PS/2 Core for Altera DE-Series Boards

For Quartus II 14.0

1 Core Overview

The PS/2 Serial Port on Altera DE-series boards is intended for connecting a keyboard or a mouse to the board. The PS/2 Core provides a connection to the PS/2 Serial Port and presents an easy-to-use communication interface to PS/2 peripherals.

2 Functional Description

The PS/2 Core handles the timing of the PS/2 Serial Data Transmission Protocol. The core comes with a 256-word fifo for storing data received from a PS/2 device. The core can be configures to provide either a memory-mapped or streaming interface to the input fifo and output port.

3 Instantiating the Core

The PS/2 core can be instantiated in a system using Qsys or as a standalone component from the IP Catalog within the Quartus II software. There are two parameters that need to be set, Avalon Type and Incoming clock rate. It is recommended to set the Avalon Type to Memory Mapped when connecting to a processor, otherwise set it to Streaming. The Incoming clock rate must be set to the value of the frequency of the clock that will be driving the PS2 Controller.

4 Software Programming Model

4.1 Register Map

When using this core with Qsys, device drivers control and communicate with the PS/2 Core through two 32-bit registers. Communication with the PS/2 peripheral is done by writing or reading the registers through the Avalon Slave Port when Memory-Mapped Avalon Type is selected for the core. Table 1 shows the details for the registers.

Table 1. PS/2 Core register map										
Offset in bytes	**Register Name**	**R/W/C**	**Bit description**							
			31...16	**15**	**14...11**	**10**	**9**	**8**	**7...1**	**0**
0	`data`	R/W	RAVAIL	RVALID	*(1)*				DATA	
4	`control`	R/C	*(1)*			CE	*(1)*	RI	*(1)*	RE

Notes on Table 1:

(1) Reserved. Read values are undefined. Write zero.

4.1.1 *data* Register

Table 2. data register bits			
Bit number	**Bit name**	**Read/Write/Clear**	**Description**
7...0	DATA	R/W	The value to transfer to/from the PS/2 core. When writing, the DATA field is interpreted as a command to be sent to the PS/2 device. When reading, the DATA field is data from the PS/2 device.
15	RVALID	R	Indicates whether the DATA field is valid. If RVALID=1, then the DATA field is valid, else the DATA is undefined.
31...16	RAVAIL	R	The number of data items remaining in the read FIFO (including this read).

4.1.2 *control* Register

Table 3. control register bits			
Bit number	**Bit name**	**Read/Write/Clear**	**Description**
0	RE	R/W	Interrupt-enable bit for read interrupts.
8	RI	R	Indicates that a read interrupt is pending.
10	CE	C	Indicates that an error occurred while trying to send a command to a PS/2 device.

4.2 Device Driver for the Nios II Processor

The PS/2 Core is packaged with C-language functions accessible through the the hardware abstraction layer (HAL) as listed below. These functions implement common operations that users need for the PS/2 Core.

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

```
#include "altera_up_avalon_ps2.h"
```

In addition, some sample functions for specific communication with the keyboard or mouse are also provided. They serve as a good starting point if the user wishes to develop more features with the PS/2 Port. To use the keyboard or mouse communication functions, the corresponding header files, `altera_up_ps2_keyboard.h` and `altera_up_ps2_mouse.h`, have to be included. These functions are described below.

4.3 PS/2 Port Documentation

4.3.1 PS2_DEVICE

Prototype:

```
typedef enum {
    PS2_MOUSE = 0;
    PS2_KEYBOARD = 1;
    PS2_UNKNOWN = 2;
} PS2_DEVICE;
```

Include: `<altera_up_avalon_ps2.h>`
Fields: `PS2_MOUSE` — Indicate that the device is a PS/2 Mouse.
`PS2_KEYBOARD` — Indicate that the device is a PS/2 Keyboard.
`PS2_UNKNOWN` — The program cannot determine what type the device is.

4.3.2 alt_up_ps2_init

Prototype: `void alt_up_ps2_init(alt_up_ps2_dev *ps2)`
Include: `<altera_up_avalon_ps2.h>`
Parameters: `ps2` – the PS/2 device structure.
Description: Initialize the PS/2 device and detect device type (mouse or keyboard).
Notes: The function will set the `device_type` field of *ps2* to `PS2_MOUSE` or `PS2_KEYBOARD` upon successful initialization, otherwise the intialization is unsuccessful.

4.3.3 alt_up_ps2_enable_read_interrupt

Prototype: `void alt_up_ps2_enable_read_interrupt(alt_up_ps2_dev *ps2)`
Include: `<altera_up_avalon_ps2.h>`
Parameters: `ps2` – the PS/2 device structure.
Returns: nothing
Description: Enable read interrupts for the PS/2 port.

4.3.4 alt_up_ps2_disable_read_interrupt

Prototype: `void alt_up_ps2_disable_read_interrupt(alt_up_ps2_dev *ps2)`
Include: `<altera_up_avalon_ps2.h>`
Parameters: `ps2` – the PS/2 device structure.
Returns: nothing
Description: Diaable read interrupts for the PS/2 port.

4.3.5 alt_up_ps2_write_data_byte

Prototype: `int alt_up_ps2_write_data_byte(alt_up_ps2_dev *ps2, unsigned char byte)`
Include: `<altera_up_avalon_ps2.h>`
Parameters: `ps2` – the PS/2 device structure.
`byte` – the byte to be written to the PS/2 port.
Returns: 0 on success, or `-EIO` on failure.
Description: Write a byte to the PS/2 port.

4.3.6 alt_up_ps2_write_data_byte_with_ack

Prototype: `int alt_up_ps2_write_data_byte_with_ack(alt_up_ps2_dev *ps2, unsigned char byte)`
Include: `<altera_up_avalon_ps2.h>`
Parameters: `ps2` – the PS/2 device structure.
`byte` – the byte to be written to the PS/2 port.
Returns: 0 on success, `-EIO` on write failure, or `-ETIMEDOUT` on timeout when waiting for the acknowledgment.
Description: Write a byte to the PS/2 port and wait for the acknowledgment.
Notes: The timeout value is defined in the PS/2 device structure .

4.3.7 alt_up_ps2_read_data_byte

Prototype: `int alt_up_ps2_read_data_byte(alt_up_ps2_dev *ps2, unsigned char *byte)`
Include: `<altera_up_avalon_ps2.h>`
Parameters: `ps2` – the PS/2 device structure.
`byte` – pointer to the memory location to store the byte.
Returns: 0 on success, or `-ETIMEDOUT` when timeout.
Description: Read a byte from the PS/2 port.
Notes: User can set disable the timeout by setting the `timeout` in to 0.

4.3.8 alt_up_ps2_clear_fifo

Prototype: `void alt_up_ps2_clear_fifo(alt_up_ps2_dev *ps2)`
Include: `<altera_up_avalon_ps2.h>`
Parameters: `ps2` – the PS/2 device structure.
Description: Clear the FIFO for the PS/2 port.

4.3.9 alt_up_ps2_read_fd

Prototype: `int alt_up_ps2_read_fd(alt_fd *fd, char *ptr, int len)`
Include: `<altera_up_avalon_ps2.h>`
Parameters: `fd` – the file descriptor for the PS/2 device.
`ptr` – memory location to store the bytes read.
`len` – number of bytes to be read.
Returns: the number of bytes actually read.
Description: Read *len* bytes from the PS/2 device.

4.3.10 alt_up_ps2_write_fd

Prototype: `int alt_up_ps2_write_fd(alt_fd *fd, const char *ptr, int len)`
Include: `<altera_up_avalon_ps2.h>`
Parameters: `fd` – the file descriptor for the PS/2 device.
`ptr` – memory location storing the bytes to write.
`len` – number of bytes to write.
Returns: the number of bytes actually written.
Description: Write *len* bytes to the PS/2 device from memory location pointed by *ptr* .

4.3.11 alt_up_ps2_open_dev

Prototype: `alt_up_ps2_dev* alt_up_ps2_open_dev(const char *name)`
Include: `<altera_up_avalon_ps2.h>`
Parameters: `name` the specified name of the device in Qsys
Returns: the PS/2 device structure
Description: Open a PS/2 device structure with *name* in Qsys.

4.4 PS/2 Keyboard Documentation

4.4.1 KB_CODE_TYPE

Prototype:

```
typedef enum {
    KB_ASCII_MAKE_CODE = 1;
    KB_BINARY_MAKE_CODE = 2;
    KB_LONG_BINARY_MAKE_CODE = 3;
    KB_BREAK_CODE = 4;
    KB_LONG_BREAK_CODE = 5;
    KB_INVALID_CODE = 6;
} KB_CODE_TYPE;
```

Include: `<altera_up_ps2_keyboard.h>`

Fields: `KB_ASCII_MAKE_CODE` — Make code that corresponds to an ASCII character. For example, the ASCII make code for key `[ A ]` is 1C.
`KB_BINARY_MAKE_CODE` — Make code that corresponds to a non-ASCII character. For example, the binary (non-ASCII) make code for key `[Left Alt]` is 11.
`KB_LONG_BINARY_MAKE_CODE` — Make code that has two bytes (the first byte is E0). For example, the long binary make code for key `[Right Alt]` is "E0 11".
`KB_BREAK_CODE` — Break code that has two bytes (the first byte is F0). For example, the break code for key `[ A ]` is "F0 1C".
`KB_LONG_BREAK_CODE` — Long break code that has three bytes (with the first two bytes "E0 F0"). For example, the long break code for key `[Right Alt]` is "E0 F0 11".
`KB_INVALID_CODE` — Scan codes that the decoding FSM is unable to decode.

Description: The enum type for the type of keyboard code received.

4.4.2 decode_scancode

Prototype: `int decode_scancode(alt_up_ps2_dev *ps2, KB_CODE_TYPE *decode_mode, alt_u8 *buf, char *ascii)`
Include: `<altera_up_ps2_keyboard.h>`
Parameters: `ps2` – the PS/2 device structure. The actually connected PS/2 device has to be a keyboard otherwise the function's behavior is undefined.
`decode_mode` – indicates which type of code (Make Code, Break Code, etc.) is received from the keyboard when the key is pressed.
`buf` – points to the location that stores the make/break code of the key pressed.
`ascii` – pointer to the memory location to store the pressed ASCII character. If a non-ASCII key is pressed, *ascii* will be set to 0
Returns: 0 for success, or negative errno for corresponding errors.
Description: Communicate with the PS/2 keyboard and get the make code of the key when a key is pressed.
Notes: For `KB_LONG_BINARY_MAKE_CODE` and `KB_BREAK_CODE` , only the second byte is returned. For `KB_LONG_BREAK_CODE` , only the third byte is returned.

4.4.3 set_keyboard_rate

Prototype: `alt_u32 set_keyboard_rate(alt_up_ps2_dev *ps2, alt_u8 rate)`
Include: `<altera_up_ps2_keyboard.h>`
Parameters: `rate` – an 8-bit number that represents the repeat/delay rate of the keyboard.
Returns: 0 on success, negative value on error.
Description: Set the repeat/delay rate of the keyboard.

4.4.4 translate_make_code

Prototype: `void translate_make_code(KB_CODE_TYPE decode_mode, alt_u8 makecode, char *str)`
Include: `<altera_up_ps2_keyboard.h>`
Parameters: `decode_mode` – the type of the make code (ASCII, binary, or long binary)
`makecode` – the last byte of the make code (if the make code has multiple bytes)
`str` – the pointer to the memory location to store the description string
Description: Translate the make code into string description.

4.4.5 reset_keyboard

Prototype: `alt_u32 reset_keyboard()`
Include: `<altera_up_ps2_keyboard.h>`
Parameters: –
Returns: 0 on passing the BAT (Basic Assurance Test), negative value on error.
Description: Send the reset command to the keyboard.

4.5 PS/2 Mouse Documentation

4.5.1 alt_up_ps2_mouse_reset

Prototype: `int alt_up_ps2_mouse_reset(alt_up_ps2_dev *ps2)`
Include: `<altera_up_ps2_mouse.h>`
Parameters: `ps2` – the PS/2 mouse device strcture
Returns: 0 on BAT is passed, `-EINVAL` when the PS/2 device is not mouse, or `-EIO` if error occurs.
Description: Reset the mouse.

4.5.2 alt_up_ps2_mouse_set_mode

Prototype: `int alt_up_ps2_mouse_set_mode(alt_up_ps2_dev *ps2, alt_u8 byte)`
Include: `<altera_up_ps2_mouse.h>`
Parameters: `ps2` – the PS/2 mouse device strcture
`byte` – the byte representing the mode (see macro definitions for details).
Returns: 0 on receiving acknowledgment, or negative number for errors.
Description: Set the operation mode of the mouse.
See also: PS/2 Mouse document