Interrupt setup tutorial

This instruction presents the tutorial of setup the interrupt on DE2i-150 board.

Step 1. Look for interrupt source:

Decide which signal will be used to generate an interrupt that stop the current CPU work. The interrupt signals usually come from peripheral device, it can also be a hardware change, like press a key button.

Step 2. Look for the interrupt IRQ:

Interrupt IRQ is the interrupt ID (an integer value) of each hardware. They are always defined in “system.h” file located in your BSP project directory.

Available IRQ in this technical package are:

-physical part

4 keys (buttons).

8 switches.

-hardware part

Leftready

Rightready

These two pulses mean one ADC value has been ready in the 16 bits data buffer.

Step 3. Declare global ID in main.c file. (Optional)

Example, if we want to use switch1 as an interrupt source. We can declare a global ID like:

alt_u32 switch1_id = SWITCH1_IRQ;

alt_u32 means the type of the variable is 32 bits unsigned integer.

SWITCH1_IRQ is switch1's interrupt ID, which was defined in "system.h" in BSP project directory.

Since declare a global variable will increase the total memory usage of your CPU, you can decide not to declare the in global. Please go to next step for further instruction:

Step 4. Declare a global integer value for ISR to store the content.

Declare a global variable in your main.c:

volatile int switch1 = 0;

Volatile type integer means the value of the variable will be changed unexpectedly by other peripheral devices. (You can just use it without fully understand it)

Step 5. Write your interrupt service routine (ISR).

ISR is a function which will be accessed when a specific interrupt happen. Generally speaking, when this switch1 interrupt happen, the CPU will stop, save its current job and directly go to this function. When the function is finished, CPU will go back to their previous job.

Please follow the ISR code structure below:

```
static void handle_switch1_interrupt(void* context, alt_u32 id) {
        volatile int* switch1ptr = (volatile int *)context;
        *switch1ptr = IORD_ALTERA_AVALON_PIO_EDGE_CAP(SWITCH1_BASE);

        /* Write to the edge capture register to reset it. */
        IOWR_ALTERA_AVALON_PIO_EDGE_CAP(SWITCH1_BASE, 0);

        /*Perform Jobs*/
        !!!! Write your own code here!!!!
}
```

“handle_switch1_interrupt” is the name of this ISR, this name format tells you that this ISR will be served for switch1’s interrupt.

What you need to modify for your other ISR:

1. Change switch1ptr to other name, like switch2ptr, key0ptr. Name does not matter.

2. Change SWITCH1_BASE to other base, for example, SWITCH2_BASE, KEY0_BASE (please look for the right base name in “system.h” file.

3. Write your own code after /*Perform Jobs*/.

For example, if you want to turn all LED on you can write:

*/*Perform Jobs*/*

IOWR_ALTERA_AVALON_PIO_DATA(LED_BASE, 0xFF);

Some tips for successful ISR programming (from Professor Peckol’s 472 note package):

Problems with ISRs
• ISRs are virtually impossible to debug!
• The debugger uses “software interrupts” to handle tricks like breakpoints and single stepping.
• thus even correct ISRs will break the debugger OR
• the debugger will break your ISRs
• ISRs take over the processor – even pre-emptive schedulers.
• ISRs can easily mess up other processes if they
– write in wrong global variables
– take up a lot of CPU time
– play around with the stack

ISR Solutions
• Keep your ISRs *small, simple, & quick*
• No loops inside ISR’s
• No complex logic (no nested if’s)
• Don’t do unrelated I/O in the ISR (for example, no printf("ISR message");).
• Guidelines: Max: 1/2 page of C, 1 page of ASM
• Do not trust / use debugger.
• Do absolute minimum of stuff in ISR, do the rest in a regular task.
• Check and recheck your use of interrupt masking and how you save and restore the context.

Step 6. Register the interrupt into the system:

Assume you just purchase a car, you need to register the car in government office and tell them what car you just bough (year, type, etc.) and you are the owner of the car now. This step is doing the very similar thing, you register the interrupt to the system so the system will know it needs to do something to serve this interrupt.

Please register the interrupt in your system initialization function by writing:

*alt_irq_register(switch1_id, (void *)&switch1, handle_switch1_interrupt);*

There are three parameters to be passed in:

1. Switch1_id: store the interrupt ID (switch1 IRQ)

2. (void *)&switch1: a void type pointer that points to the memory address of switch1 variable you declare in the previous step.

3. handle_switch1_interrupt: this is the function that serves the interrupt.

This step is telling the system which interrupt will be registered and which function will be served to this interrupt.

Step 7. Interrupt enable:

Once you register the interrupt, you need to enable the interrupt by calling:

IOWR_ALTERA_AVALON_PIO_IRQ_MASK(SWITCH1_BASE, 1);

Step 8. Initialize the interrupt process:

Put this line of code after step 6:
IOWR_ALTERA_AVALON_PIO_EDGE_CAP(SWITCH1_BASE, 0);

Now your interrupt will be ready to serve.