PS/2 Core for Altera DE2/DE1 Boards

Preliminary

1 Core Overview

The PS/2 Serial Port on Altera DE2/DE1 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. A device driver can communicate with it by reading/writing from/to its data and control registers.

3 Instantiating the Core in SOPC Builder

Designers can implement a PS/2 Core by using the SOPC Builder. There is no need to configure the core. The core comes with a 256-word FIFO for storing data received from a PS/2 device.

4 Software Programming Model

4.1 Register Map

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. 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…11**	**10**	**9**	**8**	**7…1**	**0**
0	`data`	R/W	RAVAIL	*(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.
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 Software Functions

The PS/2 Core is packaged with C-language functions accessible through the SOPC Builder-generated software development kit (SDK) libraries, as listed below. These functions implement common operations that users need for the PS/2 Core. When using the Altera Debug Client, these functions are automatically provided for use in a C-language application program. They are presented in Secition 4.3. To use the functions, the C code must include the statement:

```
#include "alt_up_ps2_port.h"
```

In addition, some sample functions for specific communication with the keyboard or mouse are also provided. They may 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, `ps2_keyboard.h` and `ps2_mouse.h`, have to be included. These functions are described in Sections 4.4 and 4.5.

4.3 PS/2 Port Functions

4.3.1 enum PS2_DEVICE

The Enum type for PS/2 device type.

Enumerator:

PS2_MOUSE

PS2_KEYBOARD

PS2_UNKNOWN

4.3.2 alt_u32 read_ctrl_reg ()

Read the contents of the Control register for the PS/2 port.

Returns:

Register contents (32 bits, bits 10, 8 and 0 are used for CE, RI and RE respectively. Other bits are reserved)

4.3.3 void write_ctrl_reg (alt_u32 *ctrl_data*)

Set the contents of the Control register.

Parameters:

ctrl_data – contents to be written into the Control register

4.3.4 alt_u8 read_RI_bit (alt_u32 *ctrl_reg*)

Extract the RI (Read Interrupt) bit from the Control register.

Parameters:

ctrl_reg – the Control register

Returns:

8-bit number, where bit 0 is the value of the RI bit

4.3.5 alt_u8 read_RE_bit (alt_u32 *ctrl_reg*)

Extract the RE (Read Interrupt Enable) bit from the Control register.

Parameters:

ctrl_reg – the Control register

Returns:

8-bit number, where bit 0 is the value of the RE bit

4.3.6 alt_u8 read_CE_bit (alt_u32 *ctrl_reg*)

Extract the CE (Command Error) bit from the Control register.

Parameters:

ctrl_reg – the Control register

Returns:

8-bit number, where bit 0 is the value of the CE bit

4.3.7 alt_u32 read_data_reg ()

Read the contents of the Data register.

Returns:

32 bits of the Data register. Bits 31-16 indicate the number of available bytes in the FIFO (RAVAIL), bits 7-0 are the data received from the PS/2 device

4.3.8 alt_u8 read_data_byte (alt_u32 *data_reg*)

Read the DATA byte from the Data register.

Parameters:

data_reg – Data register

Returns:

Bits 7-0 of the Data register

4.3.9 alt_u16 read_num_bytes_available (alt_u32 *data_reg*)

Find the number of bytes available to read in the FIFO buffer of the PS/2 port.

Parameters:

data_reg – the Data register

Returns:

The number represented by bits 31-16 of the Data register

4.3.10 PS2_DEVICE get_mode ()

Check the PS/2 peripheral's mode (whether it is a keyboard or a mouse).

Returns:

PS2_MOUSE for mouse, or PS2_KEYBOARD for keyboard

Note:

This operation will **reset** the PS/2 peripheral. Usually, it should be used only at the beginning of a program.

4.3.11 void clear_FIFO ()

Clear the FIFO's contents.

4.3.12 int wait_for_ack (unsigned *timeout*)

Wait for the acknowledge byte (0xFA) from the PS/2 peripheral.

Parameters:

timeout – the number of cycles of timeout

Returns:

`PS2_SUCCESS` on receving ACK signal, or `PS2_TIMEOUT` on timeout.

4.3.13 int write_data_byte (alt_u8 *byte*)

Send a one-byte command to the PS/2 peripheral.

Parameters:

byte – the one-byte command to be sent

Returns:

`PS2_ERROR` if the CE bit of the Control register is set to 1, otherwise `PS2_SUCCESS`

4.3.14 int write_data_byte_with_ack (alt_u8 *byte*, unsigned *timeout*)

Send a one-byte command to the PS/2 peripheral and wait for the ACK signal.

Parameters:

byte – the one-byte command to be sent. See `alt_up_ps2_port_regs.h` in the sdk directory or any reference for the PS/2 protocol for details.

Returns:

`PS2_ERROR` if the CE bit of the Control register is set to 1, or `PS2_TIMEOUT` on timeout, or `PS2_SUCCESS` if the ACK signal is received before timeout

4.3.15 int read_data_byte_with_timeout (alt_u8 * *byte*, alt_u32 *time_out*)

Read the DATA byte from the PS/2 FIFO, using a user-defined timeout value.

Parameters:

byte – the byte read from the FIFO for the PS/2 Core

time_out – the user-defined timeout value. Setting *time_out* to 0 will disable the time-out mechanism

Returns:

`PS2_SUCCESS` on reading data, or `PS2_TIMEOUT` on timeout

4.4 PS/2 Keyboard Functions

4.4.1 enum KB_CODE_TYPE

The Enum type for the type of keyboard code received.

Enumerator:

KB_ASCII_MAKE_CODE — Make Code that corresponds to an ASCII character. For example, the ASCII Make Code for letter `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 `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 `Right Alt` is "E0 11"

KB_BREAK_CODE — Normal Break Code that has two bytes (the first byte is F0). For example, the Break Code for letter `A` is "F0 1C"

KB_LONG_BREAK_CODE — Long Break Code that has three bytes (the first two bytes are E0, F0). For example, the Long Break Code for `Right Alt` is "E0 F0 11"

KB_INVALID_CODE — Codes that the decode FSM cannot decode

4.4.2 int read_make_code (KB_CODE_TYPE * *decode_mode*, alt_u8 * *buf*)

Get the make code of the key when a key is pressed.

Parameters:

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 code of the key pressed

Note:

For `KB_LONG_BINARY_MAKE_CODE` and `KB_BREAK_CODE`, only the second byte is retured. For `KB_LONG_BREAK_CODE`, only the third byte is returned

Returns:

`PS2_TIMEOUT` on timeout, or `PS2_ERROR` on error, otherwise `PS2_SUCCESS`

4.4.3 alt_u32 set_keyboard_rate (alt_u8 *rate*)

Set the repeat/delay rate of the keyboard.

Parameters:

rate – an 8-bit number that represents the repeat/delay rate of the keyboard

Returns:

PS2_SUCCESS on success, otherwise PS2_ERROR

4.4.4 alt_u32 reset_keyboard ()

Send the reset command to the keyboard.

Returns:

`PS2_SUCCESS` on passing the BAT (Basic Assurance Test), otherwise `PS2_ERROR`

4.5 PS/2 Mouse Functions

4.5.1 alt_u8 reset_mouse ()

Reset the mouse.

Returns:

`PS2_SUCCESS` on BAT is passed, otherwise `PS2_ERROR`

4.5.2 int set_mouse_mode (alt_u8 *byte*)

Set the operation mode of the mouse.

Parameters:

byte – the byte representing the mode (see macro definitions for details)

See also:

PS/2 Mouse document

Returns:

`PS2_SUCCESS` on receiving acknowledgment

4.6 Sample Program

Below is a sample program that shows some usage of the provided functions.

```
/**
 *
 * A simple program that illustrates the usage of some sdk functions
    of the
 * PS/2 Port SDK
 *
 **/
#include <alt_types.h>
#include <stdio.h>
#include "alt_up_ps2_port.h"
#include "ps2_keyboard.h"
#include "ps2_mouse.h"

int main()
{
    // clear the FIFO for the PS/2 port
    clear_FIFO();

    DECODE_MODE decode_mode;

    alt_u8 byte;

    // get whether the PS/2 device is a keyboard or a mouse
    PS2_DEVICE mode = get_mode();

    if (mode == PS2_KEYBOARD)
        printf("%s", "KEYBOARD...\n");
    else if (mode == PS2_MOUSE)
        printf("%s", "MOUSE...\n");

    if ( mode == PS2_KEYBOARD)
    {
        alt_u8 key = 0;
        int status = 0;
        do{
            // wait for the user's input and get the make code
            status = get_make_code(&decode_mode, &key);
            if (status == PS2_SUCCESS)
            {
                // print out the result
                switch (decode_mode)
                {
                    case KB_ASCII_MAKE_CODE:
                        printf("ASCII:\t%c\n", key);
                        break;
                    case KB_LONG_BINARY_MAKE_CODE:
                        printf("%s", "LONG");
                        //fall through
                    case KB_BINARY_MAKE_CODE:
                        printf("MAKE CODE:\t%X\n", key);
                        break;
                    case KB_BREAK_CODE:
                        //do nothing
                    default:
                        break;
                }
            }
            else
            {
                printf("Keyboard error....\n");
            }
        } while (1);
    }
    else if ( mode == PS2_MOUSE )
    {
        if (reset_mouse() == PS2_SUCCESS)
        {
            printf("MOUSE RESETTED...\n");
        }
        if (set_mouse_mode(MOUSE_STREAM_MODE) == PS2_SUCCESS)
        {
            printf("Set Mouse to Stream mode...\n");
        }
    }
    return 0;
}
```

When compiling the C program in the Altera Debug Client, you may wish to use the `-msmallc` option so that the *Small newlib C Library* is used to reduce the program size (See The HAL System Library in the *Nios® II Software Developer's Handbook* for details).

■