - For output pins: External signal value (may differ from output_reg if open-drain)

10.6 Implementation Notes

- All 32 pins processed in parallel
- Zero-latency output updates
- Input synchronization always active

Next: 04_interrupt_controller.md - Interrupt Controller

---

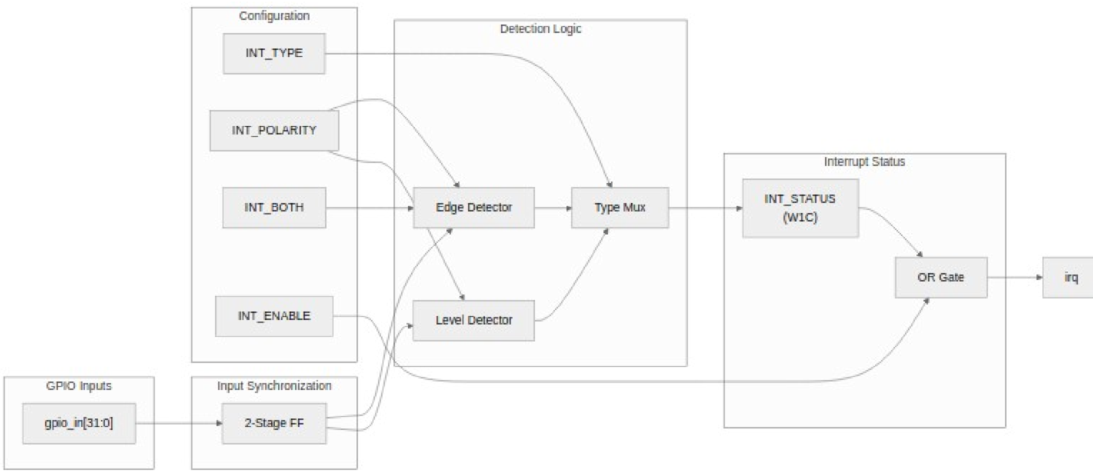
11 APB GPIO - Interrupt Controller Block

11.1 Overview

The interrupt controller provides flexible interrupt generation for each GPIO pin with support for edge and level triggering.

11.2 Block Diagram

11.2.1 Figure 2.5: Interrupt Controller Block Diagram



Interrupt Controller Block

11.3 Interrupt Modes

11.3.1 Edge-Triggered Mode

`GPIO_INT_TYPE[i] = 0`

Table 2.5: Edge-Triggered Modes

GPIO_INT_POLARITY	GPIO_INT_BOTH	Trigger Condition
0	0	Falling edge only
1	0	Rising edge only
X	1	Both edges

11.3.2 Level-Sensitive Mode

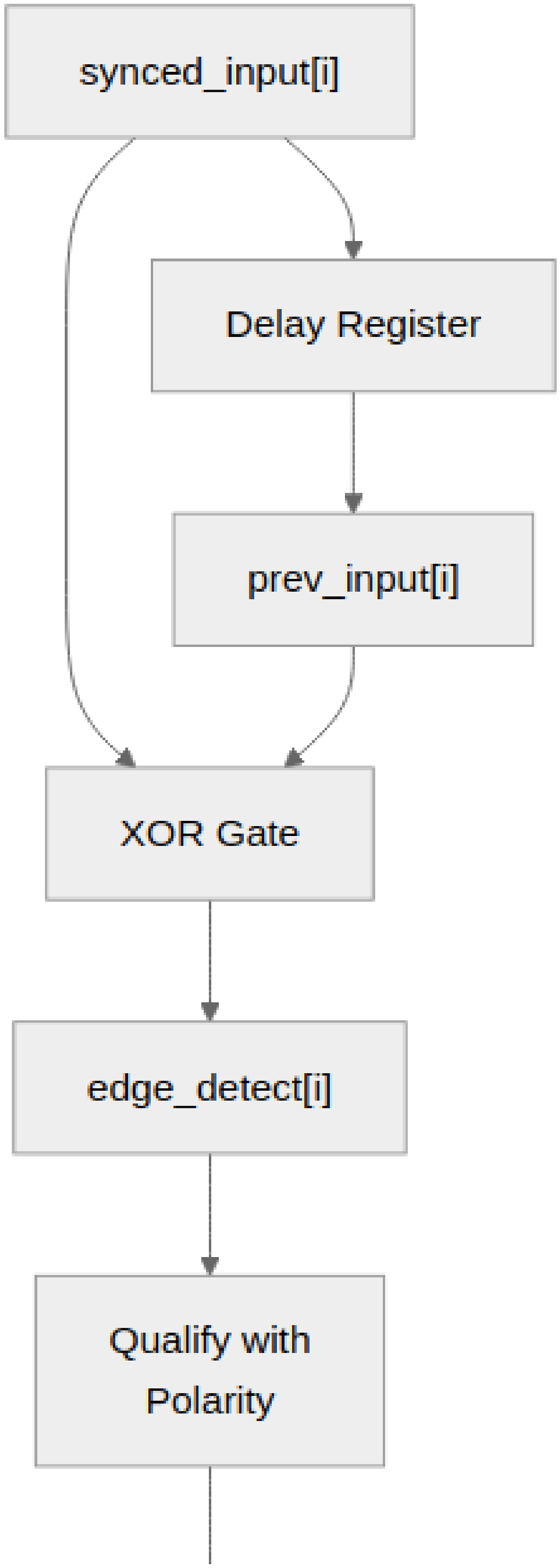
`GPIO_INT_TYPE[i] = 1`

Table 2.6: Level-Sensitive Modes

GPIO_INT_POLARITY	Trigger Condition
0	Active low (interrupt while pin = 0)
1	Active high (interrupt while pin = 1)

11.4 Edge Detection Logic

11.4.1 Figure 2.6: Edge Detection Logic


synced_input[i]
Delay Register
prev_input[i]
XOR Gate
edge_detect[i]
Qualify with Polarity

Edge Detection Logic

11.5 Interrupt Status

11.5.1 Status Register

- Each bit in `GPIO_INT_STATUS` corresponds to one pin
- Set when interrupt condition detected
- Cleared by writing 1 to the bit (W1C)

11.5.2 Interrupt Enable

- `GPIO_INT_ENABLE[i] = 1` enables interrupt for pin i
- Disabled pins don't affect `irq` output
- Status bits still set regardless of enable

11.6 Aggregate IRQ Output

```
irq = |(gpio_int_status & gpio_int_enable)
```

Single IRQ output is OR of all enabled, active interrupts.

11.7 Interrupt Handling Flow

1. Hardware detects condition, sets status bit
2. IRQ asserted to processor
3. Software reads `GPIO_INT_STATUS` to identify source
4. Software handles interrupt
5. Software writes 1 to status bit to clear
6. IRQ deasserts (if no other sources active)

11.8 Implementation Notes

- Edge detection uses synchronized input
- Level-sensitive interrupts re-trigger if not cleared
- Status bits latch until software clears

Next: 05_cdc_logic.md - CDC Logic

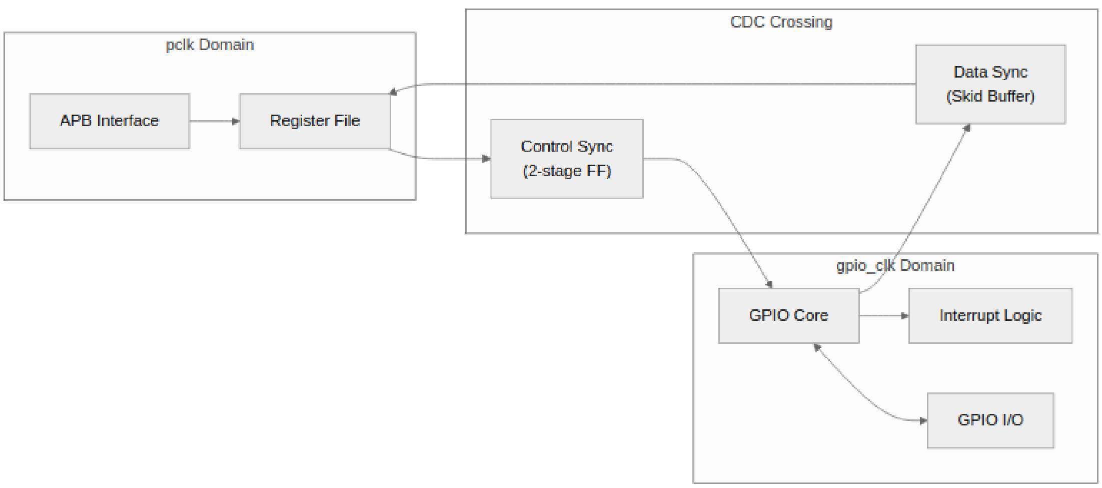
12 APB GPIO - CDC Logic Block

12.1 Overview

Optional clock domain crossing logic enables GPIO core to run on a separate clock from the APB interface.

12.2 Block Diagram

12.2.1 Figure 2.7: CDC Logic Block Diagram



CDC Logic Block

12.3 Configuration

12.3.1 Parameter: CDC_ENABLE

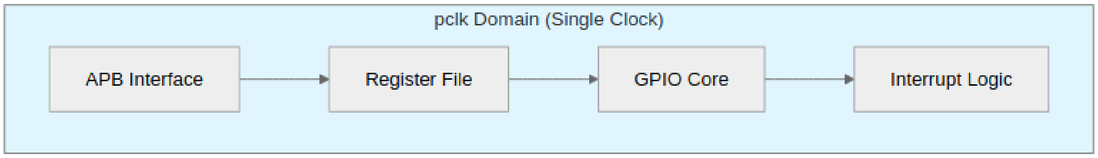
Table 2.7: CDC Enable Parameter

Value	Behavior
0	Single clock domain, all logic on pclk
1	Dual clock domain, GPIO core on gpio_clk

12.4 Clock Domains

12.4.1 When CDC_ENABLE = 0

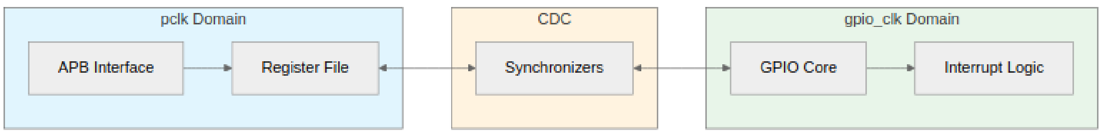
12.4.2 Figure 2.8: Single Clock Domain (CDC Disabled)



CDC Disabled

12.4.3 When CDC_ENABLE = 1

12.4.4 Figure 2.9: Dual Clock Domain (CDC Enabled)



CDC Enabled

12.5 CDC Implementation

12.5.1 APB to GPIO Direction

Register values synchronized to gpio_clk domain: - `gpio_direction` - `gpio_output` - `gpio_int_enable` - `gpio_int_type` - `gpio_int_polarity` - `gpio_int_both`

12.5.2 GPIO to APB Direction

Status values synchronized to pclk domain: - `gpio_input` (synchronized input values) - `gpio_int_status` (interrupt status)

12.6 Synchronization Method

12.6.1 Control Signals

Dual flip-flop synchronizers for single-bit controls.

12.6.2 Multi-bit Data

Skid buffers with handshake protocol for register transfers.

12.7 Timing Considerations

12.7.1 Latency

Table 2.8: CDC Latency

Path	Latency
Register write to GPIO output	2-4 gpio_clk cycles
GPIO input to register read	2-4 pclk cycles
Interrupt detection to IRQ	2-4 pclk cycles

12.7.2 Coherency

- No guaranteed atomicity across clock domains
- Software must handle potential inconsistencies
- Interrupt status always reflects gpio_clk domain

12.8 Reset Synchronization

Both resets must be asserted at power-on: 1. Assert both `presetn` and `gpio_rstn` 2. Release `gpio_rstn` first 3. Release `presetn` after gpio_clk domain stable

---

Back to: 00_overview.md - Block Descriptions Overview

Next Chapter: Chapter 3: Interfaces

---

13 APB GPIO - Interfaces Overview

13.1 External Interfaces

The APB GPIO module has the following external interfaces:

Interface	Type	Description
APB Slave	Bus	Configuration and

Interface	Type	Description
		status access
GPIO Pins	I/O	32 general-purpose I/O pins
Interrupt	Signal	Aggregate interrupt output
Clocks/Reset	System	Clock and reset inputs

13.2 Interface Summary Diagram

GPIO Interfaces

GPIO Interfaces

13.3 Chapter Contents

13.3.1 APB Slave Interface

Complete APB protocol interface for register access.

See: 01_apb_slave.md

13.3.2 GPIO Pin Interface

External GPIO pin connections with tri-state control.

See: 02_gpio_pins.md

13.3.3 Interrupt Interface

Interrupt request output signal.

See: 03_interrupt.md

13.3.4 System Interface

Clock and reset signal requirements.

See: 04_system.md

Next: 01_apb_slave.md - APB Slave Interface

14 APB GPIO - APB Slave Interface

14.1 Signal Description

14.1.1 APB Slave Signals

Signal	Width	Dir	Description
pclk	1	I	APB clock
presetn	1	I	APB reset (active low)
s_apb_psel	1	I	Peripheral select
s_apb_penable	1	I	Enable phase
s_apb_pwrite	1	I	Write transaction
s_apb_paddr	12	I	Address bus
s_apb_pwdata	32	I	Write data
s_apb_pstrb	4	I	Byte strobes
s_apb_prdata	32	O	Read data
s_apb_pready	1	O	Ready response
s_apb_pslverr	1	O	Slave error

14.2 Protocol Compliance

14.2.1 APB3/APB4 Features

Feature	Support
PSEL	Yes
PENABLE	Yes

Feature	Support
PWRITE	Yes
PADDR	12-bit
PWDATA	32-bit
PRDATA	32-bit
PREADY	Yes (always 1)
PSLVERR	Yes (always 0)
PSTRB	Yes
PPROT	No

14.3 Timing

14.3.1 Zero Wait State

All register accesses complete in minimum APB cycles: - Read: 2 cycles (setup + access) - Write: 2 cycles (setup + access)

14.3.2 Timing Diagram

```
          ___     ___     ___     ___
pclk    _|   |___|   |___|   |___|   |___

         _______________
psel    |               |_________________

                ________
penable ________|        |________________

paddr   --------|  A1    |----------------

pwdata  --------|  D1    |---------------- (write)

prdata  --------|  D1    |---------------- (read)
                ________
pready  ________|        |________________
```

14.4 Address Decoding

14.4.1 Address Map

Address	Register	Access
0x000	GPIO_CONTROL	RW
0x004	GPIO_DIRECTION	RW
0x008	GPIO_OUTPUT	RW
0x00C	GPIO_INPUT	RO
0x010	GPIO_INT_ENABLE	RW
0x014	GPIO_INT_TYPE	RW
0x018	GPIO_INT_POLARITY	RW
0x01C	GPIO_INT_BOTH	RW
0x020	GPIO_INT_STATUS	W1C

14.4.2 Byte Strobes

Byte-granular writes supported: - pstrb[3:0] corresponds to pwdata[31:0] - Unselected bytes retain previous values

14.5 Error Handling

- No address decode errors (all addresses valid)
- No timeout errors
- pslverr always 0

Next: 02_gpio_pins.md - GPIO Pin Interface

---

15 APB GPIO - GPIO Pin Interface

15.1 Signal Description

15.1.1 GPIO Signals

Signal	Width	Dir	Description
gpio_out	32	O	Output data values
gpio_oe	32	O	Output enables
gpio_in	32	I	Input data values

15.2 Pin Behavior

15.2.1 Output Mode (direction[i] = 1)

```
                    +--------+
gpio_output[i] ---->| Buffer |----> External Pin
                    |        |
gpio_oe[i] = 1 ---->| Enable |
                    +--------+
```

- gpio_oe[i] = 1 (output enabled)
- gpio_out[i] = GPIO_OUTPUT register value
- Pin driven to output value

15.2.2 Input Mode (direction[i] = 0)

```
                    +--------+
                    | Buffer |----> External Pin (Hi-Z)
                    |        |
gpio_oe[i] = 0 ---->| Enable |
                    +--------+

External Pin ----> Synchronizer ----> gpio_in[i]
```

- gpio_oe[i] = 0 (tri-state)
- gpio_out[i] = don’t care
- External value captured via synchronizer

15.3 Synchronization

15.3.1 Input Synchronizer

All inputs pass through dual flip-flop synchronizer:

```
         gpio_clk       gpio_clk
            |              |
            v              v
         +-----+        +-----+
gpio_in ->| FF1 |------->| FF2 |-----> synced_input
         +-----+        +-----+
```

- SYNC_STAGES parameter controls depth (default: 2)
- Prevents metastability from asynchronous inputs
- All 32 inputs synchronized in parallel

15.3.2 Input Latency

Input changes visible in GPIO_INPUT register after: - SYNC_STAGES cycles of gpio_clk (or pclk if CDC_ENABLE=0) - Plus APB read latency

15.4 Electrical Considerations

15.4.1 Output Characteristics

Parameter	Description
Drive	Standard CMOS output
Slew	Defined by I/O cell
Protection	ESD per I/O cell design

15.4.2 Input Characteristics

Parameter	Description
Levels	CMOS compatible
Hysteresis	Optional per I/O cell
Pull-up/down	External to GPIO module

15.5 Timing Constraints

15.5.1 Output Path

- gpio_out updates on clock edge

- gpio_oe updates on same clock edge
- No glitch-free guarantee during direction change

15.5.2 Input Path

- Setup/hold relative to synchronizer clock
- Metastability resolved by synchronizer

Next: 03_interrupt.md - Interrupt Interface

---

16 APB GPIO - Interrupt Interface

16.1 Signal Description

Signal	Width	Dir	Description
irq	1	O	Interrupt request (active high)

16.2 Interrupt Generation

16.2.1 Aggregate Logic

```
irq = |(GPIO_INT_STATUS[31:0] & GPIO_INT_ENABLE[31:0])
```

IRQ is asserted when any enabled interrupt source is active.

16.2.2 Per-Pin Configuration

Each GPIO pin can generate interrupts independently:

Register	Function
GPIO_INT_ENABLE	Enable/disable per pin
GPIO_INT_TYPE	Edge (0) or Level (1)
GPIO_INT_POLARITY	Falling/Low (0) or Rising/High

Register	Function
	(1)
GPIO_INT_BOTH	Both edges (edge mode only)
GPIO_INT_STATUS	Current interrupt status

16.3 Interrupt Modes

16.3.1 Edge-Triggered

`GPIO_INT_TYPE[i] = 0`

```
        synced_input
              |
              v
         +----------+
         | Edge     |
         | Detect   |---> Set STATUS[i]
         +----------+
              ^
              |
    Polarity/Both Config
```

- Captures transitions on synchronized input
- Status bit latches until cleared by software
- Both-edge mode ignores polarity setting

16.3.2 Level-Sensitive

`GPIO_INT_TYPE[i] = 1`

```
        synced_input
              |
              v
         +----------+
         | Level    |
         | Compare  |---> STATUS[i]
         +----------+
              ^
              |
    Polarity Config
```

- Continuously compares input to polarity
- Status follows input level
- Re-triggers if not cleared while active

16.4 Interrupt Timing

16.4.1 Edge-Triggered Latency

```
External event --> Synchronizer --> Edge detect --> STATUS set --> IRQ
                   (2 cycles)      (1 cycle)       (1 cycle)

Total: 4 clock cycles typical
```

16.4.2 Level-Sensitive Latency

```
External level --> Synchronizer --> Level compare --> IRQ
                   (2 cycles)       (1 cycle)

Total: 3 clock cycles typical
```

16.5 Interrupt Handling

16.5.1 Software Sequence

1. IRQ asserts (hardware)
2. CPU vectors to interrupt handler
3. Read GPIO_INT_STATUS to identify sources
4. Handle interrupt condition
5. Write 1 to GPIO_INT_STATUS bits to clear
6. IRQ deasserts if no other sources

16.5.2 Clearing Interrupts

Mode	Clear Method
Edge	Write 1 to STATUS bit
Level	Clear source, then write 1 to STATUS

16.6 Connection Guidelines

- Connect to interrupt controller input
- Active-high, level-sensitive recommended at controller
- Single IRQ covers all 32 GPIO pins

Next: 04_system.md - System Interface

17 APB GPIO - System Interface

17.1 Clock Signals

17.1.1 pclk - APB Clock

Parameter	Value
Purpose	APB interface clock
Frequency	50-200 MHz typical
Domain	APB bus timing

Used for: - APB protocol timing - Register file access - IRQ generation (single-clock mode)

17.1.2 gpio_clk - GPIO Clock

Parameter	Value
Purpose	GPIO core clock
Frequency	Application dependent
Usage	Only when CDC_ENABLE=1

Used for: - Input synchronization - Output register updates - Interrupt detection

17.2 Reset Signals

17.2.1 presetn - APB Reset

Parameter	Value
Polarity	Active low
Type	Asynchronous assert, synchronous deassert
Scope	APB interface logic

Resets: - APB state machine - Register file - Response logic

17.2.2 gpio_rstn - GPIO Reset

Parameter	Value
Polarity	Active low
Type	Asynchronous assert, synchronous deassert
Usage	Only when CDC_ENABLE=1

Resets: - GPIO output registers - Input synchronizers - Interrupt state

17.3 Clock Configurations

17.3.1 Single Clock Domain (CDC_ENABLE = 0)

```
            +-------------------+
pclk ------>| APB GPIO          |
            | (all logic)       |
            +-------------------+

gpio_clk: Tie to pclk or leave unconnected
```

17.3.2 Dual Clock Domain (CDC_ENABLE = 1)

```
            +-------------------+
pclk ------>| APB Interface     |
            |      |CDC|        |
gpio_clk -->| GPIO Core         |
            +-------------------+
```

17.4 Reset Sequence

17.4.1 Power-On Reset

1. Assert both presetn and gpio_rstn low
2. Clocks may be running or stopped
3. Hold reset for minimum 2 clock cycles
4. Release gpio_rstn first
5. Wait 2 gpio_clk cycles
6. Release presetn
7. Wait 2 pclk cycles before APB access

17.4.2 Timing Diagram

```
         ________________________________________
pclk    |  |  |  |  |  |  |  |  |  |  |  |  |  |
```

```
            _______________________________________
gpio_clk |  |  |  |  |  |  |  |  |  |  |  |  |  |
                                   _____________________
presetn   ____________________|
                             ___________________________
gpio_rstn ______________|

             |<- min 2 ->|<- 2 ->|<- 2 ->|
                clocks     gclk    pclk
```

17.5 Constraints

17.5.1 Clock Relationship

Mode	Constraint
CDC_ENABLE=0	Single clock, no special constraints
CDC_ENABLE=1	Set false_path between domains

17.5.2 Reset Recovery

- Allow minimum 2 clock cycles after reset deassert
- First APB transaction may start on 3rd cycle

Back to: 00_overview.md - Interfaces Overview

Next Chapter: Chapter 4: Programming Model

---

18 APB GPIO - Programming Model Overview

18.1 Register Summary

Offset	Name	Access	Description
0x000	GPIO_CONTROL	RW	Global control
0x004	GPIO_DIRECTION	RW	I/O direction
0x008	GPIO_OUTPUT	RW	Output data
0x00C	GPIO_INPUT	RO	Input data
0x010	GPIO_INT_ENABLE	RW	Interrupt enable
0x014	GPIO_INT_TYPE	RW	Edge/level select
0x018	GPIO_INT_POLARITY	RW	Interrupt polarity
0x01C	GPIO_INT_BOTH	RW	Both edges
0x020	GPIO_INT_STATUS	W1C	Interrupt status

18.2 Chapter Contents

18.2.1 Basic Operations

Fundamental GPIO read/write operations.

See: 01_basic_operations.md

18.2.2 Interrupt Configuration

Setting up and handling GPIO interrupts.

See: 02_interrupt_config.md

18.2.3 Programming Examples

Common use cases with code samples.

See: 03_examples.md

18.2.4 Software Considerations

Performance tips and best practices.

See: 04_software_notes.md

Next: 01_basic_operations.md - Basic Operations

RTL Design Sherpa · Learning Hardware Design Through Practice GitHub · Documentation Index · MIT License

---

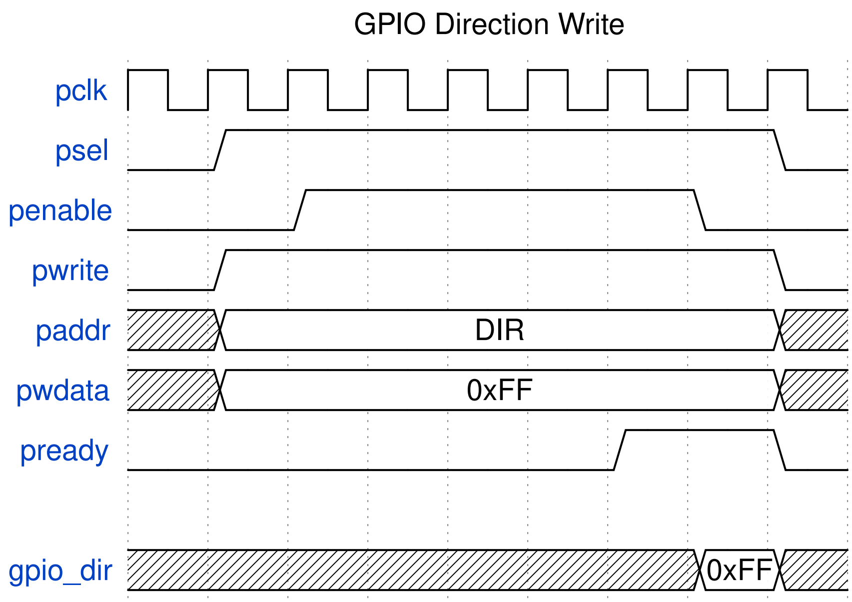
19 APB GPIO - Basic Operations

19.1 Timing Diagrams

The following diagrams show the internal signal flow for basic GPIO operations.

19.1.1 Direction Configuration

When software writes to GPIO_DIRECTION, the direction register updates and controls the output enable for each pin.

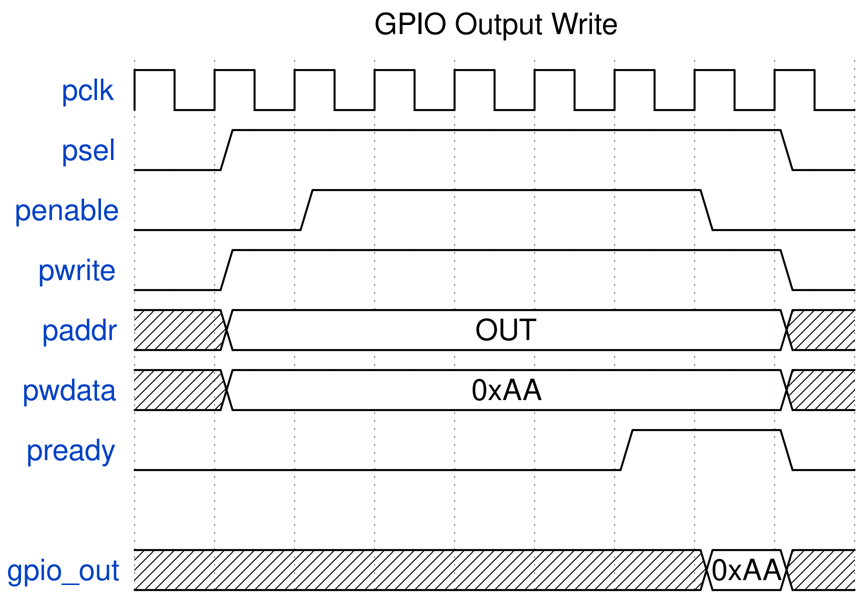


GPIO Direction Write

The APB write completes in a single cycle. The direction register (`r_gpio_direction`) updates on the clock edge following PREADY, and the output enable (`gpio_oe`) reflects the new configuration immediately.

19.1.2 Output Write

Writing to GPIO_OUTPUT sets the output data register, which drives the external pins when direction is set to output.

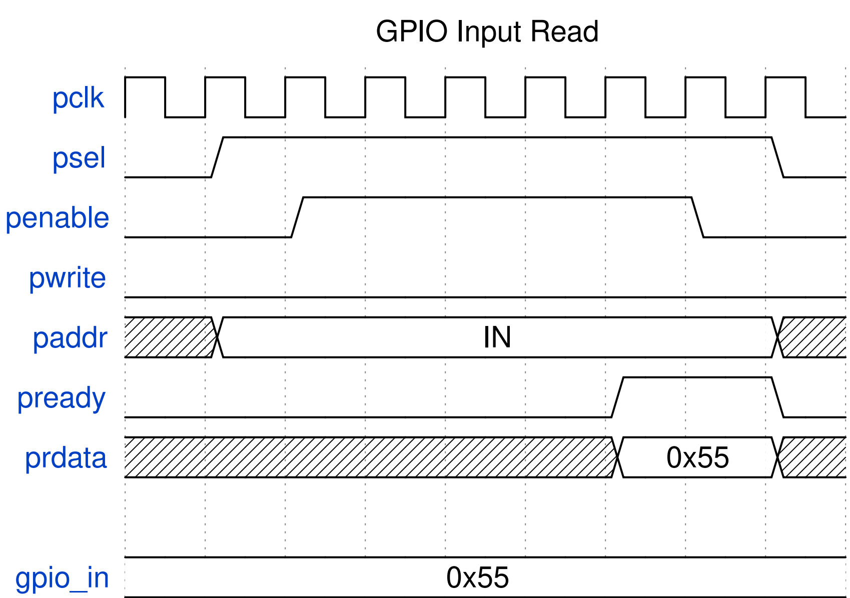


GPIO Output Write

The write data flows through the APB interface to the output register. When `gpio_oe[n]` is high (output mode), `gpio_out[n]` drives the written value to the external pin.

19.1.3 Input Read

Reading GPIO_INPUT returns the synchronized input values from external pins.

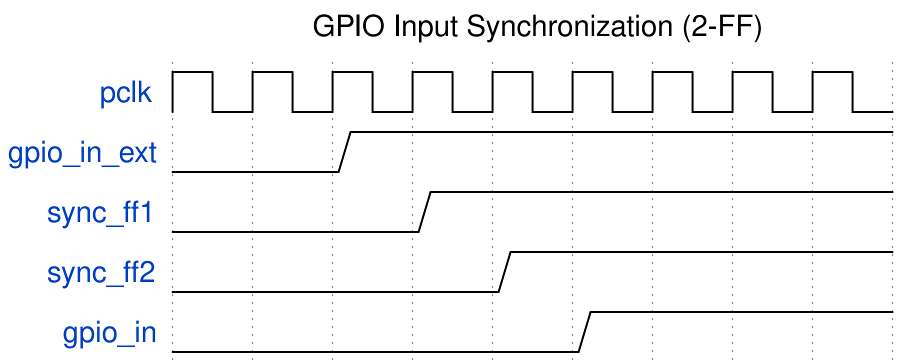


GPIO Input Read

External inputs pass through a 2-stage synchronizer before being captured. The synchronized value (`w_gpio_sync`) is returned on `s_apb_PRDATA` during the APB read transaction.

19.1.4 Input Synchronization

All GPIO inputs pass through a 2-stage synchronizer to prevent metastability.

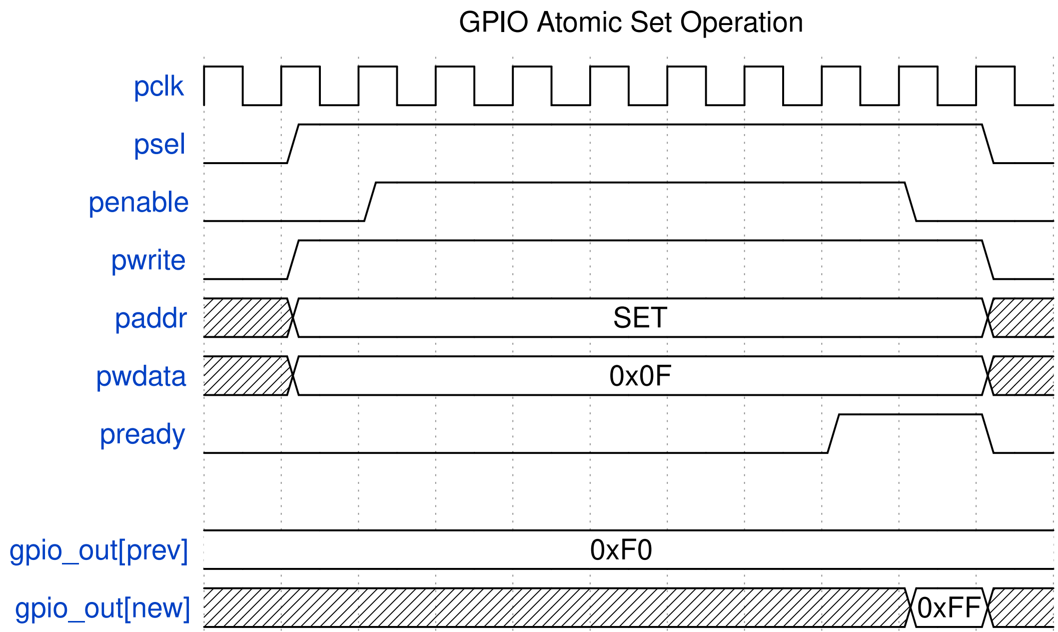


GPIO Input Sync

The synchronizer adds 2 clock cycles of latency. External asynchronous transitions on `gpio_in` propagate through `sync_stage1` and `sync_stage2` before appearing on the internal synchronized signal `w_gpio_sync`.

19.1.5 Atomic Operations

The SET, CLEAR, and TOGGLE registers provide atomic bit manipulation without read-modify-write races.



GPIO Atomic Operations

Three consecutive APB writes demonstrate: 1. **GPIO_SET**: Sets bits where write data is 1, leaves others unchanged 2. **GPIO_CLEAR**: Clears bits where write data is 1, leaves others unchanged 3. **GPIO_TOGGLE**: Inverts bits where write data is 1, leaves others unchanged

---

19.2 Initialization

19.2.1 Reset State

After reset, all registers are 0: - GPIO disabled - All pins configured as inputs - No interrupts enabled

19.2.2 Enable GPIO

```
// Enable GPIO controller
GPIO_CONTROL = 0x00000001;
```

19.3 Output Operations

19.3.1 Configure as Output

```
// Set pins 7:4 as outputs (bits = 1 for output)
GPIO_DIRECTION = 0x000000F0;
```

19.3.2 Write Output Values

```
// Set pins 7:4 to value 0101
GPIO_OUTPUT = 0x00000050;
```

19.3.3 Toggle Outputs

```
// Read current output, XOR to toggle
uint32_t current = GPIO_OUTPUT;
GPIO_OUTPUT = current ^ 0x000000F0;  // Toggle pins 7:4
```

19.3.4 Atomic Bit Operations

```
// Set specific bits (pins 5 and 7)
GPIO_OUTPUT |= 0x000000A0;

// Clear specific bits (pins 4 and 6)
GPIO_OUTPUT &= ~0x00000050;
```

19.4 Input Operations

19.4.1 Configure as Input

```
// Set pins 3:0 as inputs (bits = 0 for input)
GPIO_DIRECTION &= ~0x0000000F;
```

19.4.2 Read Input Values

```
// Read all inputs
uint32_t inputs = GPIO_INPUT;

// Check specific pin (pin 2)
if (inputs & 0x00000004) {
    // Pin 2 is high
}
```

19.4.3 Read with Mask

```
// Read only pins 3:0
uint32_t low_nibble = GPIO_INPUT & 0x0000000F;
```

19.5 Mixed I/O Configuration

19.5.1 Configure Mixed Directions

```
// Pins 31:16 = outputs, pins 15:0 = inputs
GPIO_DIRECTION = 0xFFFF0000;
```

19.5.2 Read-Modify-Write Pattern

```
// Change only pins 11:8 to outputs
uint32_t dir = GPIO_DIRECTION;
dir |= 0x00000F00;    // Set pins 11:8
GPIO_DIRECTION = dir;
```

19.6 Output Enable Behavior

19.6.1 Hardware Interface

When direction bit is set: - `gpio_oe[i]` = 1 (output enabled) - `gpio_out[i]` = GPIO_OUTPUT[i] value

When direction bit is clear: - `gpio_oe[i]` = 0 (high impedance) - `gpio_out[i]` = don't care

19.6.2 Glitch Considerations

To avoid output glitches when switching direction: 1. Set GPIO_OUTPUT to desired value 2. Then change GPIO_DIRECTION

```
// Safe direction change to output
GPIO_OUTPUT = desired_value;    // Set value first
GPIO_DIRECTION |= pin_mask;     // Then enable output
```

Next: 02_interrupt_config.md - Interrupt Configuration

---

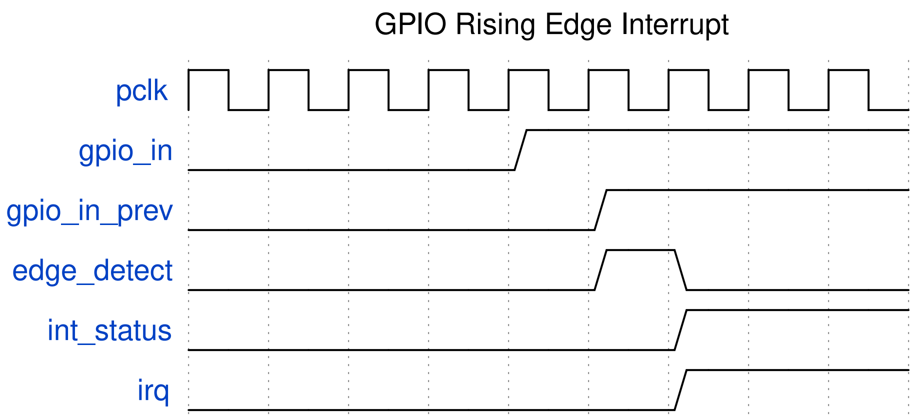
20 APB GPIO - Interrupt Configuration

20.1 Interrupt Timing Diagrams

The following diagrams illustrate GPIO interrupt detection and handling.

20.1.1 Rising Edge Interrupt

Edge-triggered interrupts detect transitions on input pins.

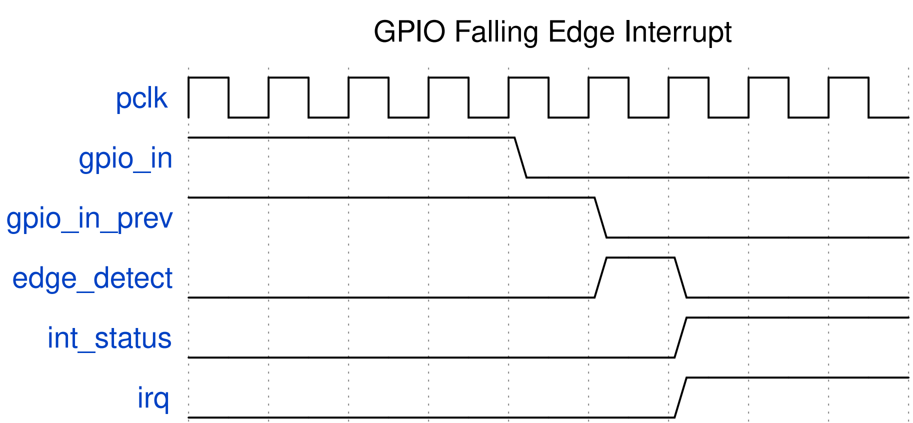


GPIO Rising Edge Interrupt

The detection sequence: 1. External input `gpio_in[0]` transitions from 0 to 1 2. 2-stage synchronizer captures the transition (`w_gpio_sync`) 3. Edge detector compares current vs. delayed value (`r_gpio_sync_d`) 4. Rising edge pulse (`w_rising_edge`) generated for one clock 5. Raw interrupt latched in `r_raw_int[0]` 6. Combined `irq` output asserts

20.1.2 Falling Edge Interrupt

Falling edge detection uses inverted polarity configuration.

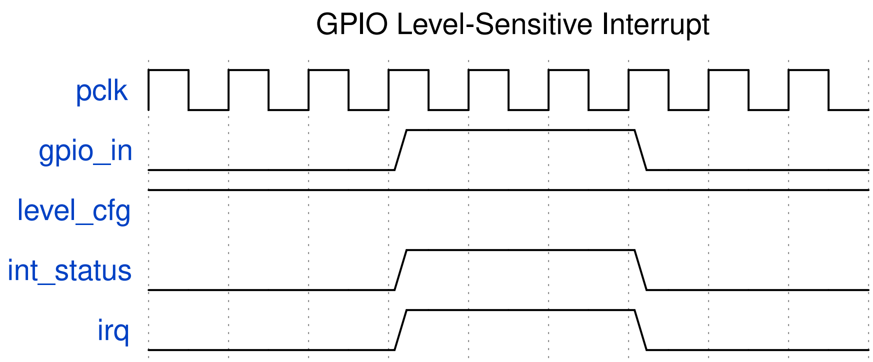


GPIO Falling Edge Interrupt

With `cfg_int_type[0]=0` (edge mode) and `cfg_int_polarity[0]=0` (falling edge), the detector triggers on 1-to-0 transitions.

20.1.3 Level-Sensitive Interrupt

Level-sensitive interrupts track the input state directly.

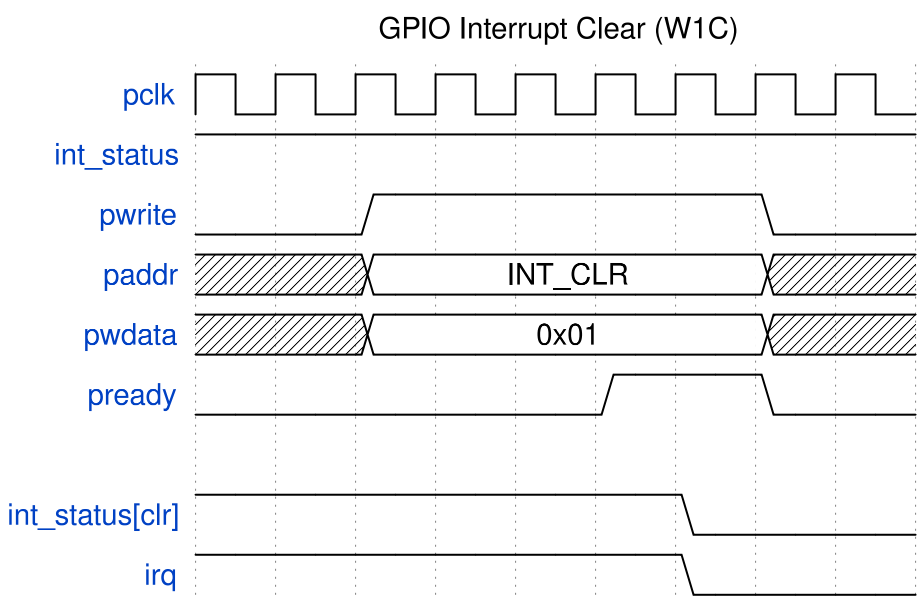


GPIO Level Interrupt

Key differences from edge mode: - `irq` follows the input level (not latched) - No edge detection logic involved - Interrupt re-asserts if source not cleared before ISR exit

20.1.4 Interrupt Clear (W1C)

Write-1-to-Clear mechanism clears latched interrupts.



GPIO Interrupt Clear

The clear sequence: 1. `r_raw_int[0]` is active (edge was detected) 2. Software writes 0x01 to INT_STATUS register 3. W1C logic clears `r_int_status[0]` 4. `irq` deasserts

Note: For level-sensitive interrupts, the external source must be cleared first, otherwise the interrupt immediately re-asserts.

20.2 Interrupt Setup

20.2.1 1. Configure Interrupt Type

```
// Edge-triggered (0) or Level-sensitive (1)
// Pins 3:0 edge, pins 7:4 level
GPIO_INT_TYPE = 0x000000F0;
```

20.2.2 2. Configure Polarity

For edge mode: - 0 = falling edge - 1 = rising edge

For level mode: - 0 = active low - 1 = active high

```
// Rising edge / active high for pins 7:0
GPIO_INT_POLARITY = 0x000000FF;
```

20.2.3 3. Configure Both-Edge Mode (Edge Mode Only)

```
// Enable both edges for pin 0
GPIO_INT_BOTH = 0x00000001;
```

20.2.4 4. Enable Interrupts

```
// Enable interrupts on pins 7:0
GPIO_INT_ENABLE = 0x000000FF;
```

20.3 Interrupt Configuration Table

INT_TYPE	INT_POLARITY	INT_BOTH	Trigger
0	0	0	Falling edge
0	1	0	Rising edge
0	X	1	Both edges
1	0	X	Active low
1	1	X	Active high

20.4 Complete Setup Examples

20.4.1 Rising Edge Interrupt

```
// Configure pin 5 for rising edge interrupt
GPIO_INT_TYPE &= ~(1 << 5);      // Edge mode
GPIO_INT_POLARITY |= (1 << 5);   // Rising edge
GPIO_INT_BOTH &= ~(1 << 5);      // Single edge
GPIO_INT_ENABLE |= (1 << 5);     // Enable
```

20.4.2 Both-Edge Interrupt

```
// Configure pin 3 for both-edge interrupt
GPIO_INT_TYPE &= ~(1 << 3);       // Edge mode
GPIO_INT_BOTH |= (1 << 3);        // Both edges
GPIO_INT_ENABLE |= (1 << 3);      // Enable
```

20.4.3 Active-Low Level Interrupt

```
// Configure pin 7 for active-low level interrupt
GPIO_INT_TYPE |= (1 << 7);        // Level mode
GPIO_INT_POLARITY &= ~(1 << 7);   // Active low
GPIO_INT_ENABLE |= (1 << 7);      // Enable
```

20.5 Interrupt Handling

20.5.1 Check Interrupt Status

```
uint32_t status = GPIO_INT_STATUS;
```

20.5.2 Clear Interrupts (Write-1-to-Clear)

```
// Clear specific interrupt (pin 5)
GPIO_INT_STATUS = (1 << 5);

// Clear all pending interrupts
GPIO_INT_STATUS = 0xFFFFFFFF;
```

20.5.3 Complete ISR Example

```
void gpio_isr(void) {
    // Read status
    uint32_t status = GPIO_INT_STATUS;

    // Handle each pending interrupt
    for (int i = 0; i < 32; i++) {
        if (status & (1 << i)) {
            handle_gpio_event(i);
        }
    }

    // Clear handled interrupts
    GPIO_INT_STATUS = status;
}
```

20.6 Level-Sensitive Considerations

20.6.1 Avoid Interrupt Storm

For level-sensitive interrupts, the source must be cleared before the status:

```c
void level_sensitive_isr(void) {
    uint32_t status = GPIO_INT_STATUS;

    // For level-sensitive pins, handle source first
    if (status & LEVEL_PIN_MASK) {
        clear_external_source();  // Clear what's driving pin
    }

    // Then clear status
    GPIO_INT_STATUS = status;
}
```

20.6.2 Masking During Handling

```c
// Temporarily disable while handling
uint32_t saved_enable = GPIO_INT_ENABLE;
GPIO_INT_ENABLE = 0;  // Disable all

// Handle interrupt source
handle_interrupt();

// Re-enable
GPIO_INT_ENABLE = saved_enable;
```

Next: 03_examples.md - Programming Examples

---

21 APB GPIO - Programming Examples

21.1 LED Control

21.1.1 Simple LED Blink

```
#define LED_PIN  (1 << 0)

void led_init(void) {
    GPIO_CONTROL = 1;           // Enable GPIO
    GPIO_DIRECTION |= LED_PIN;  // Set as output
}

void led_on(void) {
    GPIO_OUTPUT |= LED_PIN;
}

void led_off(void) {
    GPIO_OUTPUT &= ~LED_PIN;
}

void led_toggle(void) {
    GPIO_OUTPUT ^= LED_PIN;
}
```

21.1.2 Multiple LED Control

```
#define LED_MASK  0x000000FF  // LEDs on pins 7:0

void leds_write(uint8_t pattern) {
    uint32_t output = GPIO_OUTPUT;
    output = (output & ~LED_MASK) | pattern;
    GPIO_OUTPUT = output;
}
```

21.2 Button Input

21.2.1 Simple Button Read

```
#define BUTTON_PIN  (1 << 8)

void button_init(void) {
    GPIO_CONTROL = 1;              // Enable GPIO
    GPIO_DIRECTION &= ~BUTTON_PIN; // Set as input
}
```

```c
bool button_pressed(void) {
    return (GPIO_INPUT & BUTTON_PIN) != 0;
}
```

21.2.2 Button with Interrupt

```c
#define BUTTON_PIN  (1 << 8)

volatile bool button_event = false;

void button_init_irq(void) {
    GPIO_CONTROL = 1;
    GPIO_DIRECTION &= ~BUTTON_PIN;

    // Configure falling edge interrupt (button press)
    GPIO_INT_TYPE &= ~BUTTON_PIN;      // Edge mode
    GPIO_INT_POLARITY &= ~BUTTON_PIN;  // Falling edge
    GPIO_INT_BOTH &= ~BUTTON_PIN;      // Single edge
    GPIO_INT_ENABLE |= BUTTON_PIN;     // Enable
}

void button_isr(void) {
    if (GPIO_INT_STATUS & BUTTON_PIN) {
        button_event = true;
        GPIO_INT_STATUS = BUTTON_PIN;  // Clear
    }
}
```

21.3 DIP Switch Reading

21.3.1 8-Bit Switch Input

```c
#define SWITCH_MASK  0x00FF0000  // Switches on pins 23:16
#define SWITCH_SHIFT 16

void switch_init(void) {
    GPIO_CONTROL = 1;
    GPIO_DIRECTION &= ~SWITCH_MASK;  // All inputs
}

uint8_t switch_read(void) {
    return (GPIO_INPUT & SWITCH_MASK) >> SWITCH_SHIFT;
}
```

21.4 Parallel Data Interface

21.4.1 8-Bit Output Port

```
#define DATA_MASK   0x000000FF  // Data on pins 7:0
#define STROBE_PIN  (1 << 8)    // Strobe on pin 8

void data_port_init(void) {
    GPIO_CONTROL = 1;
    GPIO_DIRECTION |= (DATA_MASK | STROBE_PIN);
    GPIO_OUTPUT &= ~STROBE_PIN;  // Strobe low
}

void data_write(uint8_t data) {
    uint32_t output = GPIO_OUTPUT;
    output = (output & ~DATA_MASK) | data;
    GPIO_OUTPUT = output;

    // Generate strobe pulse
    GPIO_OUTPUT |= STROBE_PIN;
    // Small delay if needed
    GPIO_OUTPUT &= ~STROBE_PIN;
}
```

21.4.2 8-Bit Input Port with Ready

```
#define DATA_MASK   0x000000FF  // Data on pins 7:0
#define READY_PIN   (1 << 8)    // Ready on pin 8

void data_input_init(void) {
    GPIO_CONTROL = 1;
    GPIO_DIRECTION &= ~(DATA_MASK | READY_PIN);

    // Interrupt on ready rising edge
    GPIO_INT_TYPE &= ~READY_PIN;
    GPIO_INT_POLARITY |= READY_PIN;
    GPIO_INT_ENABLE |= READY_PIN;
}

uint8_t data_read(void) {
    return GPIO_INPUT & DATA_MASK;
}
```

21.5 PWM-Style Output

21.5.1 Bit-Banged PWM (Low Frequency)

```
#define PWM_PIN  (1 << 0)

void pwm_init(void) {
    GPIO_CONTROL = 1;
    GPIO_DIRECTION |= PWM_PIN;
}

// Call from timer interrupt
void pwm_update(uint8_t duty, uint8_t *counter) {
    (*counter)++;
    if (*counter >= 100) *counter = 0;

    if (*counter < duty) {
        GPIO_OUTPUT |= PWM_PIN;
    } else {
        GPIO_OUTPUT &= ~PWM_PIN;
    }
}
```

21.6 Wake-On-Change

21.6.1 Power Management Integration

```
#define WAKE_PINS  0x0000000F  // Wake sources on pins 3:0

void wake_setup(void) {
    // Configure both-edge interrupts for wake pins
    GPIO_INT_TYPE &= ~WAKE_PINS;   // Edge mode
    GPIO_INT_BOTH |= WAKE_PINS;    // Both edges
    GPIO_INT_ENABLE |= WAKE_PINS;  // Enable

    // Clear any pending before sleep
    GPIO_INT_STATUS = WAKE_PINS;
}

void enter_sleep(void) {
    wake_setup();
    // Platform-specific sleep entry
    __WFI();  // Wait for interrupt
}
```

Next: 04_software_notes.md - Software Considerations

22 APB GPIO - Software Considerations

22.1 Performance

22.1.1 Register Access Timing

Operation	APB Cycles	Notes
Read	2	Setup + access
Write	2	Setup + access
Read-Modify-Write	4	Read + write

22.1.2 Optimizing Access

Batch operations when possible:

```
// Inefficient - 4 separate writes
GPIO_OUTPUT |= (1 << 0);
GPIO_OUTPUT |= (1 << 1);
GPIO_OUTPUT |= (1 << 2);
GPIO_OUTPUT |= (1 << 3);

// Efficient - single write
GPIO_OUTPUT |= 0x0000000F;
```

Cache register values:

```
// Inefficient - 4 reads + 4 writes
for (int i = 0; i < 4; i++) {
    GPIO_DIRECTION |= (1 << i);
}

// Efficient - 1 read + 1 write
uint32_t dir = GPIO_DIRECTION;
dir |= 0x0000000F;
GPIO_DIRECTION = dir;
```

22.2 Synchronization

22.2.1 Input Latency

GPIO inputs have inherent latency: - SYNC_STAGES clock cycles (default 2) - Plus software polling/interrupt overhead

Account for latency in timing-critical code.

22.2.2 Volatile Registers

Always declare GPIO registers as volatile:

```
#define GPIO_INPUT  (*(volatile uint32_t *)0xFEC0700C)
```

22.2.3 Multi-Core Considerations

If multiple cores access GPIO:

```
// Use atomic operations or locks
spin_lock(&gpio_lock);
uint32_t val = GPIO_OUTPUT;
val |= new_bits;
GPIO_OUTPUT = val;
spin_unlock(&gpio_lock);
```

22.3 Interrupt Best Practices

22.3.1 Clear Before Return

Always clear interrupt status before ISR return:

```
void gpio_isr(void) {
    uint32_t status = GPIO_INT_STATUS;
    // Handle interrupts
    GPIO_INT_STATUS = status;  // Must clear!
}
```

22.3.2 Avoid Spurious Interrupts

Disable interrupts during configuration:

```
void reconfigure_interrupt(int pin) {
    uint32_t mask = (1 << pin);

    // Disable first
    GPIO_INT_ENABLE &= ~mask;

    // Reconfigure
```

```
    // ...

    // Clear any pending
    GPIO_INT_STATUS = mask;

    // Re-enable
    GPIO_INT_ENABLE |= mask;
}
```

22.3.3 Level-Sensitive Caution

Level interrupts can cause interrupt storms:

```
void level_isr(void) {
    // WRONG - will re-trigger immediately
    GPIO_INT_STATUS = status;

    // RIGHT - handle source first
    clear_external_interrupt_source();
    GPIO_INT_STATUS = status;
}
```

22.4 Error Handling

22.4.1 No Hardware Errors

GPIO controller doesn't generate errors: - All addresses valid - pslverr always 0

22.4.2 Software Validation

Validate configuration in software:

```
bool gpio_set_direction(uint32_t pin, bool output) {
    if (pin >= 32) return false;

    if (output) {
        GPIO_DIRECTION |= (1 << pin);
    } else {
        GPIO_DIRECTION &= ~(1 << pin);
    }
    return true;
}
```

22.5 Debug Tips

22.5.1 Read-Back Verification

```c
void gpio_debug(void) {
    printf("CONTROL:   0x%08X\n", GPIO_CONTROL);
    printf("DIRECTION: 0x%08X\n", GPIO_DIRECTION);
    printf("OUTPUT:    0x%08X\n", GPIO_OUTPUT);
    printf("INPUT:     0x%08X\n", GPIO_INPUT);
    printf("INT_STAT:  0x%08X\n", GPIO_INT_STATUS);
}
```

22.5.2 Loopback Testing

Connect output to input for self-test:

```c
bool gpio_loopback_test(int out_pin, int in_pin) {
    GPIO_DIRECTION |= (1 << out_pin);
    GPIO_DIRECTION &= ~(1 << in_pin);

    // Test high
    GPIO_OUTPUT |= (1 << out_pin);
    delay_us(10);  // Allow synchronization
    if (!(GPIO_INPUT & (1 << in_pin))) return false;

    // Test low
    GPIO_OUTPUT &= ~(1 << out_pin);
    delay_us(10);
    if (GPIO_INPUT & (1 << in_pin)) return false;

    return true;
}
```

Back to: 00_overview.md - Programming Model Overview

Next Chapter: Chapter 5: Registers

---

23 APB GPIO - Register Map

23.1 Register Summary

Offset	Name	Access	Reset	Description
0x000	GPIO_CONTROL	RW	0x00000000	Global control
0x004	GPIO_DIRECTION	RW	0x00000000	Pin direction
0x008	GPIO_OUTPUT	RW	0x00000000	Output data
0x00C	GPIO_INPUT	RO	-	Input data
0x010	GPIO_INT_ENABLE	RW	0x00000000	Interrupt enable
0x014	GPIO_INT_TYPE	RW	0x00000000	Interrupt type
0x018	GPIO_INT_POLARITY	RW	0x00000000	Interrupt polarity
0x01C	GPIO_INT_BOTH	RW	0x00000000	Both-edge enable
0x020	GPIO_INT_STATUS	W1C	0x00000000	Interrupt status

23.2 GPIO_CONTROL (0x000)

Global control register.

Bits	Name	Access	Reset	Description
31:1	Reserved	RO	0	Reserved
0	ENABLE	RW	0	GPIO enable (1=enabled)

23.3 GPIO_DIRECTION (0x004)

Pin direction control. Each bit controls one GPIO pin.

Bits	Name	Access	Reset	Description
31:0	DIR	RW	0	Direction per pin (0=input, 1=output)

23.4 GPIO_OUTPUT (0x008)

Output data register. Values driven when pin configured as output.

Bits	Name	Access	Reset	Description
31:0	DATA	RW	0	Output values per pin

23.5 GPIO_INPUT (0x00C)

Input data register. Reflects synchronized external pin values.

Bits	Name	Access	Reset	Description
31:0	DATA	RO	-	Input values per pin

Note: Value depends on external signals, not reset.

23.6 GPIO_INT_ENABLE (0x010)

Interrupt enable register. Controls which pins can generate interrupts.

Bits	Name	Access	Reset	Description
31:0	IE	RW	0	Interrupt enable per

Bits	Name	Access	Reset	Description
				pin (1=enabled)

23.7 GPIO_INT_TYPE (0x014)

Interrupt type select. Chooses edge or level sensitivity.

Bits	Name	Access	Reset	Description
31:0	TYPE	RW	0	Type per pin (0=edge, 1=level)

23.8 GPIO_INT_POLARITY (0x018)

Interrupt polarity select.

Bits	Name	Access	Reset	Description
31:0	POL	RW	0	Polarity per pin

For edge mode: 0=falling, 1=rising For level mode: 0=active-low, 1=active-high

---

23.9 GPIO_INT_BOTH (0x01C)

Both-edge interrupt enable. Only applicable in edge mode.

Bits	Name	Access	Reset	Description
31:0	BOTH	RW	0	Both edges per pin (1=both edges)

When set, GPIO_INT_POLARITY is ignored for that pin.

23.10 GPIO_INT_STATUS (0x020)

Interrupt status register. Shows pending interrupts.

Bits	Name	Access	Reset	Description
31:0	STATUS	W1C	0	Interrupt pending per pin

Access: Read returns current status. Write 1 clears the bit.

23.11 Address Calculation

For system address:

```
Register_Address = BASE_ADDR + WINDOW_OFFSET + Register_Offset

Where:
  BASE_ADDR = 0xFEC00000 (RLB base)
  WINDOW_OFFSET = 0x7000 (GPIO window)
  Register_Offset = value from table above

Example:
  GPIO_INPUT = 0xFEC00000 + 0x7000 + 0x00C = 0xFEC0700C
```

Back to: GPIO Specification Index