

Using C with Altera DE2 Board

This tutorial explains how to communicate with IO devices on the DE2 Board and how to deal with interrupts using C and the Altera Monitor Program. Two example programs are given that display the state of the toggle switches on the red LEDs. The first program uses the programmed I/O approach and the second program uses interrupts.

Contents:

Setting up the DE2 Basic Computer
Input and Output Using C
Creating Interrupts Using C

Doing this tutorial the reader will learn about:

- Communicating with simple input/output devices (specifically the switches and LEDs) using the C programming language.
- Dealing with interrupts within a C program for the Nios II processor.

All files needed for this tutorial are located in the Altera Monitor Program install directory. If the Nios II install directory is *C:/altera/80/nios2eds*, then the sample files used in Section 1 are located at:

C:/altera/80/nios2eds/bin/monitor/samples/Programs/switches_to_LEDs.

The sample files used in Section 3 are located at:

C:/altera/80/nios2eds/bin/monitor/samples/Programs/pushbutton_interrupts.

Also, the DE2 Basic Computer is located at:

C:/altera/80/nios2eds/bin/monitor/samples/Systems/DE2_Basic_Computer.qpf.

These directories must be adjusted if the Nios II install directory is different on your computer. For this tutorial Nios II was installed at *C:/altera/80/nios2eds*. You can find where you have installed nios2eds by looking at the SOPC_KIT_NIOS2 windows environment variable. To do this, open a command prompt and type `set`, and observe the text after SOPC_KIT_NIOS2 in the list of environment variables that comes up.

PREREQUISITES

The reader is expected to have access to a computer that has Quartus II software installed. The detailed examples in the tutorial were obtained using Quartus II version 8.0. The reader should be familiar with the basic operation of the Altera Monitor Program including how to compile and load a program onto the DE2 board. To learn about the Altera Monitor Program, consult the *Altera Monitor Program* tutorial available on the DE2 website and also the Altera Monitor Program Help menu. Furthermore, the reader should have a basic understanding of the C programming language.

1 Setting up the DE2 Basic Computer

This tutorial makes use of a circuit called the DE2 Basic Computer, which must be loaded onto the DE2 Board. This system was created by using Altera's SOPC Builder and is included with the Altera Monitor Program. It contains memory, simple IO interfaces, and a JTAG UART for communication with the host computer. To learn more about the DE2 Basic Computer, consult the document *Basic Computer System for Altera DE2 Board* available on the University Program website and also in the DE2 Basic Computer directory. In this tutorial we will use parallel (PIO) interfaces that are connected to the toggle switches, pushbuttons, and red LEDs. A diagram of the system that includes these elements is shown in Figure 1.

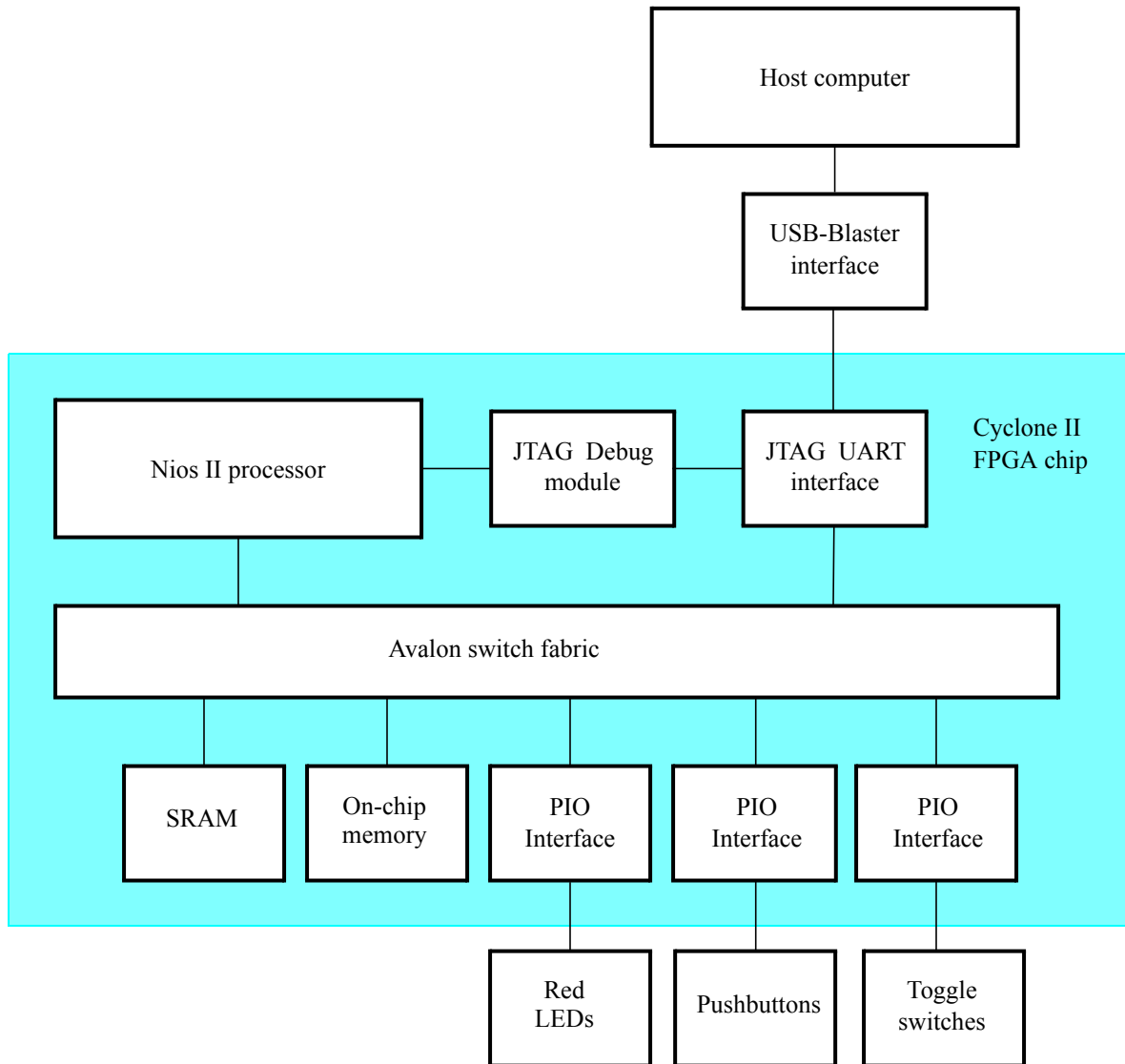


Figure 1. The DE2 Basic Computer with the components used in this tutorial.

To configure the FPGA on the DE2 board with this circuit follow these steps:

1. Open Altera's Quartus II software.
2. Click File > Open Project.
3. The location of the DE2 Basic Computer files will depend on where you have Nios II installed. For this example, Nios II is installed at *C:/altera/80/nios2eds*. Locate in the Open Project window the file *C:/altera/80/nios2eds/bin/monitor/samples/systems/DE2_Basic_Computer.qpf* and click OK.
4. Go to Tools > Programmer and load the circuit onto the DE2 board.

2 Input and Output Using C

We will now discuss a program that continuously examines the state of the switches and displays this state on the red LEDs. Figure 2 shows the code used to do this. This code is included in the file *switches_using_basic_system.c*.

```
#define SWITCHES_BASE_ADDRESS 0x10000010
#define LEDR_BASE_ADDRESS 0x10001000

int main(void)
{
    int * red_leds = (int *) LEDR_BASE_ADDRESS;          /* red_leds is a pointer to the LEDRs */
    volatile int * switches = (int *) SWITCHES_BASE_ADDRESS; /* switches point to toggle switches */
    while(1)
    {
        *(red_leds) = *(switches);                      /* Red LEDR[k] is set equal to SW[k] */
    }
    return 0;
}
```

Figure 2. The application program that displays the state of the switches.

In the DE2 Basic Computer, IO devices are memory mapped. The PIOs connected to the red LEDs, switches, and pushbuttons are located at addresses 0x10000000, 0x10000040, and 0x10000050, respectively. To access these memory locations in C, a pointer must be used. To access the values stored in the location pointed to by the pointer, the `*` character is used, as in the code:

```
*(red_leds) = *(switches);
```

This statement moves the value in memory location pointed to by *switches* to the memory location pointed to by *red_leds*.

When the variable *switches* is declared, the keyword `volatile` is used. A volatile variable is a variable that the compiler assumes can be changed by an action outside the program. This prevents certain compiler optimizations from occurring because the compiler must assume that the value pointed to by the variable could change at any time. The statement `volatile int* switches` indicates that *switches* is a pointer to a volatile integer value. This value is volatile because it can be changed by the user flipping the switches.

To run the program, perform the following:

1. Make sure the DE2 Basic Computer is loaded on the DE2 board as described in Section 1.
2. Open the Altera Monitor Program, select **Configuration > Configure System** and locate the *nios_system.ptf* file that is in the same directory as the *DE2_Basic_Computer.qpf* file loaded in Section 1. Make sure that the memory device selected for the `.data` and `.text` sections is the onchip memory. If it is not, in the memory box dropdown box select `onchip_memory` resulting in the System Configuration window shown in Figure 3. Click OK.

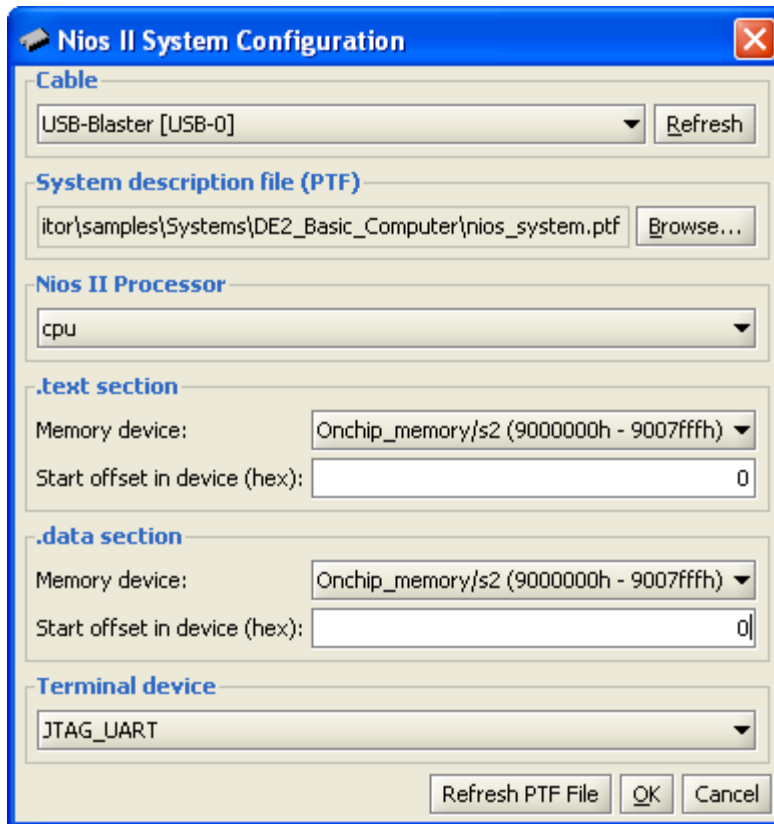


Figure 3. Select the on-chip memory for .data and .text sections.

3. Click Configuration > Configure Program and add the file *switches_using_basic_system.c* and click OK. Then click Actions > Compile and Load. When the program is downloaded, the window in Figure 4 will appear.
4. Run the program and then test the design by flipping some switches and observing the effect on the LEDs.

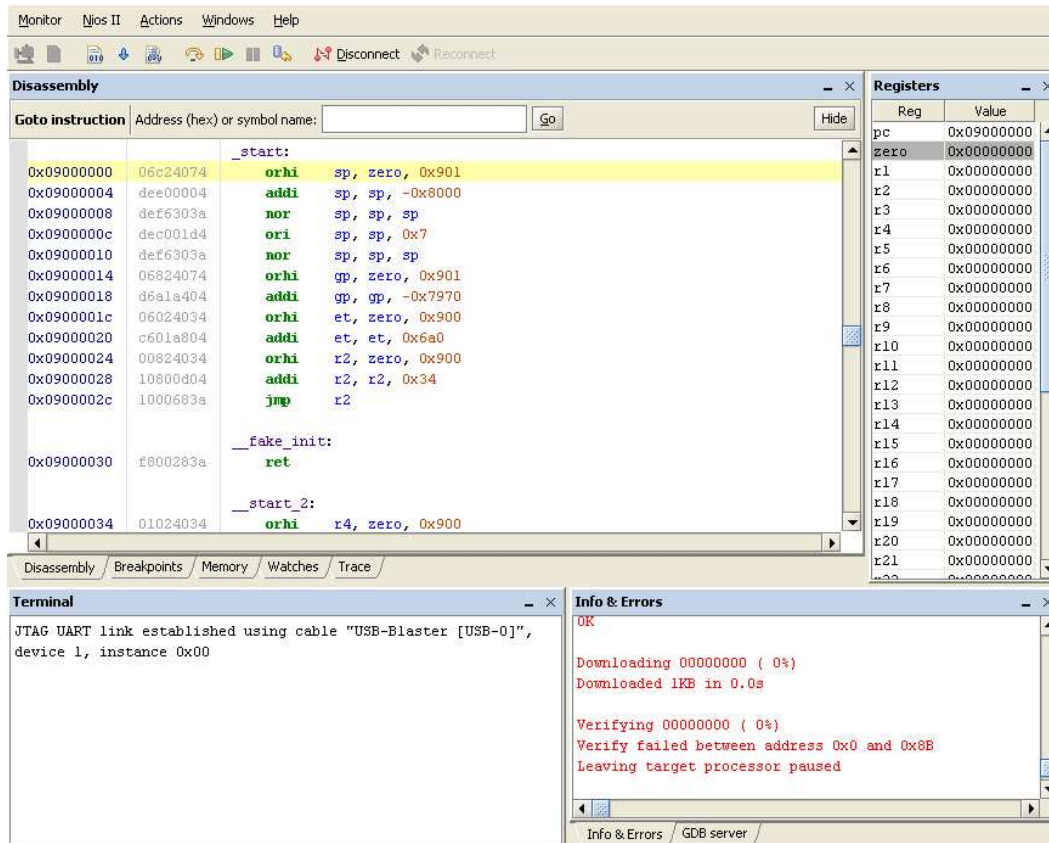


Figure 4. Sample program loaded.

3 Handling Interrupts Using C

This section explains how to create a program that enables interrupts and handles these interrupts in an interrupt service routine (ISR). The reader is expected to understand the Nios II interrupt mechanism as described in section 9 of the *Introduction to the Altera Nios II Soft Processor* tutorial.

3.1 The Sample Interrupt Program

Consider a program that displays the state of the switches on the red LEDs. The state of the LEDs can change only when the pushbutton *KEY₃* is pressed.

The sample program is implemented in two files: *pushbuttons_interrupt.c* and *isr_linkage.c*. The file *isr_linkage.c* provides the framework for dealing with interrupts. It can be used to deal with interrupts in general. It adjusts the program counter for the case of external interrupts, stores registers on the stack, and then calls the function *interrupt_handler*. Upon completion of *interrupt_handler()*, the registers are restored from the stack and control of the program is returned to the point where the interrupt was triggered. To use this function to create your own program that can handle interrupts, simply copy the file *isr_linkage.c* to your program directory and include it as a source file in the program configuration window of the Altera Monitor Program. The file *pushbuttons_interrupt.c* contains the function *interrupt_handler()*, which can be modified to create different interrupt handlers. The text of this file is given in Figure 5.

```

#define SWITCHES_BASE_ADDRESS 0x10000040
#define LEDR_BASE_ADDRESS 0x10000000
#define PUSHBUTTONS_BASE_ADDRESS 0x10000050

void switches_isr(void);
void interrupt_handler(void)
{
    int ipending;
    ipending = __builtin_rdctl(4); //Read the ipending register
    if ((ipending & 0x2) == 2) //If irq1 is high, run pushbutton_isr, otherwise return
    {
        pushbutton_isr();
    }
    return;
}

void pushbutton_isr(void)
{
    int * red_leds = (int *) LEDR_BASE_ADDRESS;
    volatile int * pushbuttons = (int *) PUSHBUTTONS_BASE_ADDRESS;
    volatile int * switches = (int *) SWITCHES_BASE_ADDRESS;

    *(red_leds) = *(switches); //Make LEDs light up to match switches

    *(pushbuttons+3) = 0; //Disable the interrupt by writing to edgecapture registers of pushbutton PIO
    return;
}

int main(void)
{
    volatile int * pushbuttons = (int *) PUSHBUTTONS_BASE_ADDRESS;

    *(pushbuttons + 2) = 0x8; //Enable KEY3 to enable interrupts

    __builtin_wrctl(3, 2); //Write 2 into ienable register
    __builtin_wrctl(0, 1); //Write 1 into status register
    while(1);
    return 0;
}

```

Figure 5. The *pushbuttons_interrupt.c* text.

3.2 Initializing Nios II Control

System registers must be initialized to enable interrupts. This is done in the main program. In the Nios II processor, there are 32 separate interrupt request bits, irq_{31-0} . To enable the k th interrupt irq_k , the following needs to be true:

- The PIE bit (b_0) in the status register ($ctl0$) is set to 1. The status register of the Nios II processor reflects the operating status of the processor. If $PIE=1$, the processor may accept external interrupts. When $PIE=0$, the processor ignores external interrupts.
- The corresponding interrupt-enable bit in the *ienable* register, $ctrl3_k$, is set to 1. When the bit $ctrl3_k=1$, the processor may accept external interrupts from devices connected to irq_k . If $ctrl3_k=0$, the processor will ignore interrupts from devices connected to irq_k .
- An interrupt-request input, irq_k , is asserted.

The first two of these conditions must be set when the software is initialized. The third condition is satisfied when KEY_3 is pressed. In the DE2 Basic Computer, the pushbutton PIO's interrupt request line is connected to irq_1 , as seen in Figure 6. This figure shows the base addresses assigned to various components in the DE2 Basic Computer, and also the irq_k bits that the components in the DE2 Basic Computer are connected to.

| Component | Address | irq bit |
|-----------------|------------|-----------|
| SRAM | 0x08000000 | |
| on-chip_memory | 0x09000000 | |
| red_leds | 0x10000000 | |
| green_leds | 0x10000010 | |
| HEX3-HEX0 | 0x10000020 | |
| HEX7-HEX4 | 0x10000030 | |
| toggle_switches | 0x10000040 | |
| Pushbuttons | 0x10000050 | 1 |
| Expansion_JP1 | 0x10000060 | 11 |
| Expansion_JP2 | 0x10000070 | 12 |
| JTAG_UART | 0x10001000 | 8 |
| Serial port | 0x10001000 | 10 |
| Interval timer | 0x10002000 | 0 |
| sysid | 0x10002020 | |

Figure 6. Base addresses and interrupt assignment of components in the example system.

The values in the control registers can be changed by using the built-in functions used by the NIOS II compiler. In the sample program, the line `__builtin_wrc1(3, 2)` writes the value 2 into $ctrl3$. This corresponds to writing 2 (binary 10) into the *ienable* register, causing irq_1 to enable interrupts. In the sample program, the line `__builtin_wrc1(0, 1)` writes the value 1 into $ctl1_0$. This corresponds to writing a 1 into the PIE bit of the status register.

3.3 Initializing PIO Registers to Enable Interrupts

The parallel port connected to the KEY_{3-0} pushbutton switches on the DE2 board comprises three 4-bit registers, as shown in Figure 7. These registers have addresses 0x10000050 to 0x1000005F and can be accessed using word operations. The read-only *Data* register provides the values of the switches KEY_3 , KEY_2 and KEY_1 . Bit 0 of the data register is not used because the corresponding switch KEY_0 is reserved for use as a reset mechanism in the DE2 Basic Computer. The *Interruptmask* register allows processor interrupts to be generated when a key is pressed. Each bit in the *Edgecapture* register is set to 1 by the parallel port when the corresponding key is pressed. The Nios II processor can read this register to determine which key has been pressed in response to receiving an

interrupt request, if the corresponding bit in the interrupt mask register is set to 1. Writing any value into the *Edgecapture* register deasserts the Nios II interrupt request and sets all bits of the *Edgecapture* register to zero.

| Address | 31 | 30 | ... | 4 | 3 | 2 | 1 | 0 | |
|------------|--------|----|-----|---|--------------------|---|---|---|------------------------|
| 0x10000050 | Unused | | | | KEY ₃₋₁ | | | | Data register |
| Unused | Unused | | | | | | | | |
| 0x10000058 | Unused | | | | Mask bits | | | | Interruptmask register |
| 0x1000005C | Unused | | | | Edge bits | | | | Edgecapture register |

Figure 7. Registers used in the pushbuttons parallel port.

The following code included in the main program will set b_3 of the pushbutton PIO interrupt mask register to 1, allowing KEY_3 to trigger interrupts:

```
*(pushbuttons + 2) = 0x8;
```

Note that a standard C compiler recognizes that *pushbuttons* is a pointer to a variable of type `int`, which is 4 bytes long. Thus, the C compiler interprets the value 2 as denoting $2*4 = 8$ bytes. Hence, the above instruction will write the value 0x8 into the addressable location 0x10000058, which is the interrupt mask register.

3.4 The Interrupt Handler

In the sample program, when an interrupt occurs the function *the_isr* is run. This function is included in the file *isr_linkage.c*.

Upon receiving an interrupt request, the processor stops executing its current instruction and begins executing the first instruction of the interrupt handler, which must be located at address 0x90000020 in the DE2 Basic Computer. If *isr_linkage.c* is included in a program that uses interrupts, *the_isr* will automatically be placed at the correct location by the Nios II compiler. The function *isr_linkage* will call *interrupt_handler* which then must determine what caused this interrupt. To determine the cause of the interrupt, the *ipending* (*ctl4*) register must be read. If the k th bit of the *ipending* register is 1, this indicates that the irq_k interrupt has occurred. In the sample program, we need to test b_3 of this register. If b_3 of the *ipending* register is 1, then the *switches_isr()* should be called. This is done as follows:

```
int ipending;
ipending = __builtin_rdctl(4)
if ((ipending & 0x2) == 2)
{
    switches_isr();
}
```

When an interrupt is raised in the sample program, the corresponding bit in the *Edgecapture* register of the pushbutton PIO is 1. Upon completion of the ISR, if the *Edgecapture* register is left alone, this bit will still be 1 and the interrupt will be raised again, even though the pushbutton was not pressed again. Thus, in the function *pushbuttons_isr* the *Edgecapture* register must be set to 0 which is achieved by:

```
*(pushbuttons + 3) = 0;
```

Again note that since *pushbuttons* is a pointer to a variable of type `int`, the addition of 3 will be interpreted as the addition of $3*4 = 12$ bytes.

The state of the switches is displayed on the red LEDs as explained in Section 2 using:

```
*(red_leds) = *(switches);
```

3.5 Loading the Sample Interrupt Program

1. Load the DE2 Basic Computer onto the DE2 board as described in Section 1.
2. Open the Altera Monitor Program, click **Configuration > Configure System** and locate the file *nios_system.ptf* in the same directory as the *DE2_Basic_Computer.qpf*. Since the ISR must be located at location 20 in the memory and the main program is much bigger than 20 bytes, the main program must be located at an offset that gives enough room for the ISR to fit. We chose in this example the offset of the `.text` and `.data` sections to be 1000 and 2000 respectively. Set these parameters and click **OK**.
3. Next, click **Configuration > Configure Program** and select Program Type C. Click **Add...** and browse for the files *pushbutton_interrupt.c* and *isr_linkage.c*. Ensure that the file *pushbutton_interrupt.c* is on the top of the list of files as in Figure 8. The file *pushbutton_interrupt.c* must be on top because the top file determines the name of the generated `.elf` and `.srec` files produced by the compiler.
4. Select **Actions > Compile and Load** to compile the program and load it onto the DE2 board.
5. Select **Actions > Continue** to run the program. Flip some switches and press *KEY₃* on the DE2 board to observe that the red LEDs display the state of the switches when *KEY₃* was pressed.

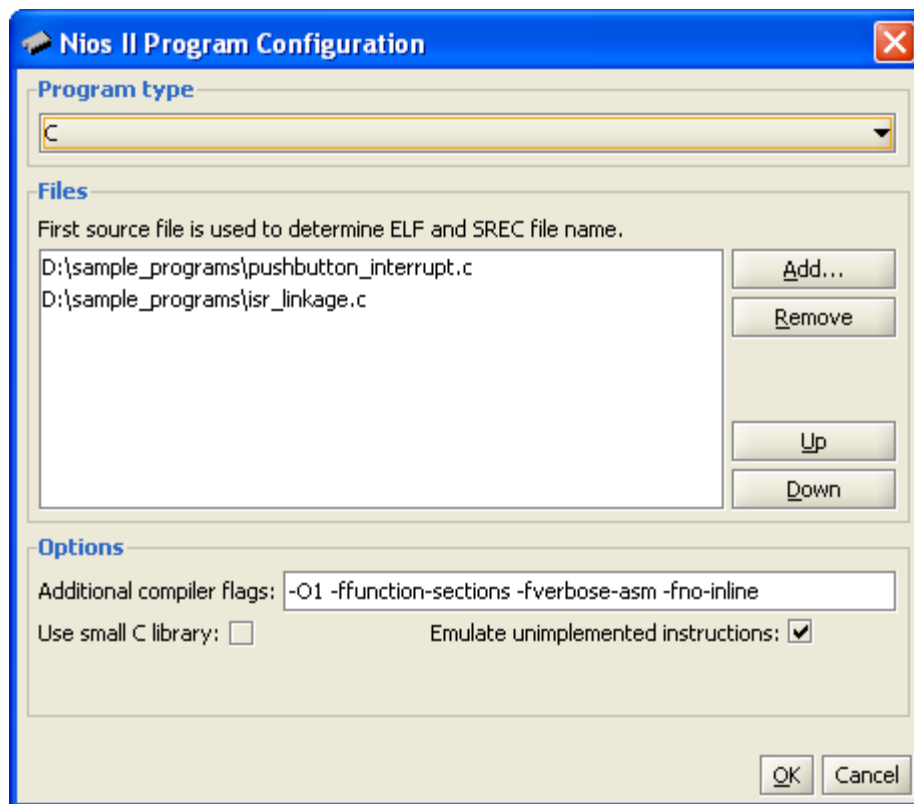


Figure 8. Place *pushbutton_interrupt.c* on top.

Copyright ©2008 Altera Corporation. All rights reserved. Altera, The Programmable Solutions Company, the stylized Altera logo, specific device designations, and all other words and logos that are identified as trademarks and/or service marks are, unless noted otherwise, the trademarks and service marks of Altera Corporation in the U.S. and other countries. All other product or service names are the property of their respective holders. Altera products are protected under numerous U.S. and foreign patents and pending applications, mask work rights, and copyrights. Altera warrants performance of its semiconductor products to current specifications in accordance with Altera's standard warranty, but reserves the right to make changes to any products and services at any time without notice. Altera assumes no responsibility or liability arising out of the application or use of any information, product, or service described herein except as expressly agreed to in writing by Altera Corporation. Altera customers are advised to obtain the latest version of device specifications before relying on any published information and before placing orders for products or services.

This document is being provided on an "as-is" basis and as an accommodation and therefore all warranties, representations or guarantees of any kind (whether express, implied or statutory) including, without limitation, warranties of merchantability, non-infringement, or fitness for a particular purpose, are specifically disclaimed.