Parallel Port for Altera DE-Series Boards

For Quartus Prime 16.0

1 Core Overview

The Parallel Port IP core provides a simple interface for general purpose I/O. This is a version of the Altera's *PIO Core* adapted for use with Altera's DE-series boards.

2 Functional Description

The Parallel Port IP core can provide up to 32 I/O ports. It captures data on its inputs and drives data to its outputs, which provides easy communication with user logic and external devices. Some example uses include:

- Controlling LEDs
- Acquiring data from switches
- Controlling display devices

3 Instantiating the Core in Qsys Builder

Designers use the Parallel Port IP core's configuration wizard in Qsys to specify the features. The following describes the available options, shown in Figure 1.

- **DE-series boards** — allows users to specify the target board.
- **Create custom parallel port** — allows users to select between using one of the presets or create a custom parallel port.
- **Presets**
  - **I/0 device** — allows users to select a preset I/O device.
  - **LEDs color** — allows users to specify a set of LEDs that are present of some DE-series boards.
  - **Seven Segment Digits** — allows users to specify a set of 7-segment displays when more than four are present on the selected DE-series board.
  - **Expansion Header** — allows users to specify expansion headers that are present on some DE-series boards.
- **Basic Settings (Presets)** — This section is only visible for preset parallel ports and it shows the data width and port direction for the selected preset configuration.

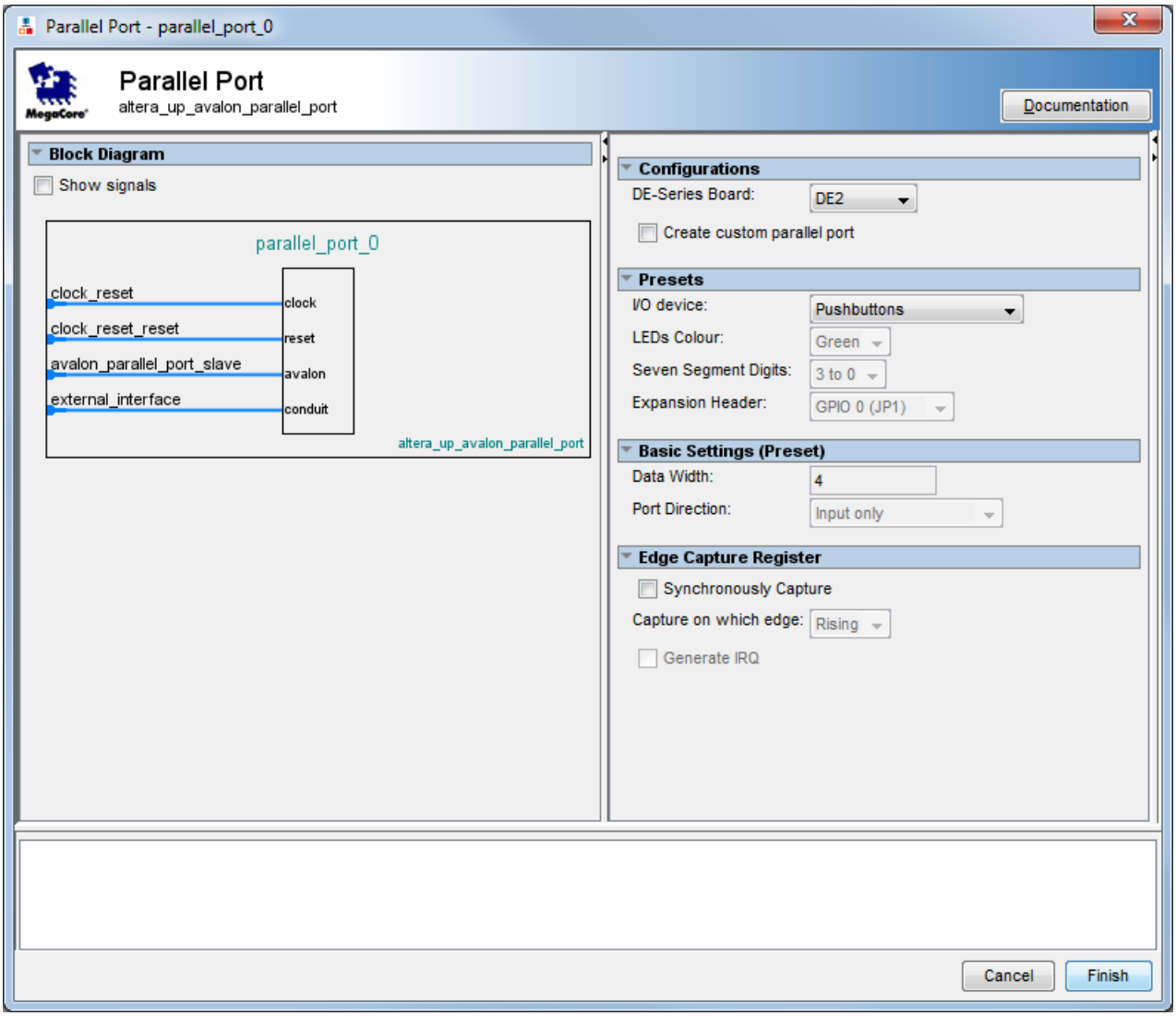

Figure 1. Parallel port's Qsys wizard.

- **Basic Settings** — This section is only visible for custom parallel ports.
  - **Data Width** — The width of the parallel port can be set to any integer value between 1 and 32.
  - **Port Direction** — You can set the parallel port's direction to one of the options shown in Table 1.
- **Edge Capture Register**
  - **Synchronously Capture** — Enables the edge capture register.
  - **Capture on which Edge** — allows users to select the type of edge to capture. This can either be rising edge, falling edge or both edges.
  - **Generate IRQ** — allows users to enable interrupts, so that an IRQ will be asserted when an edge is captured.

Table 1. Direction Settings	
Setting	**Description**
Bidirectional (tristate) ports	In this mode, each parallel port bit can either be used as input or output, and the direction of each bit is individually selectable.
Input ports only	In this mode the parallel port can only capture input.
Output ports only	In this mode the parallel port can only drive output.

4 Software Programming Model

4.1 Register Map

The Parallel Port IP core can have up to four registers, as shown in Table 2. These registers have a configurable data width, n, which is set through the Qsys component wizard.

Not all of these registers are generated in a given parallel port interface. For example, the *Direction* register is included only when a bidirectional interface is specified.

Table 2. Parallel Port register map				
Offset in bytes	**Register name**		**Read/Write**	**Bits** $(n-1)\ldots 0$
0	`data`	Input	R	Data value currently on Parallel Port inputs.
		Output	W	New value to drive on Parallel Port outputs.
4	`direction`		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.
8	`interruptmask`		R/W	IRQ enable/disable for each input port. Setting a bit to 1 enables interrupts for the corresponding port.
12	`edgecapture`		R/W	Edge detection for each input port.

Notes on Table 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.

4.1.1 *Data* Register

This register holds the n bits of data that are transferred between the Parallel Port interface and the Nios II processor. It can be implemented as an input, output, or a bidirectional register.

Reading from the *data* register returns the value present at the input ports and/or the last data written to an output port.

Writing to the *data* register stores the value to a register that drives the output ports. If the core is configured in input-only mode, writing to *data* register has no effect. 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).

4.1.2 *Direction* Register

The *direction* register is enabled only when the Parallel Port is set to bidirectional, and is used to specify data transfer direction for each bit. When bit *n* in the *direction* register is set to 1, the corresponding bit of the *data* register is set to be output. When it si set to 0, the corresponding port bit is set as input, and is held at high-impedance state until driven by an outside source. In the case, when the *direction* register does not exist, reading it returns an undefined value, writing the *direction* register has no effect.

After reset, all bits of the *direction* register are 0, so that all bidirectional I/O ports are configured as inputs.

4.1.3 *Interruptmask* Register

The *interruptmask* register only exists when the hardware is configured to generate IRQs. Setting a bit in the *interruptmask* register to 1 enables interrupts for the corresponding bit in the parallel port's *edgecapture* register. If the core cannot generate IRQs, reading the *interruptmask* register returns an undefined value, and writing to the *interruptmask* register has no effect.

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

4.1.4 *Edgecapture* Register

When the parallel port is configured to detect edges, the *edgecapture* register is created to indicate on which bit(s) of the port an edge has occurred. If bit *n* in the *edgecapture* register is set to 1 whenever an edge is detected on input port *n*.

The type of edge(s) to detect is specified in hardware at system generation time. The *edgecapture* register only exists when the hardware is configured to capture edges. An Avalon-MM master peripheral can read the *edgecapture* register to determine if an edge has occurred on any of the parallel input bits.If the core is not configured to capture edges, reading from the *edgecapture* register returns an undefined value, and writing to edgecapture has no effect.

4.2 Device Driver for the Nios II Processor

The Parallel Port core is packaged with C-language functions accessible through the hardware abstraction layer (HAL). These functions implement basic operations for the Parallel Port.

To use the functions, the C code must include the statement:

```
#include "altera_up_avalon_parallel_port.h"
```

4.2.1 alt_up_parallel_port_open_dev

Prototype: `alt_up_parallel_port_dev* alt_up_parallel_port_open_dev(const char *name)`
Include: `<altera_up_avalon_parallel_port.h>`
Parameters: `name` – the parallel port name. For example, if the parallel port name in Qsys is "green_leds", then *name* should be "/dev/green_leds"
Returns: The corresponding device structure, or NULL if the device is not found.
Description: Open the parallel port device specified by *name* .

4.2.2 alt_up_parallel_port_read_data

Prototype: `unsigned int alt_up_parallel_port_read_data( alt_up_parallel_port_dev *parallel_port)`
Include: `<altera_up_avalon_parallel_port.h>`
Parameters: `parallel_port` – struct for the parallel port device .
Returns: data – The data read for the parallel port.
Description: Read from the data register of the parallel port.

4.2.3 alt_up_parallel_port_write_data

Prototype: `void alt_up_parallel_port_write_data(alt_up_parallel_port_dev *parallel_port, unsigned data)`
Include: `<altera_up_avalon_parallel_port.h>`
Parameters: `parallel_port` – struct for the parallel port device.
`data` – The data to be written to the parallel port.
Description: Write to the data register of the parallel port.

4.2.4 alt_up_parallel_port_read_direction

Prototype: `unsigned int alt_up_parallel_port_read_direction( alt_up_parallel_port_dev *parallel_port)`
Include: `<altera_up_avalon_parallel_port.h>`
Parameters: `parallel_port` – struct for the parallel port device.
Returns: direction – The direction read for the parallel port.
Description: Read from the direction register of the parallel port.

4.2.5 alt_up_parallel_port_set_port_direction

Prototype: `void alt_up_parallel_port_set_port_direction( alt_up_parallel_port_dev *parallel_port, unsigned direction)`
Include: `<altera_up_avalon_parallel_port.h>`
Parameters: `parallel_port` – struct for the parallel port device.
Description: Set the direction register of the parallel port.

4.2.6 alt_up_parallel_port_set_all_bits_to_input

Prototype: `void alt_up_parallel_port_set_all_bits_to_input(alt_up_parallel_port_dev *parallel_port)`
Include: `<altera_up_avalon_parallel_port.h>`
Parameters: `parallel_port` – struct for the parallel port device.
Description: Set the direction of all bits of the parallel port to be inputs.

4.2.7 alt_up_parallel_port_set_all_bits_to_output

Prototype: `void alt_up_parallel_port_set_all_bits_to_output(alt_up_parallel_port_dev *parallel_port)`
Include: `<altera_up_avalon_parallel_port.h>`
Parameters: `parallel_port` – struct for the parallel port device.
Description: Set the direction of one bits of the parallel port to be outputs.

4.2.8 alt_up_parallel_port_set_bit_to_input

Prototype: `void alt_up_parallel_port_set_bit_to_input(alt_up_parallel_port_dev *parallel_port, unsigned int bit)`
Include: `<altera_up_avalon_parallel_port.h>`
Parameters: `parallel_port` – struct for the parallel port device.
`bit` – The bit of the parallel port to be set as an input.
Description: Set the direction of one bit of the parallel port to be input.

4.2.9 alt_up_parallel_port_set_bit_to_output

Prototype: `void alt_up_parallel_port_set_bit_to_output(alt_up_parallel_port_dev *parallel_port, unsigned int bit)`
Include: `<altera_up_avalon_parallel_port.h>`
Parameters: `parallel_port` – struct for the parallel port device.
`bit` – The bit of the parallel port to be set as an output.
Description: Set the direction of one bit of the parallel port to be output.

4.2.10 alt_up_parallel_port_read_interrupt_mask

Prototype: `unsigned int alt_up_parallel_port_read_interrupt_mask(alt_up_parallel_port_dev *parallel_port)`
Include: `<altera_up_avalon_parallel_port.h>`
Parameters: `parallel_port` – struct for the parallel port device.
Returns: data – The current interrupt mask of the parallel port.
Description: Read from the interrupt mask register of the parallel port.

4.2.11 alt_up_parallel_port_set_interrupt_mask

Prototype: `void alt_up_parallel_port_set_interrupt_mask(`
`alt_up_parallel_port_dev *parallel_port,`
`unsigned mask)`
Include: `<altera_up_avalon_parallel_port.h>`
Parameters: `parallel_port` – struct for the parallel port device.
`mask` – The interrupt mask to be set in the parallel port.
Description: Set the interrupt mask register of the parallel port.

4.2.12 alt_up_parallel_port_read_edge_capture

Prototype: `unsigned int alt_up_parallel_port_read_edge_capture(`
`alt_up_parallel_port_dev *parallel_port)`
Include: `<altera_up_avalon_parallel_port.h>`
Parameters: `parallel_port` – struct for the parallel port device.
Returns: data – The current edge capture register of the parallel port.
Description: Read from the edge capture register of the parallel port.

4.2.13 alt_up_parallel_port_read_interrupt_pending

Prototype: `unsigned int alt_up_parallel_port_read_interrupt_pending(`
`alt_up_parallel_port_dev *parallel_port)`
Include: `<altera_up_avalon_parallel_port.h>`
Parameters: `parallel_port` – struct for the parallel port device.
Returns: pending – The interrupt pending bit field of the parallel port.
Description: Read interrupt pending information of the parallel port.

5 Appendix

The Parallel Port IP core differs from the PIO Core in the following ways:

- The Parallel Port core provides presets for ease of use with the simple I/Os on the DE-series boards, such as the LEDs and Switches.
- The PIO core provides an fourth direction option, named "Both input and output ports".
- The bit set/clear option of the PIO core is not available in the Parallel Port core.
- The Parallel Port core allows for edge-sensitive interrupts, while the PIO Core additionally allows for level-sensitive interrupts.
- For the Parallel Port core, the edge capture register is cleared when that register is read.
- The PIO core provides extra simulation support.
- The Parallel Port IP core includes device drivers that can be accessed by HAL.