

PS/2 KEYBOARD INTERFACE

Interfacing a PS/2 keyboard with an Altera DE2 board

Nelson Stoik

Table of Contents

Background	3
Caution Regarding Interrupts	3
Project Setup.....	4
Qsys.....	4
Quartus	5
Nios II SBT for Eclipse.....	5
Software API.....	6
Void ps2_initialisation.....	6
Void clear_input()	6
Void enable_scanning().....	6
Void disable_scanning()	6
Void input_monitoring_task(void * pdata).....	6
Int read_and_decode(char *decoded_string)	6
Sample API Code	7
Test Project – hello_ucosii.c	7
Input Monitoring Task.....	8
References	9

Background

The Altera DE2 board has a 6-pin mini-DIN connector PS2 serial port on it. The PS2 port is most commonly used for interfacing with a PS2 mouse or keyboard. Altera provides a PS2 controller component in the University Program section that is able to interface with the PS2 serial port and provide an easy-to-use communication [1]. There are also driver files created by Altera that are automatically added to the board support package once the PS2 component has been added to the hardware. PS2 keyboards work by outputting a unique make code for each key when pressed and then outputting another unique break code for each key when released [2]. The PS2 Controller reads these make and break codes and stores them in its buffer. These make and break codes can then be read and decoded into their ASCII equivalent.

To fully use this Appnote, a working PS2 keyboard is required. It is assumed that you already have a working Nios II system created to which this tutorial builds upon. Once this tutorial is complete, keys pressed on an attached PS2 keyboard will be displayed on the Eclipse console that is connected to the Altera DE2 board.

Caution Regarding Interrupts

Although the Altera PS2 Controller does have an IRQ signal to enable interrupts, they are not used in this particular setup. The PS2 Controller requires the use of legacy interrupts and there was some trouble getting the legacy interrupts to co-exist with the non-legacy interrupts used by other components. It is possible to use the PS2 Controller interrupts but that is not covered in this application note.

Project Setup

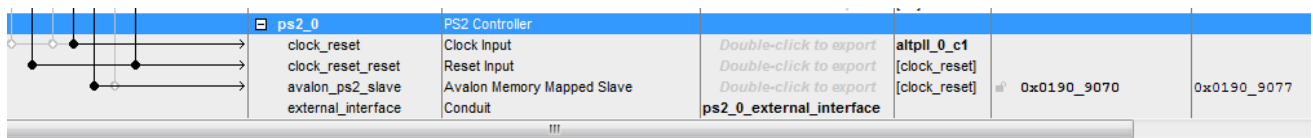
The following describes how to add the Altera provided PS2 Controller to an already existing Nios II system. First the core is added to the Nios II system using Qsys. Next the signals are connected in the top level file of the system in Quartus and the DE2 board is programmed. Finally the demo files are loaded into an Eclipse project. This tutorial assumes that you already have a basic Nios II system configured.

To fully use this Appnote, a working PS2 keyboard is required. Once this tutorial is complete, keys pressed on an attached PS2 keyboard will be displayed on the Eclipse console that is connected to the Altera DE2 board.

Qsys

First you need to add a PS2 controller to your NIOS II system.

1. Add a PS2 instance from the component library found under: University Program > Generic IO > PS2 controller
 - a. Set the Avalon Type as: Memory Mapped
 - b. Set the Incoming clock rate as: 50000000
 - c. Click finish
 - d. You can rename the controller to something more descriptive if desired (for this tutorial the default ps2_0 name was used)
 - e. Connect the PS2 controller to the rest of the NIOS II system, remembering to export the conduit.



The screenshot shows the Qsys component library with the PS2 Controller component selected. The component is named 'ps2_0' and is located under 'University Program > Generic IO > PS2 controller'. The component's properties are listed in a table below.

Component Name	Component Type	Export Name	Export Name	Export Name	Export Name
ps2_0	PS2 Controller	clock_reset	Double-click to export	altpll_0_c1	
		clock_reset_reset	Double-click to export	[clock_reset]	
		avalon_ps2_slave	Double-click to export	[clock_reset]	
		external_interface	ps2_0_external_interface		0x0190_9070

2. If you have conflicting addresses, generate new base addresses by clicking System > Assign Base Addresses.
3. Save your changes and generate the SOPC.

Quartus

Next you need to connect the new signals from the NIOS II system to the pins of the FPGA board. In your top level .vhd files:

1. Add the following to the top level entity:
 - a. PS2_CLK : inout std_logic
 - b. PS2_DAT : inout std_logic
2. Add the following to the niosII_system component declaration (this information is from the HDL Example tab auto generated in Qsys):
 - a. ps2_0_external_interface_CLK : inout std_logic
 - b. ps2_0_external_interace_DAT : inout std_logic
3. Finally connect the newly added signals in the niosII_system component instantiation:
 - a. ps2_0_external_interface_CLK => PS2_CLK
 - b. ps2_0_external_interfcae_DAT => PS2_DAT
4. Compile the design
5. Program the DE2 board with the newly generated .sof file.

Nios II SBT for Eclipse

1. Create a new Nios II Application and BSP from Template using the newly created .sopcinfo file from Quartus.
2. Download and add the PS2_Controller.c and PS2_Controller.h files and add them to your project.
3. Replace the contents of the auto generated hello_ucosii.c with the downloaded hello_ucosii.c file that contains the setup for the task and a small test program that enables and disables input

Software API

The following functions are implemented in the PS2_Controller.c and PS2_Controller.h files

Void ps2_initialisation

Inputs	None
Returns	None
Description	Set all variables to their starting values and starts the ps2 monitoring task.

This function can only be called once. In the current configuration, this function is called first thing by the input_monitoring_task upon being started.

Void clear_input()

Inputs	None
Returns	None
Description	Clears all input from the ps2 controller

Void enable_scanning()

Inputs	None
Returns	None
Description	Enable the task to scan for data from the ps2 device

Note: Scanning is enabled by default after ps2_initialisation is run

Void disable_scanning()

Inputs	None
Returns	None
Description	Set all variables to their starting values and starts the ps2 monitoring task. Call once before starting the main program

Void input_monitoring_task(void * pdata)

Inputs	Data that may be passed to any running task. Not used for this task
Returns	None
Description	A task that polls the ps2 controller for available input and prints it to the console. It is paused by calling disable_scanning() and resumed by calling enable_scanning()

Int read_and_decode(char *decoded_string)

Inputs	A string containing the data read from the ps2 controller. The parsed input is returned with this string as well
Returns	None
Description	Read the codes from the ps2 controller and convert them to ASCII characters

Sample API Code

Test Project – hello_ucosii.c

The hello_ucosii.c file contains some sample code to show how to enable and disable scanning from the PS2 Controller. It also sets up the input_monitoring_task to run in the main program.

```
#include <stdio.h>
#include "includes.h"
#include "PS2_Controller.h"

/* Definition of Task Stacks */
#define TASK_STACKSIZE 2048
OS_STK task1_stk[TASK_STACKSIZE];
OS_STK ps2_task_stk[TASK_STACKSIZE];

/* Definition of Task Priorities */
#define TASK1_PRIORITY 2
#define PS2_TASK_PRIORITY 1

void task1(void* pdata) {
    while (1) {
        OSTimeDlyHMSM(0, 0, 15, 0);
        printf("Turning off input\n");
        disable_scanning();
        OSTimeDlyHMSM(0, 0, 15, 0);

        printf("Turning on input\n");
        enable_scanning();
        OSTimeDlyHMSM(0, 0, 15, 0);

        printf("Turning off input again\n");
        disable_scanning();
        OSTimeDlyHMSM(0, 0, 15, 0);

        printf("Clearing and turning on input\n");
        clear_input();
        enable_scanning();
    }
}

/* The main function creates two task and starts multi-tasking */
int main(void) {
    INT8U err;

    //create the input_monitoring_task
    err = OSTaskCreateExt(input_monitoring_task, NULL,
        (void *) &ps2_task_stk[TASK_STACKSIZE - 1], PS2_TASK_PRIORITY,
        PS2_TASK_PRIORITY, ps2_task_stk, TASK_STACKSIZE, NULL, 0);

    err = OSTaskCreateExt(task1, NULL, (void *) &task1_stk[TASK_STACKSIZE - 1],
        TASK1_PRIORITY, TASK1_PRIORITY, task1_stk, TASK_STACKSIZE, NULL, 0);

    OSStart();
    return 0;
}
```

Input Monitoring Task

The following code shows how the `input_monitoring_task` works in the `PS2_Controller.c` file

```
void input_monitoring_task(void* pdata)
{
    //message read from the ps2 controller
    char message_received[DECODE_STRING_LENGTH];
    int read_return = READ_BAD;
    INT8U err;
    ps2_initialisation();
    memset(message_received, 0, DECODE_STRING_LENGTH);

    while (1){
        //reset the variable and attempt to read from PS2
        memset(message_received, 0, DECODE_STRING_LENGTH);
        read_return = read_and_decode(message_received);

        //good read from PS2, parse it
        if (read_return == READ_AND_DECODE_GOOD){
            //ignore blanks; these are the break codes from the keyboard
            if(strncmp(message_received, "", sizeof("")) != 0){
                printf("%s\n", message_received);
            }
        }
        //bad read, wait and try again
        else {
            //printf("bad read\n");
            OSTimeDlyHMSM(0, 0, 0, 500);
        }

        //check if the input has been disabled, wait until it has been enabled again
        OSMutexPend(input_mutex, 0, &err);
        if(input_enabled == PS2_INPUT_DISABLED){
            OSMutexPost(input_mutex);
            OSSemPend(wait_for_input_enabled, 0, &err);
        }
        else{
            OSMutexPost(input_mutex);
        }
    }
}
```


References

[1] PS2 Controller
Altera IP Core - PS2 Controller - 02/02/2015
http://www.altera.com/education/univ/materials/comp_org/ip-cores/unv-ip-cores.html

[2] Keyboard Make and Break Codes
Keyboard Scan Codes: Set 2 – 02/02/2015
<http://www.computer-engineering.org/ps2keyboard/scancodes2.html>