27. PIO Core

27.1. Core Overview

The parallel input/output (PIO) core with Avalon interface provides a memory-mapped interface between an Avalon Memory-Mapped (Avalon-MM) slave port and general-purpose I/O ports. The I/O ports connect either to on-chip user logic, or to I/O pins that connect to devices external to the FPGA.

The PIO core provides easy I/O access to user logic or external devices in situations where a "bit banging" approach is sufficient. Some example uses are:

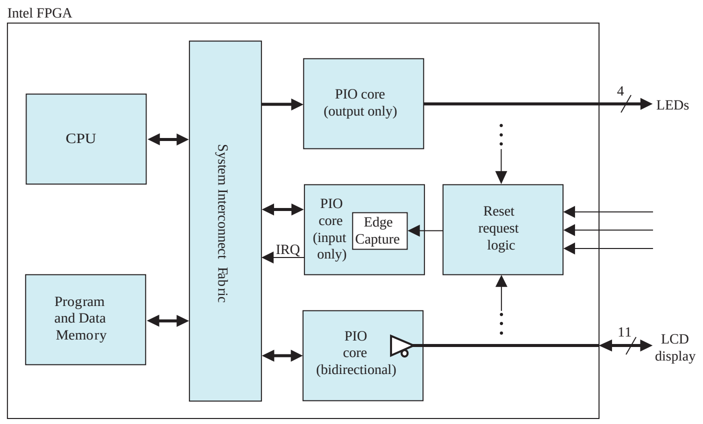
- Controlling LEDs
- Acquiring data from switches
- Controlling display devices
- Configuring and communicating with off-chip devices, such as application-specific standard products (ASSP)

  The PIO core interrupt request (IRQ) output can assert an interrupt based on input signals.

27.2. Functional Description

Each PIO core can provide up to 32 I/O ports. An intelligent host such as a microprocessor controls the PIO ports by reading and writing the register-mapped Avalon-MM interface. Under control of the host, the PIO core captures data on its inputs and drives data to its outputs. When the PIO ports are connected directly to I/O pins, the host can tristate the pins by writing control registers in the PIO core. The example below shows a processor-based system that uses multiple PIO cores to drive LEDs, capture edges from on-chip reset-request control logic, and control an off-chip LCD display.

Figure 79. System Using Multiple PIO Cores



When integrated into an Platform Designer-generated system, the PIO core has two user-visible features:

- A memory-mapped register space with four registers: `data`, `direction`, `interruptmask`, and `edgecapture`
- 1 to 32 I/O ports

  The I/O ports can be connected to logic inside the FPGA, or to device pins that connect to off-chip devices. The registers provide an interface to the I/O ports via the Avalon-MM interface. See **Register Map for the PIO Core** table for a description of the registers.

27.2.1. Data Input and Output

The PIO core I/O ports can connect to either on-chip or off-chip logic. The core can be configured with inputs only, outputs only, or both inputs and outputs. If the core is used to control bidirectional I/O pins on the device, the core provides a bidirectional mode with tristate control.

The hardware logic is separate for reading and writing the data register. Reading the data register returns the value present on the input ports (if present). Writing data affects the value driven to the output ports (if present). These ports are independent; reading the data register does not return previously-written data.

27.2.2. Edge Capture

The PIO core can be configured to capture edges on its input ports. It can capture low-to-high transitions, high-to-low transitions, or both. Whenever an input detects an edge, the condition is indicated in the `edgecapture` register. The types of edges detected is specified during IP generation in the Platform Designer, and cannot be changed via the registers.

27.2.3. IRQ Generation

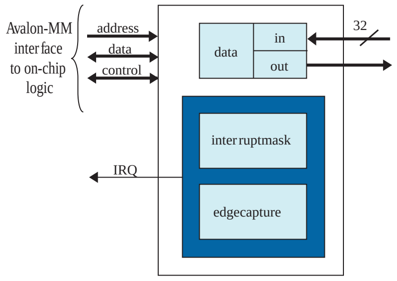
The PIO core can be configured to generate an IRQ on certain input conditions. The IRQ conditions can be either:

- Level-sensitive—The PIO core hardware can detect a high level. A NOT gate can be inserted external to the core to provide negative sensitivity.
- Edge-sensitive—The core's edge capture configuration determines which type of edge causes an IRQ

  Interrupts are individually maskable for each input port. The interrupt mask determines which input port can generate interrupts.

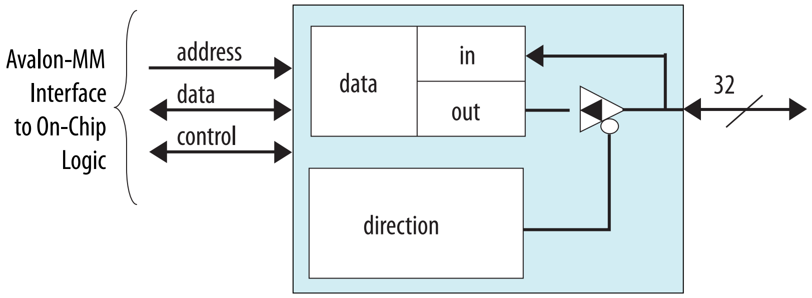
27.3. Example Configurations

Figure 80. PIO Core with Input Ports, Output Ports, and IRQ Support



The block diagram below shows the PIO core configured in bidirectional mode, without support for IRQs.

Figure 81. PIO Cores with Bidirectional Ports



27.3.1. Avalon-MM Interface

The PIO core's Avalon-MM interface consists of a single Avalon-MM slave port. The slave port is capable of fundamental Avalon-MM read and write transfers. The Avalon-MM slave port provides an IRQ output so that the core can assert interrupts.

27.4. Configuration

The following sections describe the available configuration options.

27.4.1. Basic Settings

The **Basic Settings** page allows you to specify the width, direction and reset value of the I/O ports.

27.4.1.1. Width

The width of the I/O ports can be set to any integer value between 1 and 32.

27.4.1.2. Direction

You can set the port direction to one of the options shown below.

Table 272. Direction Settings

Setting	Description
Bidirectional (tristate) ports	In this mode, each PIO bit shares one device pin for driving and capturing data. The direction of each pin is individually selectable. To tristate an FPGA I/O pin, set the direction to input.
Input ports only	In this mode the PIO ports can capture input only.
Output ports only	In this mode the PIO ports can drive output only.
Both input and output ports	In this mode, the input and output ports buses are separate, unidirectional buses of n bits wide.

27.4.1.3. Output Port Reset Value

You can specify the reset value of the output ports. The range of legal values depends on the port width.

27.4.1.4. Output Register

The option **Enable individual bit set/clear output register** allows you to set or clear individual bits of the output port. When this option is turned on, two additional registers—`outset` and `outclear`—are implemented. You can use these registers to specify the output bit to set and clear.

27.4.2. Input Options

The **Input Options** page allows you to specify edge-capture and IRQ generation settings. The **Input Options** page is not available when **Output ports only** is selected on the **Basic Settings** page.

27.4.2.1. Edge Capture Register

Turn on **Synchronously capture** to include the edge capture register, `edgecapture`, in the core. The edge capture register allows the core to detect and generate an optional interrupt when an edge of the specified type occurs on an input port. The user must further specify the following features:

- Select the type of edge to detect:
  - **Rising Edge**
  - **Falling Edge**
  - **Either Edge**
- Turn on **Enable bit-clearing for edge capture register** to clear individual bit in the edge capture register. To clear a given bit, write 1 to the bit in the edge capture register.

27.4.2.2. Interrupt

Turn on **Generate IRQ** to assert an IRQ output when a specified event occurs on input ports. The user must further specify the cause of an IRQ event:

- **Level—** The core generates an IRQ whenever a specific input is high and interrupts are enabled for that input in the `interruptmask` register.
- **Edge**— The core generates an IRQ whenever a specific bit in the edge capture register is high and interrupts are enabled for that bit in the `interruptmask` register.

  When **Generate IRQ** is off, the `interruptmask` register does not exist.

27.4.3. Simulation

The **Simulation** page allows you to specify the value of the input ports during simulation. Turn on **Hardwire PIO inputs in test bench** to set the PIO input ports to a certain value in the testbench, and specify the value in **Drive inputs to** field.

27.5. Software Programming Model

This section describes the software programming model for the PIO core, including the register map and software constructs used to access the hardware. For Nios II processor users, Intel provides the HAL system library header file that defines the PIO core registers. The PIO core does not match the generic device model categories supported by the HAL, so it cannot be accessed via the HAL API or the ANSI C standard library.

The Nios II Embedded Design Suite (EDS) provides several example designs that demonstrate usage of the PIO core. In particular, the `count_binary.c` example uses the PIO core to drive LEDs, and detect button presses using PIO edge-detect interrupts.

27.5.1. Software Files

The PIO core is accompanied by one software file, `altera_avalon_pio_regs.h`. This file defines the core's register map, providing symbolic constants to access the hardware.

27.5.2. Register Map

An Avalon-MM master peripheral, such as a CPU, controls and communicates with the PIO core via the four 32-bit registers, shown below. The table assumes that the PIO core's I/O ports are configured to a width of n bits.

Table 273. Register Map for the PIO Core

Offset	Register Name		R/W	(n-1)	...	2	1	0
0	`data`	read access	R	Data value currently on PIO inputs				
		write access	W	New value to drive on PIO outputs				
1	`direction` (1)		R/W	Individual direction control for each I/O port. A value of 0 sets the direction to input; 1 sets the direction to output.				

continued...

Send Feedback

Offset	Register Name	R/W	(n-1)	...	2	1	0
2	`interruptmask` (1)	R/W	IRQ enable/disable for each input port. Setting a bit to 1 enables interrupts for the corresponding port.				
3	`edgecapture` (1) , (2)	R/W	Edge detection for each input port.				
4	`outset`	W	Specifies which bit of the output port to set. Outset value is not stored into a physical register in the IP core. Hence it's value is not reserve for future use.				
5	`outclear`	W	Specifies which output bit to clear. Outclear value is not stored into a physical register in the IP core. Hence it's value is not reserve for future use.				

Note :
1. This register may not exist, depending on the hardware configuration. If a register is not present, reading the register returns an undefined value, and writing the register has no effect.
2. If the option **Enable bit-clearing for edge capture register** is turned off, writing any value to the `edgecapture` register clears all bits in the register. Otherwise, writing a 1 to a particular bit in the register clears only that bit.

27.5.2.1. data Register

Reading from `data` returns the value present at the input ports if the PIO core hardware is configured to input, or inout mode only. If the PIO core hardware is configured to output-only mode, reading from the `data` register returns the value present at the output ports. Whereas, if the PIO core hardware is configured to bidirectional mode, reading from data register returns value depending on the direction register value, setting to 1 returns value present at the output ports, setting to 0 returns undefined value.

Writing to `data` stores the value to a register that drives the output ports. If the PIO core hardware is configured in input-only mode, writing to `data` has no effect. If the PIO core hardware is in bidirectional mode, the registered value appears on an output port only when the corresponding bit in the `direction` register is set to 1 (output).

27.5.2.2. direction Register

The `direction` register controls the data direction for each PIO port, assuming the port is bidirectional. When bit n in `direction` is set to 1, port n drives out the value in the corresponding bit of the `data` register.

The `direction` register only exists when the PIO core hardware is configured in bidirectional mode. In input-only, output-only and inout mode, the `direction` register does not exist. In this case, reading `direction` returns an undefined value, writing `direction` has no effect. The mode (input, output, inout or bidirectional) is specified at system generation time, and cannot be changed at runtime.

After reset, all `direction` register bits are 0, so that all bidirectional I/O ports are configured as inputs. If those PIO ports are connected to device pins, the pins are held in a high-impedance state. In bi-directional mode, you will need to write to the `direction` register to change the direction of the PIO port (0-input, 1-output).

27.5.2.3. interruptmask Register

Setting a bit in the `interruptmask` register to 1 enables interrupts for the corresponding PIO input port. Interrupt behavior depends on the hardware configuration of the PIO core. See the **Interrupt Behavior** section.

The `interruptmask` register only exists when the hardware is configured to generate IRQs. If the core cannot generate IRQs, reading `interruptmask` returns an undefined value, and writing to `interruptmask` has no effect.

After reset, all bits of `interruptmask` are zero, so that interrupts are disabled for all PIO ports.

27.5.2.4. edgecapture Register

Bit *n* in the `edgecapture` register is set to 1 whenever an edge is detected on input port n. An Avalon-MM master peripheral can read the `edgecapture` register to determine if an edge has occurred on any of the PIO input ports. If the edge capture register bit has been previously set, `in_port` toggling activity will not change value.

If the option Enable bit-clearing for the edge capture register is turned off, writing any value to the `edgecapture` register clears all bits in the register. Otherwise, writing a 1 to a particular bit in the register clears only that bit.

The type of edge(s) to detect is specified during IP generation in Platform Designer. The `edgecapture` register only exists when the hardware is configured to capture edges. If the core is not configured to capture edges, reading from `edgecapture` returns an undefined value, and writing to `edgecapture` has no effect.

27.5.2.5. outset and outclear Register

You can use the `outset` and `outclear` registers to set and clear individual bits of the output port. For example, to set bit 6 of the output port, write `0x40` to the `outset` register. Writing `0x08` to the `outclear` register clears bit 3 of the output port.

These registers are only present when the option **Enable individual bit set/clear output register** is turned on. Outset and `outclear` registers are not physical registers inside the IP core, hence the output port value will only be affected by the current update outset value or current update outclear value only.

27.5.3. Interrupt Behavior

The PIO core outputs a single IRQ signal that can connect to any master peripheral in the system. The master can read either the `data` register or the `edgecapture` register to determine which input port caused the interrupt.

When the hardware is configured for level-sensitive interrupts, the IRQ is asserted whenever corresponding bits in the `data` and `interruptmask` registers are 1. When the hardware is configured for edge-sensitive interrupts, the IRQ is asserted whenever corresponding bits in the `edgecapture` and `interruptmask` registers are 1. The IRQ remains asserted until explicitly acknowledged by disabling the appropriate bit(s) in `interruptmask`, or by writing to `edgecapture`.

27.5.4. Software Files

The PIO core is accompanied by the following software file. This file provide low-level access to the hardware. Application developers should not modify the file.

Send Feedback

- `altera_avalon_pio_regs.h`—This file defines the core's register map, providing symbolic constants to access the low-level hardware. The symbols in this file are used by device driver functions.

27.6. PIO Core Revision History

Document Version	Intel Quartus Prime Version	Changes
2018.05.07	18.0	Implemented editorial enhancements.

Date	Version	Changes
December 2015	2015.12.16	Updated "edgecapture Register" section
June 2015	2015.06.12	• Updated "Register Map" section • Updated "data Register" section • Updated "direction Register" section • Updated "outset and outclear Register" section
July 2014	2014.07.24	Removed mention of SOPC Builder, updated to Platform Designer
December 2013	v13.1.0	Updated note (2) in Register map for PIO Core Table
December 2010	v10.1.0	Removed the “Device Support”, “Instantiating the Core in SOPC Builder”, and “Referenced Documents” sections
July 2010	v10.0.0	No change from previous release
November 2009	v9.1.0	No change from previous release
March 2009	v9.0.0	Added a section on new registers, `outset` and `outclear`
November 2008	v8.1.0	Changed to 8-1/2 x 11 page size. Added the description for **Output Port Reset Value** and **Simulation** parameters
May 2008	v8.0.0	No change from previous release