Nios PIO

January 2003, Version 3.0 Data Sheet

General Description

The Nios® parallel input/output (PIO) module is included in the Nios Development Kit, Stratix™ Edition. It is a 1- to 32-bit parallel I/O (input, output, and edge-capture). The PIO is an Altera® SOPC Builder library component with available system choices to define logic and interface signals with the Stratix device on the Nios development board, Stratix edition. The PIO's Verilog HDL or VHDL source code is available for development and includes the necessary software subroutines for easy system integration.

Functional Description

A parallel input/output (PIO) module is a memory-mapped interface between software and user-defined logic. The PIO has two distinct applications:

- Providing a PIO interface between software and user-defined logic that also resides within the same device.
- Providing a PIO interface between software and user peripheral logic that resides outside the device.

The following figures illustrate various 4-port PIO configurations. In Figure 1, the tri-state PIO includes bidirectional input/output ports. In Figure 2 on page 2, the output-only PIO includes output-only ports. In Figure 3 on page 2, the input-only PIO includes input-only ports.

Figure 1. Tri-State PIO

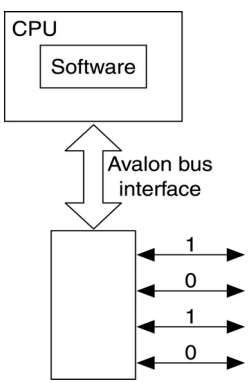


Figure 2. Output-Only PIO

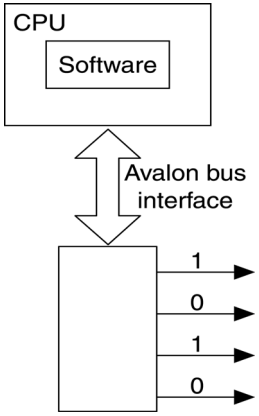


Figure 3. Input-Only PIO

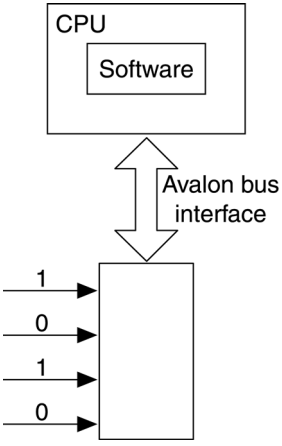


Table 1 lists and describes the PIO registers.

PIO Registers

Table 1. PIO Register Map

A1..A0	Register Name		R/W	Variable Size—1 to 32 bits
0	data	read	RO	Data value currently on PIO inputs
		write	WO	New value to drive on PIO outputs
1	direction		RW	Data direction (optional): Individual control for each PIO bit
2	interruptmask		RW	Interrupt mask (optional): Per-bit IRQ enable/disable
3	edgecapture[1]		RW	Edge capture (optional): Per-bit synchronous edge detect and hold

Note
(1) A write operation to the `edgecapture` register clears all bits in register 0.

data Register

For an input-only PIO (when the PTF parameters `has_out` and `has_tri` are both set to 0), writing to the `data` register has no effect. For an output-only PIO (when `has_in` and `has_tri` are both set to 0), reading from the `data` register produces an undefined result.

direction Register

When the PTF parameter `has_tri` is set to 1, both the PIO's input ports and output ports connect to one pin with tri-state control on the Stratix device. The `direction` register controls the data direction for each PIO bit. When `direction` is set to 1, the corresponding PIO port's data direction is out. When `direction` is set to 0, the data direction is in.

At system reset, all `direction` register bits are set to 0. When `has_tri` is 0, this register does not exist. For details, see "has_tri" on page 6.

interruptmask Register

When a bit in the `interruptmask` register is set to 1, interrupts are enabled for the corresponding PIO port. The PTF parameter `irq_type` determines how interrupts are generated based on PIO inputs. For details, see "irq_type" on page 7. If the PTF setting `irq_type` is `NONE`, this register does not exist, because there are no interrupts to enable.

At system reset, the `interruptmask` register is all zeros, and interrupts are disabled for all PIO ports.

edgecapture Register

If the PTF parameter edge_type is set to RISING, FALLING, or ANY, the edgecapture register sets a bit to 1 to indicate when an edge was detected on the corresponding PIO input port. The edgecapture register's behavior depends on the edge_type value. For details, see "edge_type" on page 7. If edge_type is set to NONE, the edgecapture register does not exist.

A write operation to the edgecapture register clears all bits in the data register.

Software Data Structure

```
typedef volatile struct
{
   int np_piodata;          // read/write, up to 32 bits
   int np_piodirection;     // write/readable, up to 32 bits,
                            // 1->output bit
   int np_piointerruptmask; // write/readable, up to 32 bits,
                            // 1->enable interrupt
   int np_pioedgecapture;   // read, up to 32 bits,
                            // cleared by any write
}  np_pio;
```

Figure 4. Example: Direct access to PIO

```
void TurnOnLEDs(void)
{
  // the reference design has a PIO named na_led_pio
  // that controls two LEDs on the development board

   na_led_pio->np_piodirection = 3; // Set direction: output
   na_led_pio->np_piodata = 0;      // both LEDs off
   nr_delay(1000);                  // wait 1 second
   na_led_pio->np_piodata = 1;      // turn on first led
   nr_delay(1000);                  // wait 1 second
   na_led_pio->np_piodata = 3;      // both LEDs on
}
```

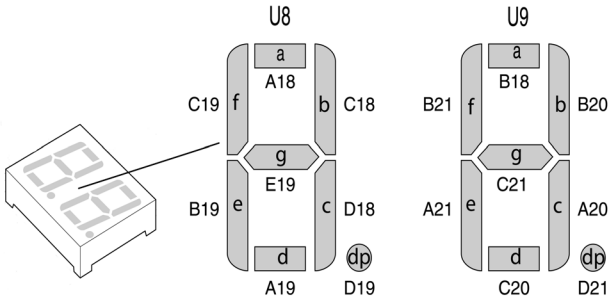
Software Subroutine

The `nr_pio_showhex` subroutine is available in the Nios library (**lib** folder in the custom SDK) when one or more PIO peripherals are present in the Nios system. This function is declared in the include file **nios.h**.

nr_pio_showhex

This subroutine assumes a 16-bit wide PIO named `na_seven_seg_pio` is attached to a two-digit seven-segment display, in which segments are illuminated when the corresponding bits are set to `0`. PIO bits are assigned to the seven-segment display elements as shown in Figure 5:

Figure 5. Dual Seven-Segment Display



Syntax

```
void nr_pio_showhex(int value);
```

Parameter

The `value` parameter indicates the data to be sent to the seven-segment display.

Example

```
#include "nios.h"
void main(void)
{
    int c;

    printf("Please enter a character:\n");

    while((c = nr_uart_rxchar(0)) == -1); // wait for valid input

    nr_pio_showhex(c);
    printf("Your character is:\t%c, in hex:0x%2x\m", c, c);
}
```

PTF Assignments

Table 2 lists the PIO's PTF parameters. Detailed descriptions follow the table.

Table 2. PIO PTF Parameters

Parameter	Section(1)	Type	Allowed Values	Default
has_tri	M/WSA	Boolean	1, 0	0
has_in	M/WSA	Boolean	1, 0	0
has_out	M/WSA	Boolean	1, 0	1
irq_type	M/WSA	String	NONE, LEVEL, EDGE	NONE
edge_type	M/WSA	String	NONE, RISING, FALLING, ANY	NONE
Data_Width	M/SBI	Integer	1 .. 32	8

Note
(1) The Section column describes the parameter's location in the PTF:
M/WSA = `MODULE/WIZARD_SCRIPT_ARGUMENTS`
M/SBI = `MODULE/SYSTEM_BUILDER_INFO`

has_tri

When `has_tri` is set to 1, the PIO has combined bidirectional input/output ports instead of separate input ports and separate output ports.

has_in

When `has_in` is set to 1, the specified number of separate input signals are available.

has_out

When `has_out` is set to 1, the specified number of separate output signals are available.

`has_in` and `has_out` can both be set to 1 at the same time to allow for separate input and output signals. For example, if `Data_Width` = 4, `has_in` = 1, and `has_out` = 1, a peripheral with eight I/O ports is created: four dedicated input and four dedicated output.

If `has_tri` is set to 1, both the `has_in` and `has_out` values are ignored.

irq_type

The `irq_type` assignment specifies when an interrupt request is generated. The `irq_type` values are shown in Table 3:

Table 3. irq_type Values

Value	Description
LEVEL	Generate an interrupt when a PIO input logic level is detected.
EDGE	Generate an interrupt when a PIO input logic level transition is detected. This value can only be used if the `edge_type` parameter is set to RISING, FALLING, or ANY.
NONE	The PIO does not generate an interrupt; the `interruptmask` register does not exist.

When `irq_type` is set to LEVEL or EDGE, the PIO includes both an IRQ pin to the system and the internal `interruptmask` register.

edge_type

The `edge_type` assignment specifies the transition type to be detected in the `edgecapture` register during an input to the PIO (see "edgecapture Register" on page 4). The allowed `edge_type` values are shown in Table 4:

Table 4. edge_type Values

Value	Description
RISING	Detect the transition from logical 0 to logical 1.
FALLING	Detect the transition from logical 1 to logical 0.
ANY	Detect any logic level transition.
NONE	Detect only the logic level; the `edgecapture` register does not exist.

Data_Width

The `Data_Width` assignment specifies the number of bits in the PIO register. If the `has_tri` parameter is set to 0, the `Data_Width` assignment allows between 1 and 32 input ports and between 1 and 32 output ports. If `has_tri` is set to 1, the `Data_Width` assignment allows a bidirectional interface.

101 Innovation Drive
San Jose, CA 95134
(408) 544-7000
http://www.altera.com
Applications Hotline:
(800) 800-EPLD
Literature Services:
lit_req@altera.com