University of Applied Sciences Ulm
Computer Science Department
Computer Engineering Laboratory

Embedded Systems Lab Exercise 1:

Simple I/O and Interrupt Programming

Author:	Manfred Strahnen
Version:	6.0
Datum:	18.11.2022

Important Hint:

Each lab exercise has to be prepared. For this lab exercise this means, that you must have:

- read this documentation,

- a written documentation of your answers to assignment 1

BEFORE YOU ENTER THE LAB ROOM. If you are not prepared you are not allowed to attend the appointment. Successful attendance will be certified if you have solved the assignments given in this document and if you have created an expressive lab documentation.

Contents:

1 Goals....................2
2 Initial Situation....................2
3 Assignments....................3
4 Appendix A: NIOS II SBT – Create, build, run and debug a new application project and BSP from template....................7
5 Appendix B: NIOS II SBT – Create a new application project....................8
6 Appendix C: *pio_0* PIO Device....................9
7 Literature....................9

1 Goals

After successful accomplishment of the exercises given in this document you should:

- be familiar with NIOS II SBT development environment,
- be able to develop software which handles simple I/O devices like parallel digital I/O ports (PIO),
- be able to develop software which makes use of interrupts,
- be able to measure interrupt latency with a logic analyzer.

2 Initial Situation

For embedded systems lab exercises we will use the DE1-SoC computer already known from microcomputers course. But this time programming shall be done using C high-level programming language instead of using assembly language. The lab environment, consisting of the DE1-SoC target board, a development PC and a Logic Analyzer device is illustrated in Figure 1 and is described throughout this document.

A detailed description of the DE1-SoC computer system is given in document [1]. Since not all parts of the computer system are used here, you do not need to read entire documentation. There are hints which chapters you have to read. The DE1-SoC computer system was originally developed by Intel/Altera. But we have extended the system. The most important extensions are:

- an I^2C controller has been added,
- expansion ports JP1/GPIO0 and JP2/GPIO1 are used in a different way. GPIO1 is used to connect to LSA-Board, which provides interfaces to Lego Mindstorms sensors and actors. GPIO0 provides access to control signals used to control e. g. the KEY and LED devices of the board. A logic analyzer can easily be connected to the control signals in order to verify correct signal generation and signal timing.

In this lab exercise you have to develop software which controls simple I/O devices like pushbuttons and LEDs which are connected via simple PIO I/O controllers. As an example you have to write a program which polls the status of the buttons, and if a certain button has been pressed, a corresponding LED device shall be switched on. Because 'polling an I/O device' is not an elegant and efficient solution, you later have to extend this software by using interrupts. Interrupt handling will be supported by Altera's board support package BSP. Interrupt handling, especially interrupt latency, has to be verified using a logic analyzer measurement tool.

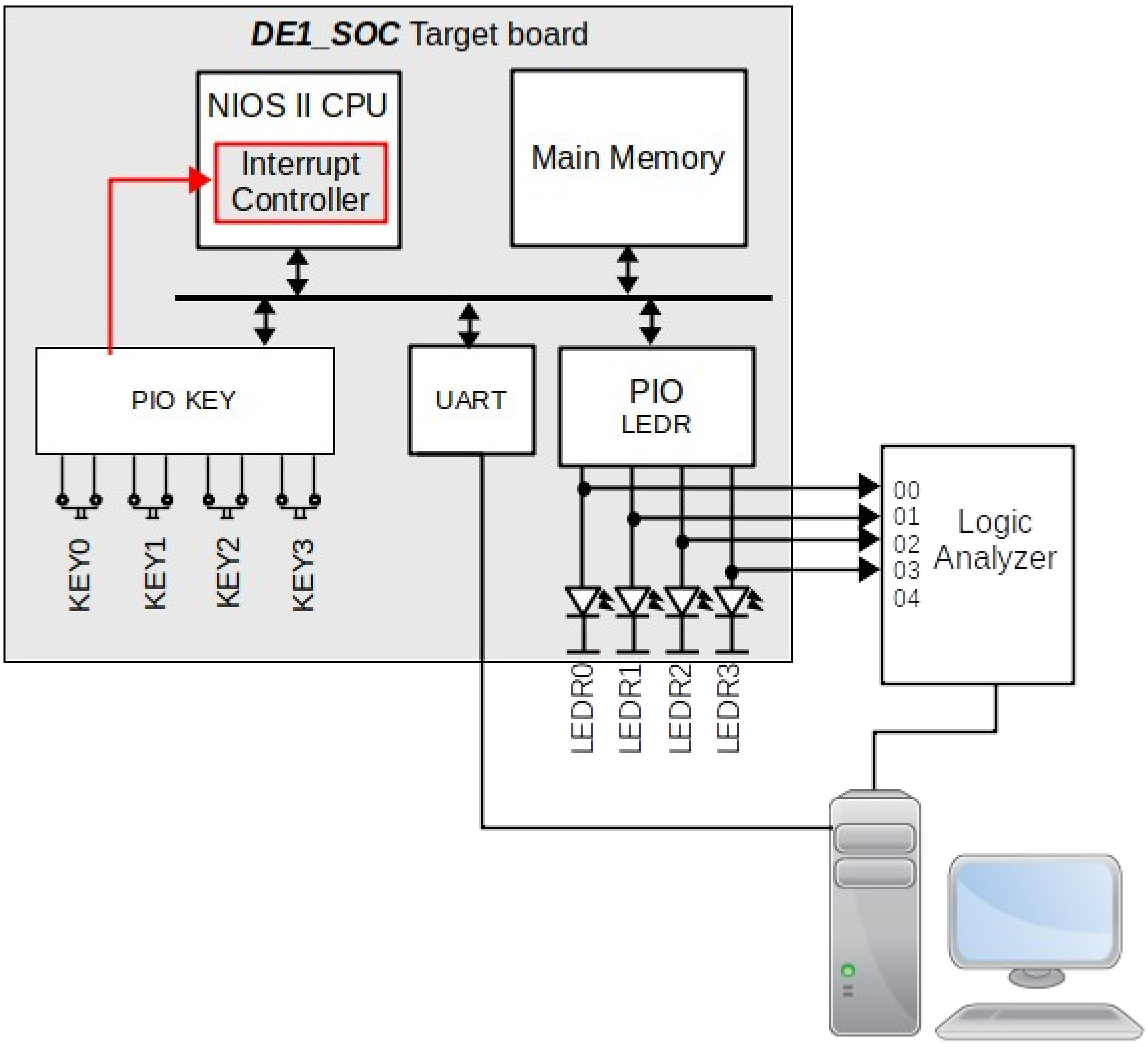

Figure 1: Lab environment

3 Assignments

You should consider following hints:

- You should use the same lab computer for each appointment. On this computer you should create a directory c:\User\Public\<YourName>. Very important: the directory name should NOT contain any spaces! You should use the folder you have just created as your 'working directory', all your files should be stored there.

Assignment 1: (Pre-Assignment)

Please read document [1], chapters:

- 2.2: NIOS II Processor (Do NOT look at the ‘Altera Monitor Program’ mentioned there, because we will not use this program!),
- 2.4: Parallel Ports,
- 2.4.1: Red LED Parallel Port,

- 2.4.4: Pushbutton Key Parallel Port,
- 2.4.6: Using the Parallel Port with Assembly Language and C Code ((only C code is interesting for us, ignore the assembly language code)).

and answer the following questions:

- Within this lab exercise you have to access the I/O registers of the LEDR PIO I/O controller, which controls the red LEDs and the KEY PIO I/O controller, which controls the KEYs. Please give a table with all I/O registers of those controllers. The table should have three columns, first column should specify the I/O controller together with the name of the I/O register, second column should give I/O register's address, and third column should give a short description of the I/O register.
- Figure 9 of document [1] shows an example of C code using parallel ports. Here the attribute 'volatile' is used. Why is it necessary to use the attribute 'volatile' here?
- Figure 9 of document [1] gives an example of C code that uses parallel ports. Author of that program has used data type 'int' to access registers. What is the disadvantage of using 'int' instead of using 'int32_t'?

Assignment 2: (hello world)

Please use the procedure given in chapter 4 (Appendix A) to set up and test a first “hello world” project in order to test the NIOS II SBT software development environment together with DE1-SoC hardware. Setup, compile, download and run the project.

Assignment 3: (PIO devices)

Please write a program which continuously polls the status of the pushbutton devices KEY0, KEY1, KEY2 and KEY3. If one of the pushbuttons is pressed a corresponding LED shall be switched on. If KEY0 is pressed, red LED LEDR0 shall be switched on, if KEY1 is pressed, LEDR1 shall be switched on, and so on. LED status shall stay stable until another key has been pressed. This means that e. g. if LEDR1 has been switched on, because KEY1 has been pressed, LEDR1 shall stay switched on – even if KEY1 has been released in the meantime - until a different is pressed.

In order to do that, you have to set up a new application project. Because hardware has not changed we can reuse the BSP already generated in assignment 2. Chapter 5 (Appendix B) gives a short description of how to set up a new application project and how to link to an existing BSP.

Assignment 4: (Interrupt handling)

The program developed in assignment 3 simply polls the pushbutton devices. As already mentioned, polling an I/O device neither is a very elegant nor a very efficient way to handle I/O devices. A better way would be using interrupts. Fortunately the PIO I/O controller KEY which controls the pushbuttons can be initialized to generate an interrupt anytime one of the buttons has been pressed.

Setup a new application project and write a program which does the same as the program developed in assignment 3, but now using interrupts. Using interrupts is supported by

Altera's BSP and is decribed in document [3]. Further on the source code given below shall assist you in writing the requested software.

Hint: Telling the interrupt requesting device to release it's interrupt request, because the request already has been handled, is done via device's edge capture register. But different to what is specified in chapter 3 of document [1], you have to write 0xF into the edge capture register in order to release the interrupt request.

```
#include <stdio.h>
#include <stdint.h>
#include <sys/alt_irq.h>      // BSP's irq functions


#define KEY_IRQ   1     // interrupt level is fixed by hardware

volatile uint32_t isr_context;   // not really needed

/***************************************************************
 * void isr_keys(void *context, uint32_t id)
 *
 * Interrupt service routine is entered if KEY0, KEY1, KEY2 or
 * KEY3 is pressed. ...
 **************************************************************/
void isr_keys(void *context, uint32_t id)
{
    // Do whatever shall be done in this ISR

    // Instruct interrupt source to release it’s request → Have a look
    // at device's edge capture register

}

int main()
{
      printf("lab1Ex4 has been started\n");

    // unmask KEY interrupts → Have a look at device's interrupt mask reg.

    // register ISR
    alt_irq_register(KEY_IRQ, (void*) &isr_context, isr_keys);

    // Do whatever shall be done in main()

      return 0;
}
```

Assignment 5: (Interrupt latency measurement)

In order to measure interrupt latency, DE1-SoC computer system has been extended by an additional PIO I/O controller, not described in document [1]. Chapter 6 provides a description of the new PIO named *pio0*. Extend your program so that:

- *pio0* is configured as output device,

- *pio0* bit 0 is set to '1' at the beginning of the interrupt service routine and is set to '0' just before leaving it.

Signal key0, which is the KEY0 output signal, as well as *pio0*'s output signals are available at connector GPIO0. GPIO0 pin assignment is given in 1.

Please use the logic analyzer to measure KEY0 related interrupt latency. A *logic analyzer* is a measuring device which can be used to measure the status and the timing of digital signals. A short introduction to our - very easy to use - logic analyzer can be found here:

https://support.saleae.com/getting_started

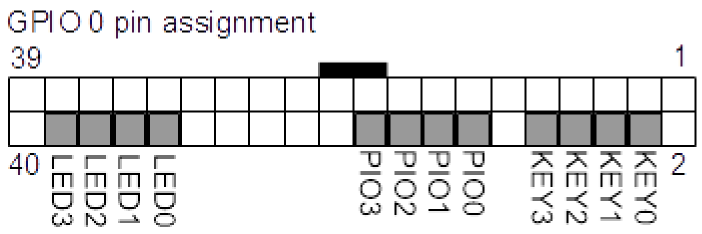

Figure 1: GPIO0 pin assignment

4 Appendix A: NIOS II SBT – Create, build, run and debug a new application project and BSP from template

!!! Important Hints !!!

If you like debugging complex tools or if you like to torture yourself, you can skip over following hints.

- Directory names or file names should have NO special characters like blanks.
- All directories and files have to be installed in your ‘working directory’ on the local disk (NOT a network drive) of your computer system!

NIOS II SBT (Software Build Tools) provide some sample projects (called templates). For the first project you should use following procedure to create a sample project and a BSP project. For later project you can re-use the BSP project and you only have to set up a new application project which is described in chapter 5. Following description is an excerpt of document [2].

- Copy file *p:\strahnen\EmSys\Lab\Templates\Computer_System.sopcinfo* to your (local!) working directory.
- Start 'NIOS II SBT for Eclipse' with:
  Start > All programs > Intel FPGA 18.1.0.625... > NIOS Software Build Tools ...
  Select <yourWorkingDirectory> to be the workspace directory.
- On the File menu, point to **New**, and click **Nios II Application and BSP from Template**. The **Nios II Application and BSP from Template** wizard appears.
- For SOPC Information File name, browse to your workspace directory and open the SOPC Information File *Computer_System.sopcinfo*.
  Unfortunately Eclipse needs some time to process the information given in that file.
- In the **Project name** box, type project's name, e. g. *lab1As2*.
- In the **Templates** list, select the **Hello World** project template.
- Click **Next**. Change BSP **Project name** to: *nios_bareMetal_BSP*.
- Click **Finish**. The Nios II SBT for Eclipse creates the **lab1As2** and the **nios_bareMetal_BSP** projects and returns to the Nios II perspective. *lab1As2* project is the application project.
- In the Project Explorer view, expand **lab1Ex1**. Double-click **hello_world.c** to view the source code.

In order to build, run and debug the just created project, you have to:

- To build the program, right-click the **lab1As2** project in the Project Explorer view, and click **Build Project**. The **Build Project** dialog box appears and the Nios II SBT for Eclipse begins compiling the project. When compilation completes, the message “Build completed” appears in the Console view. The completion time varies depending on your system.

- To run the program, right-click **lab1As2**, point to **Run As**, and click **Nios II Hardware**. The Nios II SBT for Eclipse downloads the program to the FPGA on the target board and executes the code. The message “Hello from Nios II!” displays in the *Nios II Console* view.
- To debug the program, right-click **lab1As2**, point to **Debug As**, and click **Nios II Hardware**. The Nios II SBT for Eclipse downloads the program to the FPGA on the target board, changes the 'perspective' and stops program at main. You can now set breakpoints, go to breakpoints, inspect content of variables, single step through the code and so on.

5 Appendix B: NIOS II SBT – Create a new application project

!!! Important Hints !!!

If you like debugging complex tools or if you like to torture yourself, you can skip over following hints.

- Directory names or files names should have NO special characters like blanks.
- All directories and files have to be installed in your ‘working directory’ on the local disk (NOT a network drive) of your computer system!

NIOS II SBT (Software Build Tools) allows to create a new application project and to link that project to an existing board support package. Following description is an excerpt of document [2].

- Start 'NIOS II SBT for Eclipse' with:
  Start > All programs > Intel FPGA 18.1.0.625... > NIOS Software Build Tools ...
  Select <yourWorkingDirectory> to be the workspace directory.
- On the File menu, point to **New**, and click **Nios II Application**. The **Nios II Application** wizard appears.
- In the **Project name** box, type project's name, e. g. *lab1As3*.
- In the **BSP location** select box select an existing BSP, e. g. nios_bareMetal_BSP.
- *Used default location* should automatically be adjusted to:
  <yourWorkingDirectory>\lab1As3.
- Click **Finish**. The Nios II SBT for Eclipse creates the **lab1As3** project and returns to the Nios II perspective.
- In the Project Explorer view, right-click **lab1As3** and click **New** → **Source file**. In the than appearing **Source file view** enter **source file**'s name, e. g. lab1As3.c, and select **Default C source template**.
- Open and edit the just created source code file. Compilation, execution and debugging project has already been described in Chapter 4.

6 Appendix C: *pio_0* PIO Device

pio0 is a bidirectional PIO device with 4 in/out signal lines, which by default are configured as inputs. *pio0* provides the control/status registers described in table 6-1. Content of device's data register indicates the status of those signal lines configured as input. By writing to device's data register the status of those signal lines configured as output can be fixed.

pio0's direction register can be used to configure each pin as input or output. A rising edge on one of the input/output signals lines may optionally be used to generate an interrupt request to the CPU. Interrupts can be enabled/disabled for any PIO input individually by setting/resetting the corresponding bit in PIO's *interruptmask* register. If an interrupt has been generated the inputs responsible for that interrupt will be indicated in so-called *edgecapture* register. A bit set to '1' indicates that a rising edge and thereby an interrupt request has been active on the corresponding input port. After the interrupt has been recognized by the CPU this bit has to be cleared by writing value '1' into the corresponding bit position. This demands the PIO device to release its interrupt request.

pio0's I/O signals, named PIO0, PIO1, PIO2 and PIO3 are available at connector GPIO0.

Address	*Register*	*Description*			
		3	*2*	*1*	*0*
0xff204060	data	Status of I/O ports PIO0, PIO1, PIO2 and PIO3			
0xff204064	direction	Configure direction of I/O ports (BitX = 0: Port X is configured as input, BitX = 1: Port X is configured as output)			
0xff204068	interruptmask	Enable/Disable interrupts for corresponding input port (BitX = 1: Interrupts enabled for port X, Bit x = 0: Interrupts disabled for port X)			
0xff20406C	edgecapture	Indicates active interrupt channels (BitX = 1: Interrupt request active for port X)			

Table 6-1: Registers of pio0 device

7 Literature

[1] Intel/Altera: "DE1-SoC Computer System with NIOS II", public\strahnen\EmSys\Lab\Doku\DE1-SoC_Computer_NIOS.pdf

[2] Intel/Altera: "My First Nios II Software Tutorial", public\strahnen\EmSys\Lab\Doku\tt_my_first_nios_sw.pdf

[3] M. Strahnen lecture notes: "II-5: Case Study: NIOS II Board Support Package", public\strahnen\EmSys\Lectures\emsys_II-5_BSP-Case-Study_v10.pdf