Tutorial of Interfacing with PIO interrupt

Kwan Yin Lau (group 3)
Jan 28, 2014

Introduction

On the DE2 board, we can access any switch, LEDs using the parallel input/output (PIO) core with Avalon® interface to read or write data using the data registers in PIO. But when we want to interface with a keypad or any button-typed external input, it is much suitable to use the interrupt feature provided by the PIO core. Such that once a keypad or a button is pressed, an interrupt can be generated, then it can be handled by an ISR and do whatever useful things. This tutorial will demonstrate how to interface a keypad such that when KEY3 on board is pressed, a message will be display on the LCD screen.

Procedure

1. In Qsys, add a new key interface that using PIO. Select Peripherals -> Microcontroller Peripherals -> PIO (Parallel I/O).
2. Under Basic settings, set width as 1, Direction as input. Under Output register, check Synchronously capture, choose RISING from Edge Type drop down list, check Enable bit-clearing for edge capture register. Under Interrupt, check Generate IRQ, choose EDGE for IRQ Type. Click Finish. Rename the PIO as key_3.
3. Connect the PIO clk signal to your clock source clk, connect clock source clk_reset and processor reset_n to PIO reset, connect processor data master to s1. Double click under export column on the external connection Conduit endpoint. Click on the IRQ column to add the IRQ. Re do Assign base address if necessary.
4. Generate the new design.
5. Modify your top-level *.vhd as follows:
    a. under port of entity, declare the input port
    ```
    KEY   : in  std_logic_vector (3 downto 0);
    ```
    b. under the component of your niosII_system port, declare
    ```
    key_3_external_connection_export      : in std_logic:= 'X';
    ```
    c. under u0 port map, add
    ```
    key_3_external_connection_export      => KEY(3),
    ```
    d. make sure KEY(0) is port map to reset_reset_n for system reset, don't change that.
6. Compile the new design.
7. Modify your software with reference to the following code.

```
#include <stdio.h>
#include "includes.h"
#include "altera_up_avalon_character_lcd.h"
#include "altera_up_avalon_character_lcd_regs.h"
#include "altera_avalon_pio_regs.h"
#include "sys/alt_irq.h"
#include "alt_types.h"

/* Definition of Task Stacks */
```

```
#define   TASK_STACKSIZE             2048
#define   Q_STACKSIZE          64
#define   KEY_PRESSED          1
#define   WAIT_FOREVER         0
OS_STK    task_LCD_stk[TASK_STACKSIZE];
OS_STK    Q_stk[Q_STACKSIZE];
OS_EVENT *KEY_Q;

/* Definition of Task Priorities */
#define TASK_LCD_PRIORITY      1

/* Prints message from queue to LCD */
void task_LCD(void* pdata)
{
      INT8U err;
      alt_up_character_lcd_dev * char_lcd_dev;
      // open the Character LCD port
      char_lcd_dev = alt_up_character_lcd_open_dev ("/dev/character_lcd_0");
      if ( char_lcd_dev == NULL)
            alt_printf ("Error: could not open character LCD device\n");
      else
            alt_printf ("Opened character LCD device\n");
      alt_up_character_lcd_init (char_lcd_dev);

  while (1)
  {
        IOWR_ALT_UP_CHARACTER_LCD_COMMAND(CHARACTER_LCD_0_BASE,
ALT_UP_CHARACTER_LCD_COMM_CLEAR_DISPLAY);
        alt_printf ("pend...\n");
        if (OSQPend (KEY_Q,WAIT_FOREVER,&err) == KEY_PRESSED) {
              //Do whatever job here
              alt_up_character_lcd_string(char_lcd_dev, "KEY PRESSED");
              OSTimeDlyHMSM(0, 0, 1, 0);
        }
        if (err != OS_NO_ERR) alt_printf ("Queue error\n");
  }
}

/*
 * ISR of key which fires when key 3 is pressed,
 * sends a message to LCD task
 */
static void key_ISR( void * context) {
//interrupt safe instructions...
      OSQPost(KEY_Q, KEY_PRESSED);

//signal to the IIC that interrupt processing form key is complete.
//Acknowledge the IRQ : Clear the edgecapture register by writing a 1 to it
      IOWR_ALTERA_AVALON_PIO_EDGE_CAP(KEY_3_BASE, KEY_3_BIT_CLEARING_EDGE_REGISTER);
}
```

```
/* The main function creates LCD task and set up the ISR */
int main(void)
{
      OSInit();
      KEY_Q = OSQCreate (Q_stk, TASK_STACKSIZE);
  OSTaskCreateExt(task_LCD,
                  NULL,
                  (void *)&task_LCD_stk[TASK_STACKSIZE-1],
                  TASK_LCD_PRIORITY,
                  TASK_LCD_PRIORITY,
                  task_LCD_stk,
                  TASK_STACKSIZE,
                  NULL,
                  0);
  alt_ic_isr_register(KEY_3_IRQ_INTERRUPT_CONTROLLER_ID,      //alt_u32 ic_id
                                KEY_3_IRQ,       //alt_u32 irq
                                key_ISR,  //alt_isr_func isr
                                NULL,
                                NULL);
  //Setting interruptmask register to 1 enables key interrupts
  IOWR_ALTERA_AVALON_PIO_IRQ_MASK(KEY_3_BASE, KEY_3_CAPTURE);
  OSStart();
  return 0;
}
```

Description

- The PIO hardware configurations sets up the functionality of: when a key is press, the external input signal to PIO goes high, the rising edge will be capture by the PIO core, the edge capture register will be automatically set to 1. An IRQ will be generated since it is rising edge sensitive.
- When the software start up, it first create the queue and the LCD task, then set up the ISR with parameters of our PIO KEY_3, `KEY_3_IRQ_INTERRUPT_CONTROLLER_ID` and `KEY_3_IRQ` from system.h and key_ISR which is the name of ISR function.
- next we need to enable interrupt form the PIO by setting interrupt mask register to 1, since all PIO interrupts are disabled in start up.
- In the ISR function key_ISR, it post a `KEY_PRESSED` message to the queue. Then it needs to signal to the IIC that interrupt processing form PIO is complete. To deassert the IRQ, we need to write to `edgecapture` register. Since the option Enable bit-clearing for edge capture register is turned on, writing a 1 to a particular bit in the `edgecapture` register clears only that bit. If Enable bit-clearing for edge capture is turned off, writing any value to the `edgecapture` register clears all bits in the register.
- In the LCD task, it just pend on the queue and display a message on LCD, how to do that is not the focus of this tutorial, details will be omitted here.
- In this simple demonstration, only one bit input is used for PIO, as we are interfacing with only one key button. But it is possible to interface with multiple inputs with one PIO core, just increase the width of input bits when adding a new PIO. Then each `edgecapture` register will be monitoring its corresponding input bit. In case of writing to the PIO registers, just be careful which bit you are changing.

Source Documentation and Reference

PIO: Chapter 10 of the Embedded Peripherals IP User Guide – Altera Quartus® II design software
- ug_embedded_ip.pdf
- altera_avalon_pio_regs.h

LCD: 16x2 Character Display for Altera DE2-Series Boards – Altera University Program
- ug_embedded_ip.pdf
- altera_up_avalon_character_lcd_regs.h
- altera_up_avalon_character_lcd.h

Code modified from Hello MicroC/OS-II template from Nios II Software Build Tools for Eclipse.