11. PIO Core with Avalon Interface

NII51007-6.0.0

Core Overview

The parallel input/output (PIO) core provides a memory-mapped interface between an Avalon® 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:

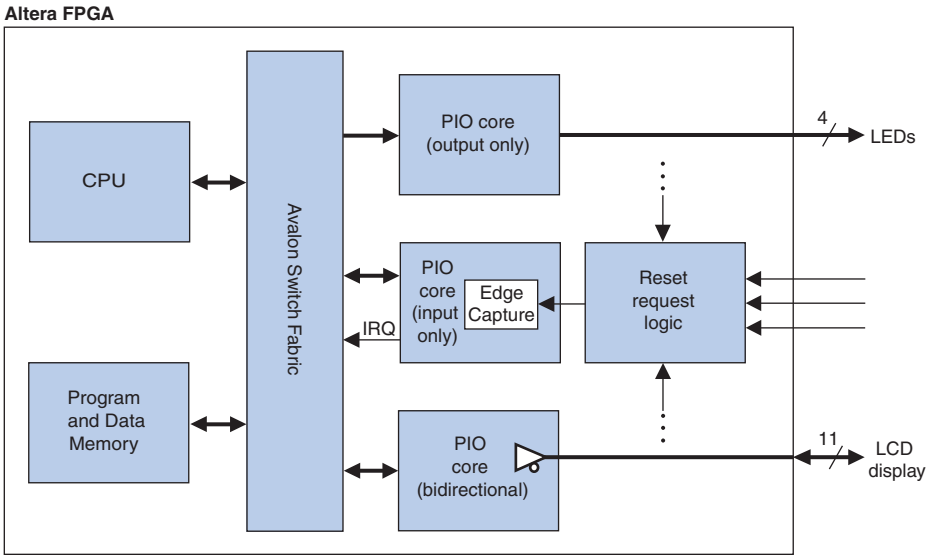
- Controlling LEDs
- Acquiring data from switches
- Controlling display devices
- Configuring and communicating with off-chip devices, such as application-specific standard products (ASSP)

The PIO core interrupt request (IRQ) output can assert an interrupt based on input signals. The PIO core is SOPC Builder ready and integrates easily into any SOPC Builder-generated system.

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 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. Figure 11–1 shows an example of a processor-based system that uses multiple PIO cores to blink LEDs, capture edges from on-chip reset-request control logic, and control an off-chip LCD display.

Figure 11–1. An Example System Using Multiple PIO Cores



When integrated into an SOPC Builder-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 interface. See Table 11–2 on page 11–7 for a description of the registers. Some registers are not necessary in certain hardware configurations, in which case the unnecessary registers do not exist. Reading a non-existent register returns an undefined value, and writing a non-existent register has no effect.

Data Input & 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 will be 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.

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 type of edges to detect is specified at system generation time, and cannot be changed via the registers.

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.

Example Configurations

Figure 11–2 shows a block diagram of the PIO core configured with input and output ports, as well as support for IRQs.

Figure 11–2. PIO Core with Input & Output Ports & with IRQ Support

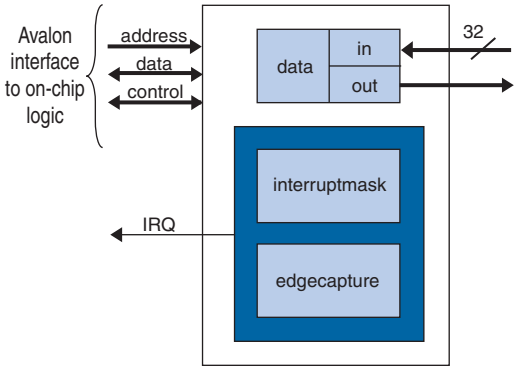
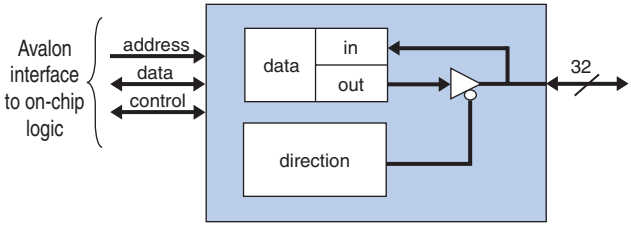


Figure 11–3 shows a block diagram of the PIO core configured in bidirectional mode, without support for IRQs.

Figure 11–3. PIO Core with Bidirectional Ports



Avalon Interface

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

Instantiating the PIO Core in SOPC Builder

The hardware feature set is configured via the PIO core's SOPC Builder configuration wizard. The following sections describe the available options.

The configuration wizard has two tabs, **Basic Settings** and **Input Options**.

Basic Settings

The **Basic Settings** tab allows the designer to specify the width and direction of the I/O ports.

- The **Width** setting can be any integer value between 1 and 32. For a value of *n*, the I/O ports become *n*-bits wide.
- The **Direction** setting has four options, as shown in Table 11–1.

Table 11–1. 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.

Input Options

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

Edge Capture Register

When the **Synchronously capture** option is turned on, the PIO core contains the edge capture register, `edgecapture`. The user must further specify what type of edge(s) to detect:

- **Rising Edge**
- **Falling Edge**
- **Either Edge**

The edge capture register allows the core to detect and (optionally) generate an interrupt when an edge of the specified type occurs on an input port.

When the **Synchronously capture** option is turned off, the `edgecapture` register does not exist.

Interrupt

When the **Generate IRQ** option is turned on, the PIO core is able 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 the **Generate IRQ** option is turned off, the `interruptmask` register does not exist.

Device & Tools Support

The PIO core supports all Altera® FPGA families.

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, Altera 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.

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 low-level hardware.

Legacy SDK Routines

The PIO core is supported by the legacy SDK routines for the first-generation Nios processor. For details on these routines, refer to the PIO documentation that accompanied the first-generation Nios processor. For details on upgrading programs based on the legacy SDK to the HAL system library API, refer to *AN 350: Upgrading Nios Processor Systems to the Nios II Processor*.

Register Map

An Avalon master peripheral, such as a CPU, controls and communicates with the PIO core via the four 32-bit registers, shown in Table 11–2. The table assumes that the PIO core's I/O ports are configured to a width of *n* bits.

Table 11–2. 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.				
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.				

Notes to Table 11–2:

(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) Writing any value to `edgecapture` clears all bits to 0.

data Register

Reading from `data` returns the value present at the input ports. If the PIO core hardware is configured in output-only mode, reading from `data` returns an 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).

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

After reset, all bits of direction 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.

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 "Interrupt Behavior" on page 11–9.

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.

edgecapture Register

Bit *n* in the `edgecapture` register is set to 1 whenever an edge is detected on input port *n*. An Avalon master peripheral can read the `edgecapture` register to determine if an edge has occurred on any of the PIO input ports. Writing any value to `edgecapture` clears all bits in the register.

The type of edge(s) to detect is fixed in hardware at system generation time. 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.

Interrupt Behavior

The PIO core outputs a single interrupt-request (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`.

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.

- **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.