

Namn:
Laborationen godkänd:

Computer Organization 6 hp



LUNDS TEKNISKA HÖGSKOLA
Lunds universitet

Introduction to the lab environment

Purpose

The purpose of this laboratory exercise is to give an introduction to the lab environment that will be used throughout the labs in the Computer Organization course EITF70. One should get the knowledge on how to use the Xilinx SDK tool to setup, debug and run application projects. Furthermore, one should get the knowledge on how to write a simple application program in the high level programming language C, understand the different data types and how data is stored in memory.

The Lab Environment

Throughout the labs of this course we shall use the Xilinx SDK tool to create different applications that will be executed on a specific hardware platform that is implemented on an FPGA board. The application projects are to be written in the high level programming language C. The applications are to be executed on a hardware platform specifically tailored for the needs of this course. The hardware platform is based on a 32-bit RISC (Reduced Instruction Set Computer) processor, MicroBlaze. The MicroBlaze processor is a soft microprocessor core designed by Xilinx. The MicroBlaze core can be used to design hardware that can be implemented on a Xilinx FPGA (Field Programmable Gate Array) board. An FPGA is an integrated circuit that contains an array of programmable logic blocks. By programming the FPGA one can implement different hardware designs that are specified using a hardware description language (HDL). In these labs, we shall use the Nexys 4 FPGA board. Apart from the FPGA, this board contains a number of ports and peripherals allowing it to host designs ranging from introductory combinational circuits to powerful processors. Throughout the labs, we shall use primarily the following peripherals available on the Nexys 4 board: the 16 switches, the 16 LEDs, the eight seven-segment displays and the five pushbuttons. The Nexys 4 board, along with the peripherals discussed earlier, is illustrated in **Figure 1**.

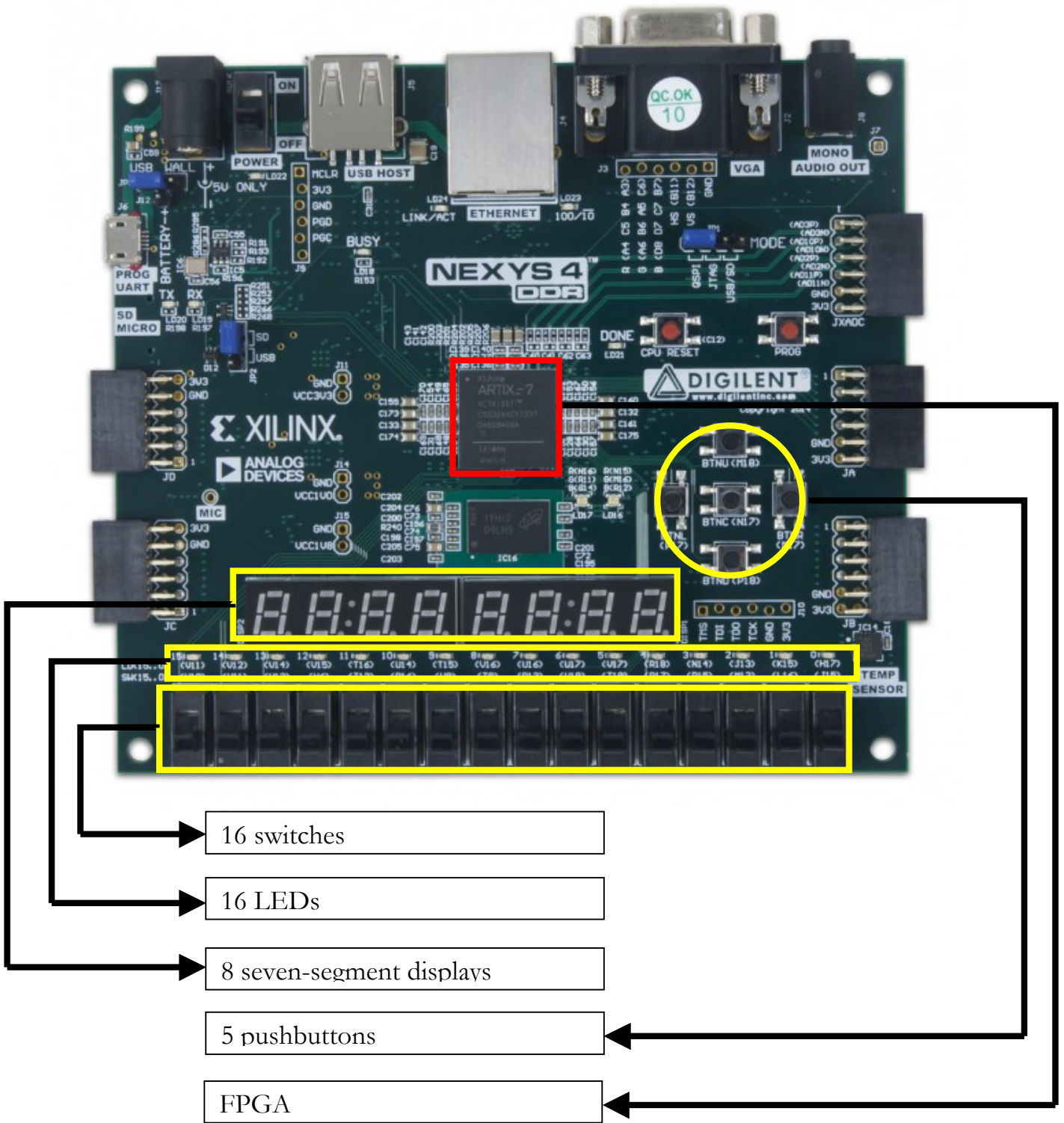


Figure 1. Nexys 4 board

The Hardware Platform

The hardware platform is based on a MicroBlaze microprocessor. The processor is connected to a number of I/O devices and a memory. The following I/O devices are available:

- *SWITCHES*, a configurable I/O device interfacing the 16 switches available on the Nexys 4 board
- *LEDS*, a configurable I/O device interfacing the 16 LEDs available on the Nexys 4 board
- *7SEGMENT DISPLAYS*, an output device interfacing the 8 seven-segment displays available on the Nexys 4 board
- *PUSH BUTTONS*, a configurable I/O device interfacing the five pushbuttons available on the Nexys 4board
- *TIMER1*, a configurable timer/counter device that can generate interrupts
- *TIMER2*, a configurable timer/counter device that can generate interrupts
- *INTERRUPT CONTROLLER*, used for combining multiple interrupts that come from various devices, namely, the timers.

The memory stores the program that is to be executed by the MicroBlaze microprocessor. Furthermore, all necessary data is stored in the memory. A block representation of the hardware platform is given in **Figure 2**.

The MicroBlaze microprocessor operates at 100MHZ frequency.

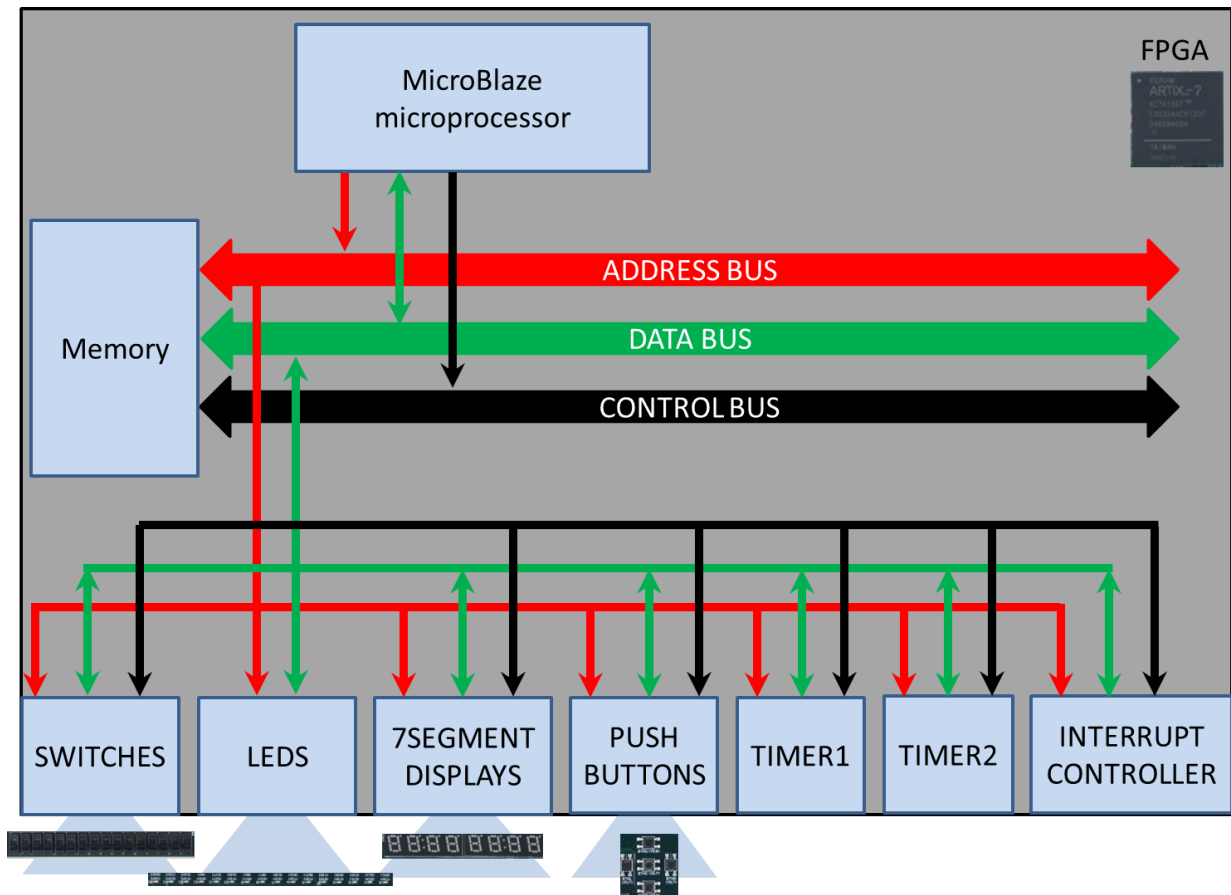


Figure 2. The hardware platform

The Xilinx SDK

This part guides you through the process of setting up a new project using the Xilinx SDK tool. It includes the different steps, namely, setting up the workspace, creating a new project, setting up the target hardware platform, writing the first application, programming the FPGA and running/debugging the application.

Setting up the workspace for Xilinx SDK

In the file explorer, create a folder with the name “EITF70” (or choose a different name that does not contain any other characters except for digits and letters) in the following location “C:\Users*user_name*\”, where *user_name* is your login ID. All projects that will be created during the labs of this course will be stored in this folder, and it is referred to as a workspace. When you open Xilinx SDK, the following window pops up:

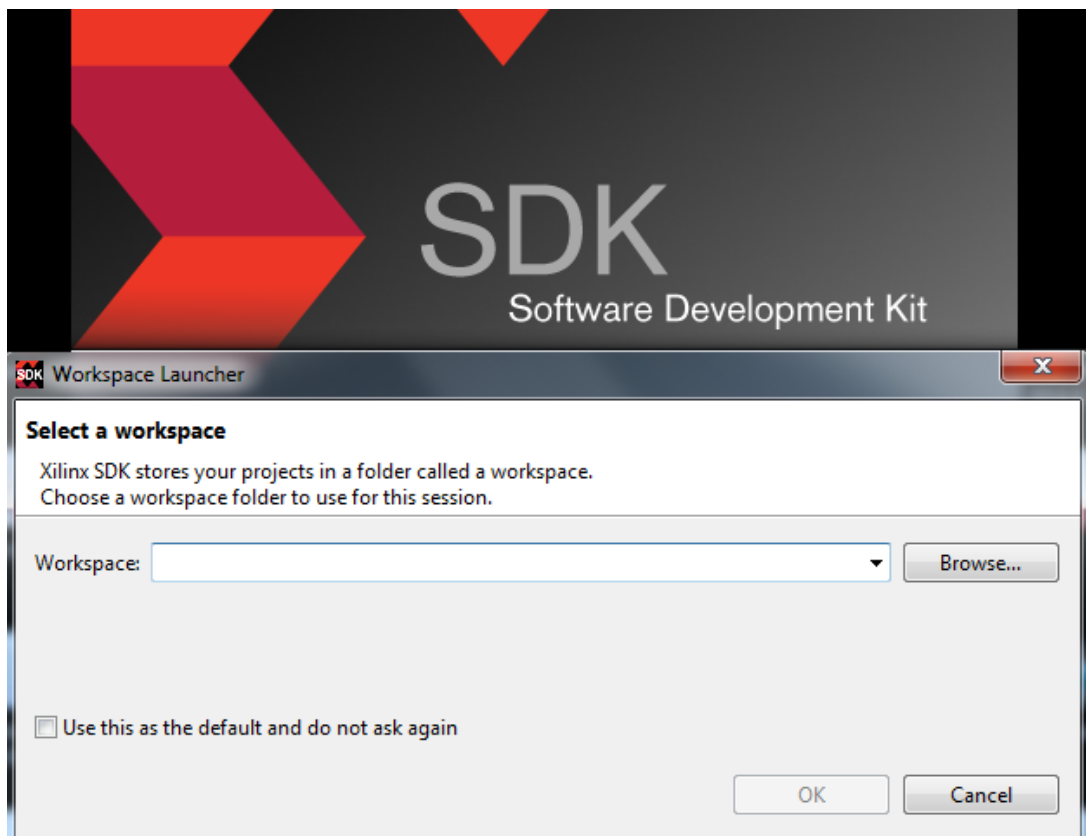


Figure 3. Setting up the workspace

Click the “Browse” button, and locate the folder that you have just created, or just type in “C:\Users*user_name**folder*”, where *user_name* is your login ID and *folder* is the name of the folder that you have created. Next, click the “OK” button.

Note that the workspace is located on the local drive “C:\” and hence, it only exists on the particular machine on which you are working. To enable portability, once you are done with a lab session, copy the workspace to your network drive “H:\”. Next time you log in to a different machine, copy the workspace from your network drive “H:\” to “C:\Users\user_name\” to resume the work from the last lab session.

Creating a new project

To create a new project, from the File menu, select new application project, as illustrated in **Figure 4**.

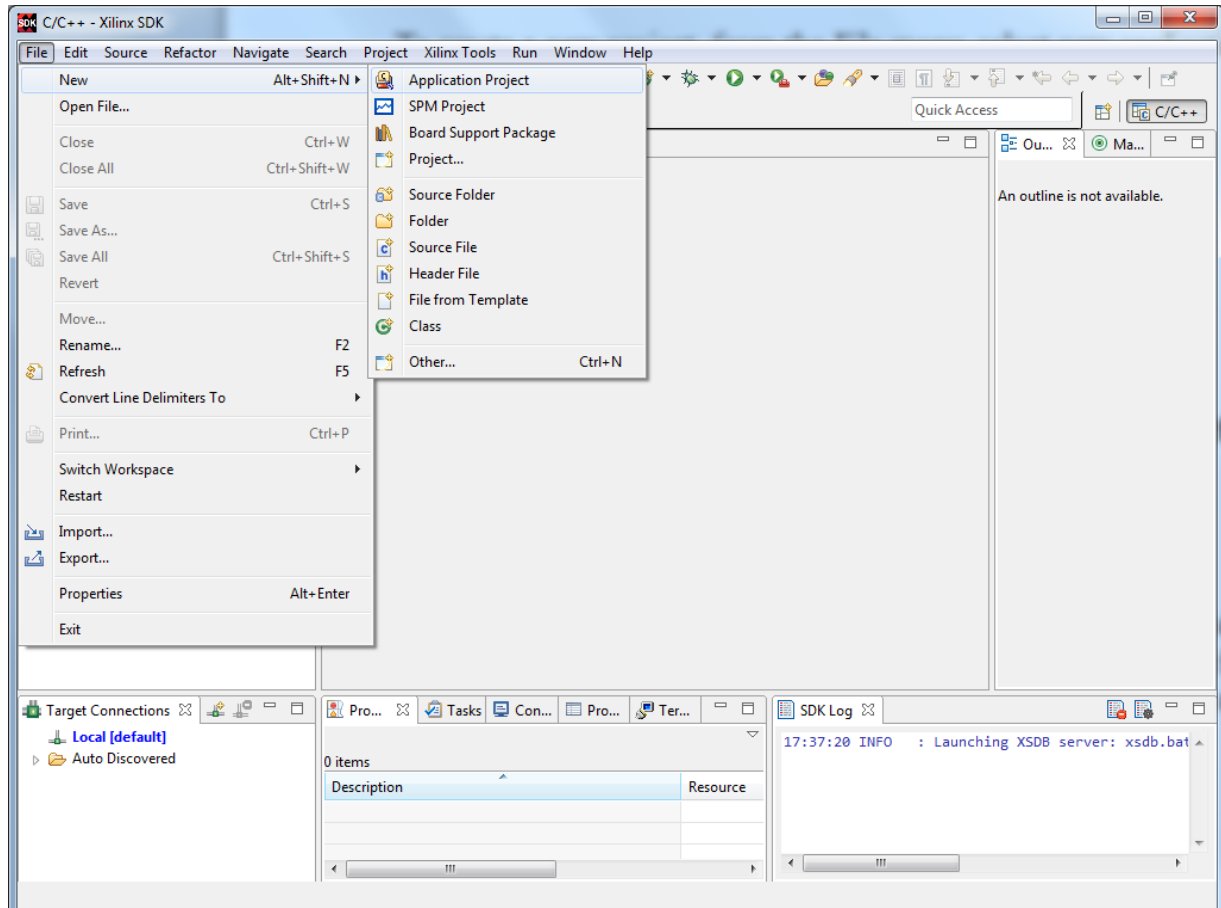


Figure 4. Creating a new project

The applications that will be created throughout the labs will be written in C language. All applications should be executed on a hardware platform that has been specifically designed for the purpose of this course. The hardware platform is based on a MicroBlaze processor, an intellectual property (IP) softcore provided by Xilinx. Observe that when you start using the Xilinx SDK tool, there is a set of predefined hardware platforms that you can choose from. However, the hardware platform that you need to use is not available in this list. Therefore, you first need to add the new hardware platform.

Setting up the target hardware platform

To add the new hardware platform, download the hardware specification that is available on the course web-page (“[MicroBlazeHWPlatform.hdf](#)”) and save it in “C:\Users*user_name*\”, where *user_name* is your login ID. Note that this file must have the extension “.hdf”. Next, in the Target Hardware section, click the “New” button as illustrated in **Figure 5**.

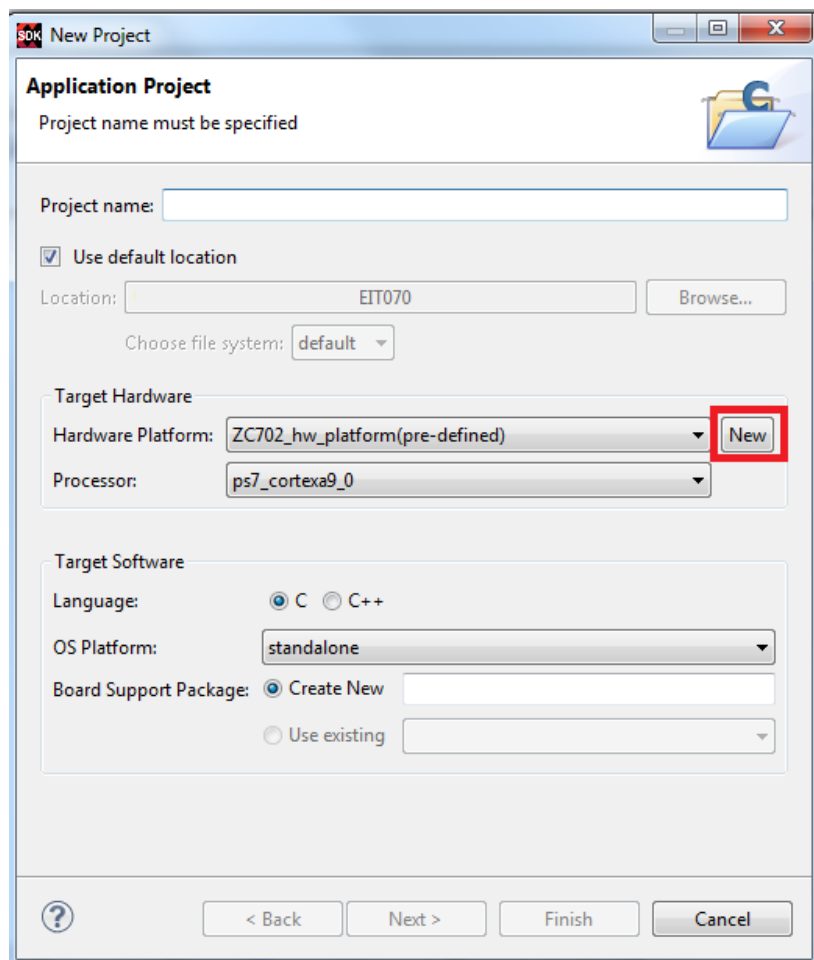


Figure 5. Adding a new hardware platform

By following these steps, you are creating a new hardware project. Name this project as “MicroBlazePlatform”. For the hardware specification, click the “Browse” button and locate the hardware specification file that you downloaded from the course web-page. This is illustrated in **Figure 6**.

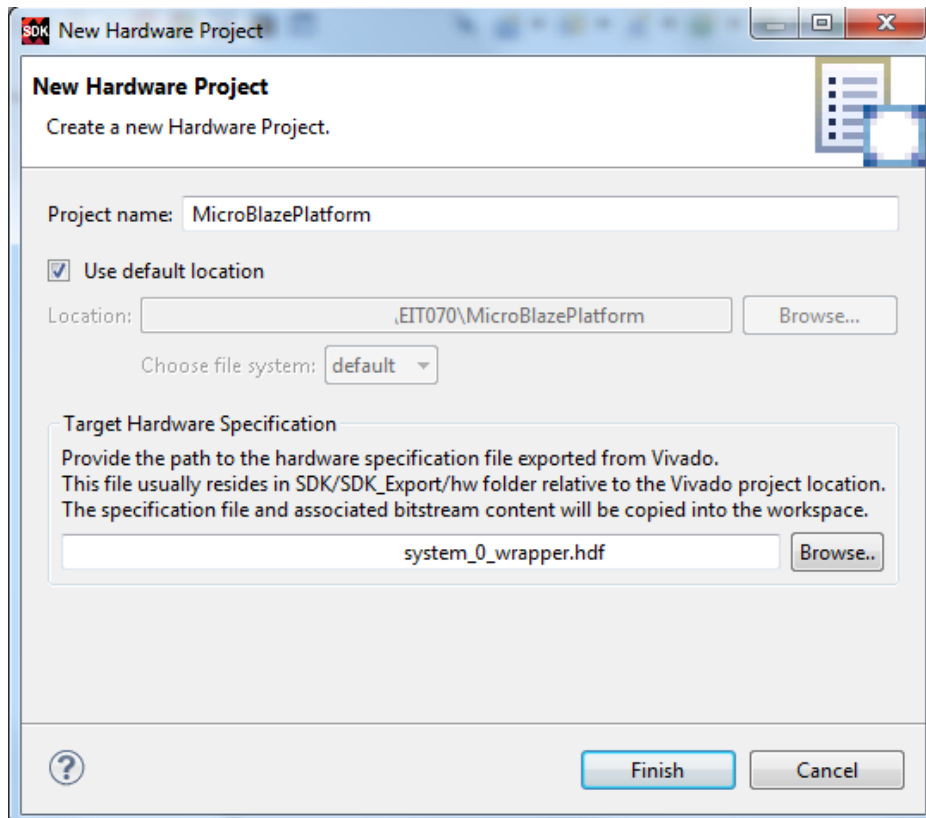


Figure 6. Creating a new hardware project

By completing these steps, you have successfully created a new application project that runs on the hardware platform designed for the purposes of this course.

By using the same workspace, next time when you create a project, the target hardware platform will be available in the drop-down list, and thus, it is sufficient to do this step only once. However, the labs are organized such that you only need one project that you will use in the remaining laboratory exercises.

To be able to write an application in your application project you need to add a source file where you write the code. To add a source file to the project, in the Project Explorer, select the “**src**” folder of the project to which you want to add the source file, click the right mouse button, and select new source file. These steps are illustrated in **Figure 7**. For future reference, name this file as “**main.c**”. This file will contain the main program that will be executed when you run the project.

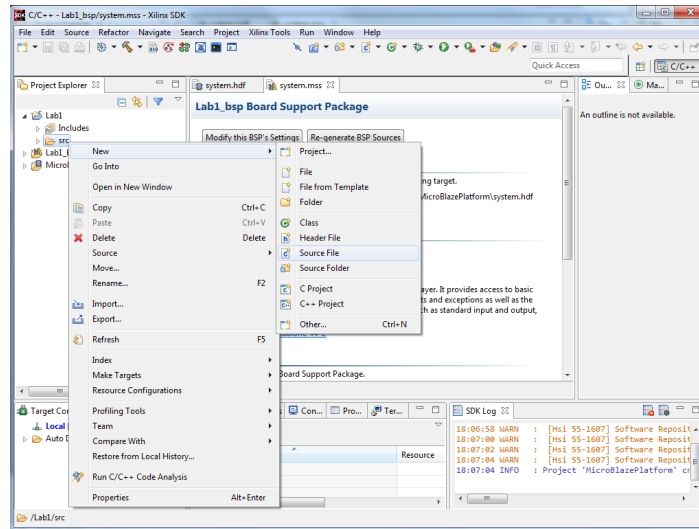


Figure 7. Adding a new source file

The first C program


In the recently created source file, i.e. “main.c”, type the following code:



```
char b;
int main(){
    b=9;
    while (1==1){
        if (b<10){
            b--;
        }
    }
}
```

Analyze the code.

- What does the code do?
- What is the expected value of the variable “b”?

Running/debugging the program

Before you can run/debug the program, two steps need to be performed. First, you need to program the FPGA of the Nexys 4 board according to the provided hardware specification. To do so, click the “Program FPGA”  button. Observe that this process is only done as long as the FPGA has not been programmed before. Once the FPGA is programmed, i.e. the hardware platform is implemented on the FPGA, the same hardware platform will be available until the board is disconnected from the computer. Initially, the FPGA board is not programmed.

Once the FPGA is programmed, the second step is to build the project. This step is responsible to convert the source files into executable files that later can be executed by the MicroBlaze processor. To build the project, click the “Build All”  button. Make sure that all the changes in the source file are saved, before you build the project. To save the changes in a source file, click the “Save All”  button. By default, whenever you save the changes in the source files, a build is performed. Now, you are ready to run/debug the program. To run/debug the program, in the Project Explorer, select the project application that you want to run, click the right mouse button and select “**Debug As**” → “**4 Launch on Hardware (GDB)**”. These steps are illustrated **Figure 8** (Note: The option GDB was numbered as 1 in an older version of the tool, however in the latest version GDB is numbered as 4).

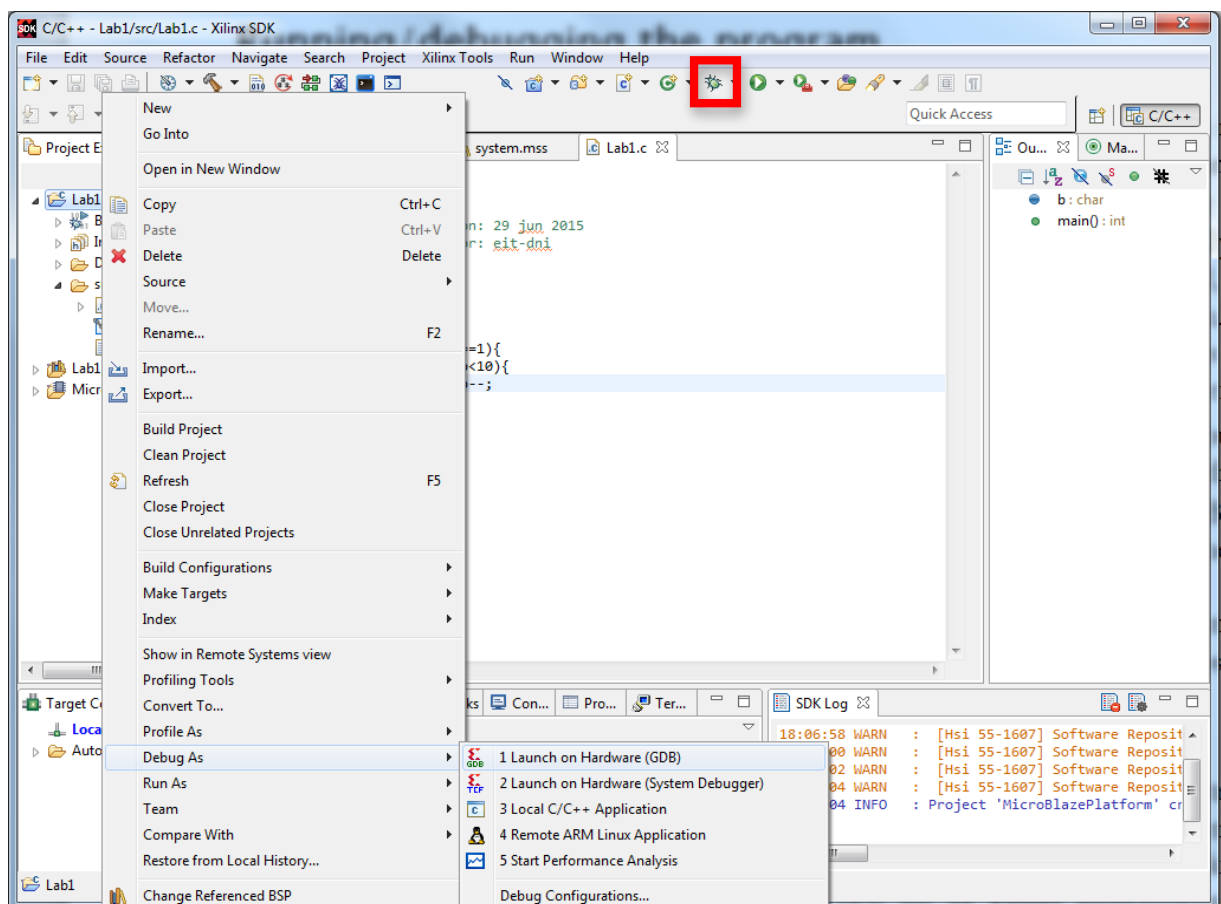



Figure 8. Running/Debugging the application

To run/debug the program you can also use the shortcut button , where from the drop-down list you select “**Debug As**” → “**4 Launch on Hardware (GDB)**”. Note that if you want to use the shortcut button, make sure that the correct project is selected in the Project Explorer.

When debugging the project, the tool asks you to open the Debug Perspective. Check the “Remember my decision” box, and click the “Yes” button to proceed.

The Debug Perspective allows you to control the execution of your program by setting breakpoints, suspending launched programs, stepping through your code, and examining the contents of variables.

Initially, the program is suspended on the first line of the main function, highlighted in green, as shown in **Figure 9**.

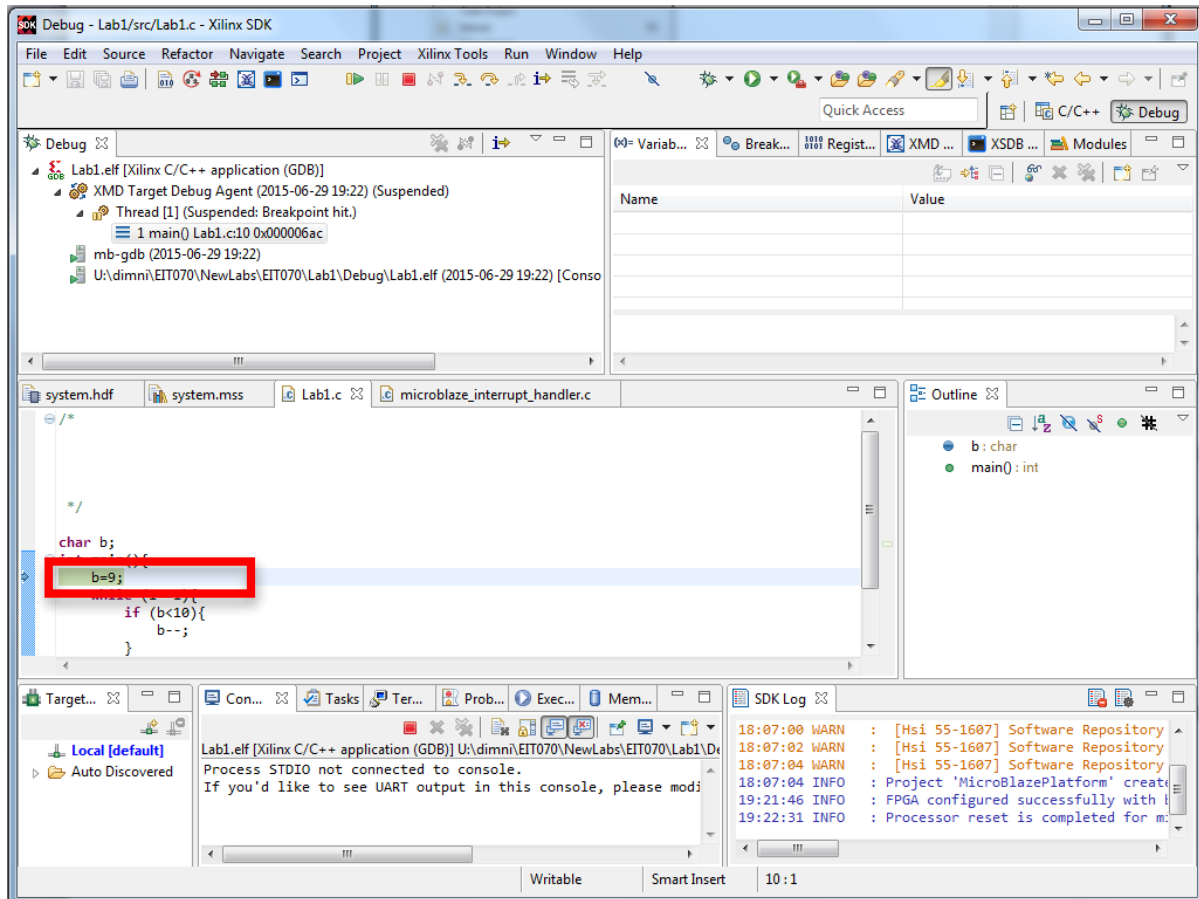


Figure 9. Debugging the main program

By adding expressions, you can trace the changes of different expressions, or variables, as the program is running. To open the expression view, from the Window menu, select Show View→Expressions. To add a new expression, click the “Add new expression” button and type the expression that you would like to evaluate. To evaluate the current value of a given variable, type the name of the variable. An example to evaluate the current value of the variable “b” is provided in **Figure 10**.

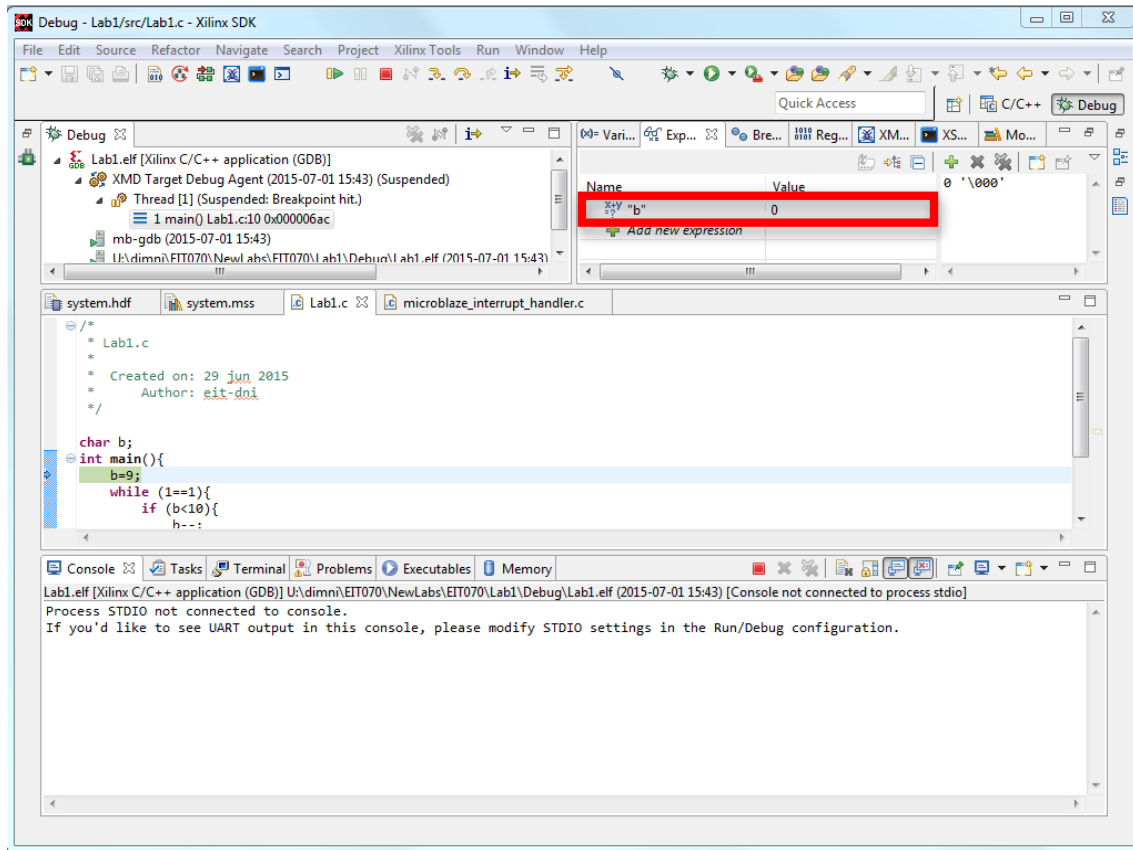








Figure 10. Evaluating the value of a variable

Observe that you can change the formatting of the value to either see the default, decimal, hexadecimal, octal or binary representation. To do so, on the Value field, in the Expression view, click the right mouse button, select Format, and change the formatting.

At this point, the program is suspended, i.e. it is not running. From this point, you can either step through the code or you can let the program run. To step through the code, click the “Step Into”  button. After each step, the program is again suspended. If you decide to run the program, click the “Resume”  button. If you want to suspend the program from running, click the “Suspend”  button.

Click the “Resume”  button and let the program run for a while, and then click the “Suspend”  button.

- What is the value of the variable “b”? _____
- What is the binary representation of “b”? _____

Terminate the program by clicking the “Terminate”  button.

In the Debug Perspective, you can also examine/modify the contents of the memory. To do this, first start debugging the program. Next, in the Debug Perspective, from the Window menu, select Show View → Memory. In the Memory

view, you can add a monitor, to inspect/modify a given memory location. To add a monitor, click the “Add Memory Monitor” button. A pop-up box appears as shown in **Figure 11**. Type in the address “0x1a80” and click the “OK” button.

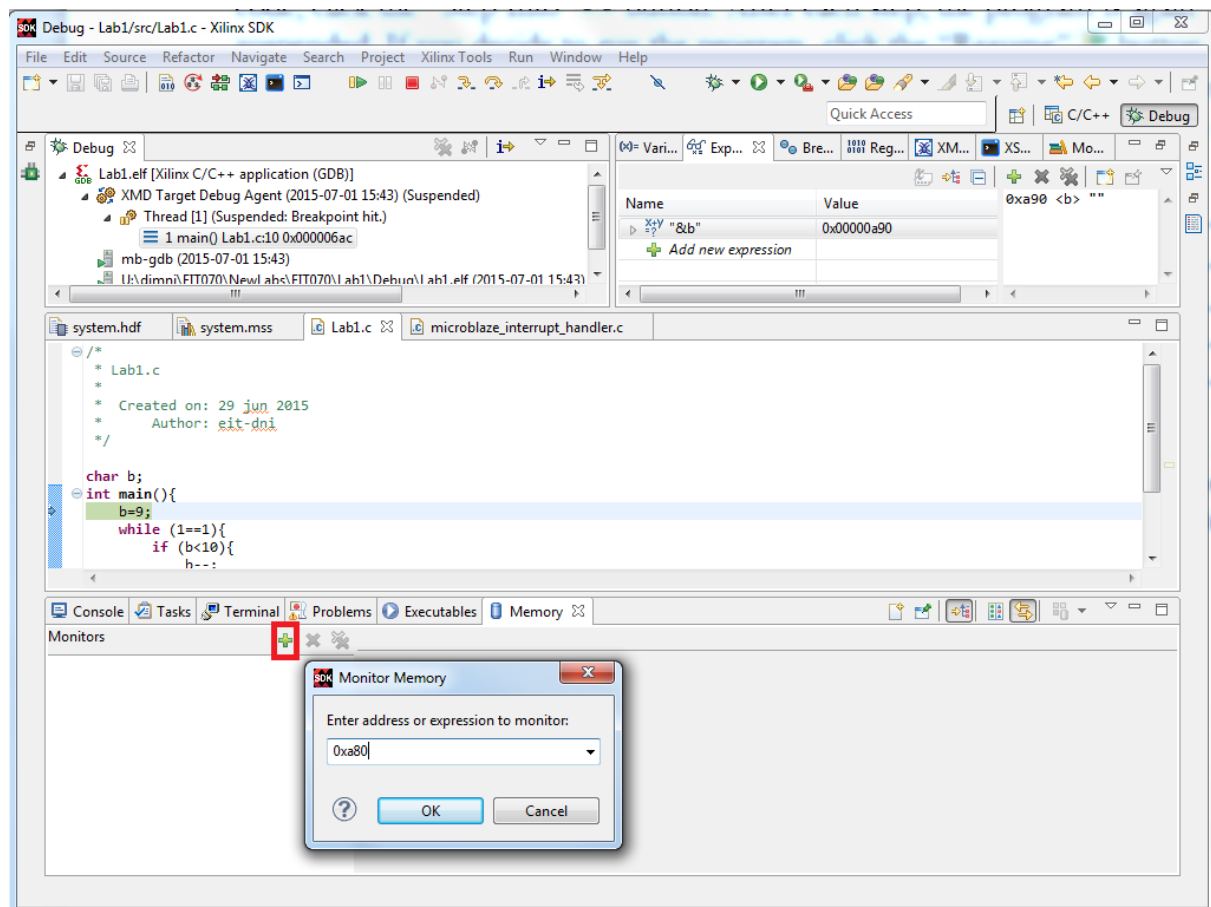


Figure 11. Adding a memory monitor

The memory is byte addressable (8 bits are stored at each memory address). The memory is displayed as a matrix, where the memory address is obtained by summing up the row number and the column number.

By default, the memory monitor shows a view where 4 bytes (8 hexadecimal digits) are combined together, and 4 such entries are displayed per row. However, the Memory view allows you to change the layout, i.e. how many entries per row, how many columns to be combined to present an entry. To change the layout, position the mouse over the memory, click the right mouse button, and select “Format”. In the pop up window, you can specify the row size, i.e. the number of units (bytes) to be displayed per row, and you can select how many units (bytes) to be displayed per column. In **Figure 12**, we show the settings for a layout where one word (4 bytes) is displayed per row and each column displays one byte. Click the “Save as Defaults” button to have this layout as default.

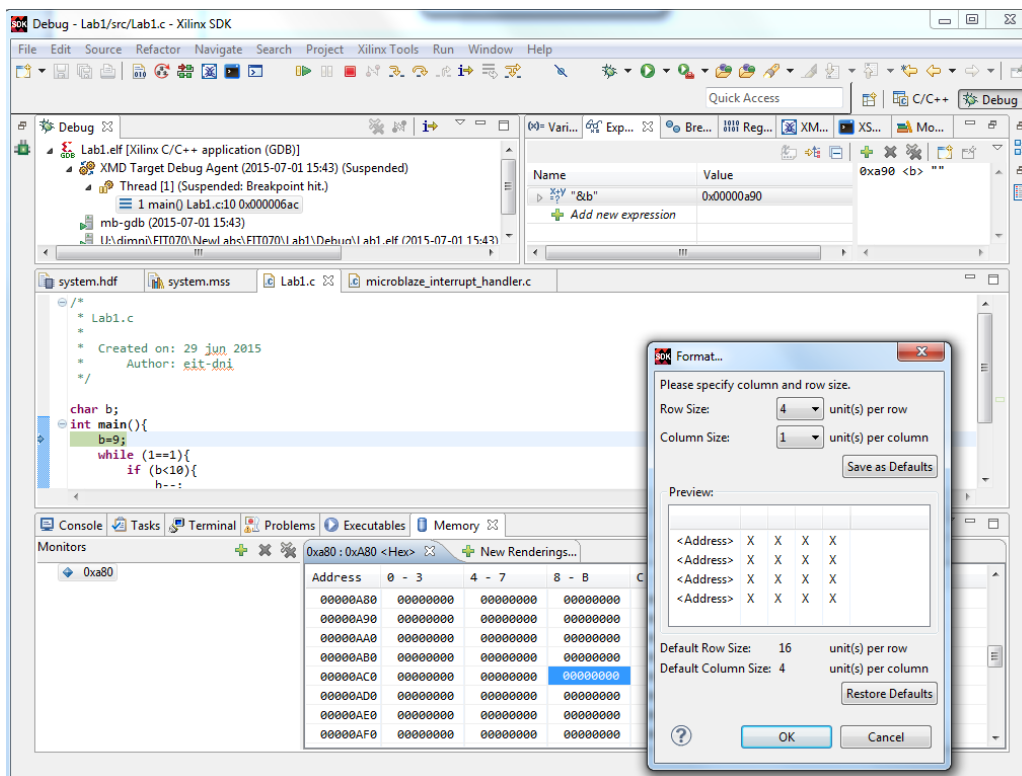



Figure 12. Changing the layout of the memory window

Next, step through the program. Answer the following questions.

- Are there any changes in the memory? _____
- Which memory address is affected? _____
- Compare the contents of the variable “b” with the contents of the affected memory address. What is the conclusion? _____
- What is stored at that memory address? _____
- Modify the contents of the memory address next to the affected memory address by typing a non-zero value at that address. Does this have any impact on the variable “b”? _____
- How many bits are required to store a variable of type char? _____

Run the program.


- Why is the value of the variable “b” larger than 10? _____

Terminate the program by clicking the “Terminate”  button.


In the rest of the assignments of this laboratory exercise we shall examine the different data types in C, and how they are stored in memory. Before you start with a new assignment, comment the contents of the “main.c” file. A comment starts with “/*” and ends with “*/”. At the beginning of the comment put a note such that you know to which assignment the commented code belongs to.

Assignment 1.

The purpose of this assignment is to analyze the “unsigned char” data type.

Open the C/C++ perspective, by clicking the  icon in the top-right corner. Comment the contents of the “main.c” file, and add the source code, as shown in the figure below.

```
unsigned char b;
int main(){
    b=9;
    while (1==1){
        if (b<10){
            b--;
        }
    }
}
```

Build the project after you have done the changes, and then debug the program. In the debug perspective, run the program by pressing the “Resume”  button.

- At which memory address is the variable “b” stored? _____
- What is the value of the variable “b”? _____
- Why is the value different compared to the previous case? _____

- How many bits are required to store a variable of type unsigned char? _____


Terminate the program, and switch to C/C++ perspective.

Assignment 2.

The purpose of this assignment is to analyze the “int” data type.

Comment the contents of the “main.c” file, and add the source code, as shown in the figure below.

```
int b;
int main(){
    b=9;
    while (1==1){
        if (b<10){
            b--;
        }
    }
}
```


Build the project after you have done the changes, and then debug the program. In the debug perspective, step through the program by pressing the “Step Into”  button.

- At which memory address is the variable “b” stored? _____
- Modify the contents of the memory address next to the affected memory address by typing a non-zero value at that address. Does this have any impact on the variable “b”? _____

Undo the modifications from the previous step, and keep on stepping through the program, until the value of “b” has reached the value of zero. Step through again and make sure that the statement “b--” is executed.

- What happens in the memory? _____
- How many bytes are needed to store a variable of type int? _____

Run the program.

- What is the value of the variable “b”? _____
- What is the binary representation of “b”? _____
- What is the hexadecimal representation of “b”? _____
- At which address is the most significant byte of the variable “b” stored? _____
- At which address is the least significant byte of the variable “b” stored? _____
- Which format is used (big vs. little endian)? _____


Terminate the program, and switch to C/C++ perspective.

Assignment 3.

The purpose of this assignment is to analyze the “unsigned int” data type.

Comment the contents of the “main.c” file, and add the source code, as shown in the figure below.

```
unsigned int b;
int main(){
    b=9;
    while (1==1){
        if (b<10){
            b--;
        }
    }
}
```

Build the project after you have done the changes, and then debug the program. In the debug perspective, step through the program by pressing the “Step Into”  button.

- At which memory address is the variable “b” stored? _____
- Modify the contents of the memory address next to the affected memory address by typing a non-zero value at that address. Does this have any impact on the variable “b”? _____

Keep on stepping through the program, until the value of “b” has reached the value of zero. Step through again and make sure that the statement “b--” is executed.

- What is the value of the variable “b”? _____
- What happens with the memory? _____
- How many bytes are needed to store a variable of type unsigned int? _____

Run the program.

- What is the value of the variable “b”? _____
- What is the difference when the keyword “unsigned” is used? _____


Terminate the program, and switch to C/C++ perspective.

Assignment 4.

The purpose of this assignment is to analyze the array data type. In particular, this assignment focuses on an array of elements where each element is of type `char`.

Comment the contents of the “main.c” file, and add the source code, as shown in the figure below.

```
char b[10];
int main(){
int i;
    while (1==1){
        for (i=0;i<10; i++)
            b[i]=i;
    }
}
```

Build the project after you have done the changes, and then debug the program. In the debug perspective, step through the program by pressing the “Step Into”  button. Open the memory view and observe what happens as you step through the program.

- What is the value of the variable “b”? _____
- What does this value represent? _____
- Where in the memory, are the elements of the array stored? _____

- What is the address of the element `b[3]`? _____
- What is the size of each element of the array? _____


Terminate the program, and switch to C/C++ perspective.

Assignment 5.

The purpose of this assignment is to analyze the array data type. In particular, this assignment focuses on an array of elements where each element is of type `int`.

Comment the contents of the “main.c” file, and add the source code, as shown in the figure below.

```
int b[10];
int main(){
int i;
    while (1==1){
        for (i=0;i<10; i++)
            b[i]=i;
    }
}
```

Build the project after you have done the changes, and then debug the program. In the debug perspective, step through the program by pressing the “Step Into”  button. Open the memory view and observe what happens as you step through the program.

- What is the value of the variable “b”? _____
- What does this value represent? _____
- Where in the memory, are the elements of the array stored? _____
- What is the address of the element `b[3]`? _____
- What is the size of each element of the array? _____

Terminate the program, and switch to C/C++ perspective.

Pointers

As shown with the previous examples, variables of different types (int,char) are stored in memory. A memory address is associated to each variable, and the value of the variable is stored at that memory address. In C, a special operator “&” (address of) is used to obtain the memory address at which a variable is stored. For example, given that a variable “b” is declared, the memory address where this variable is stored can be obtained by using the expression “&b”. Next, we discuss a new type of a variable, i.e. a *pointer*. A *pointer* is a variable whose value is interpreted as a memory address. This allows a pointer to read/change the contents of the memory address to which it points to. Since at a given memory address a specific data type is stored, when a pointer is declared, it is declared such that it *points* to a memory address where a specific data type is stored. In C, a pointer is declared as:

```
<data_type> *pointer_name;
```

The previous line declares a pointer (pointer_name) that points to a memory address where a specific <data_type> resides. By using the declared pointer, we can access (read or write) the memory address to which it points. The following line shows how to write to a memory location where the pointer points to:

```
*pointer_name=5;
```

Next, we show an example on how to use pointers.

Assignment 6.

The purpose of this assignment is to get a better understanding on pointers. In particular, in this assignment we shall use an example of a pointer that points to char type.

Comment the contents of the “main.c” file, and add the source code, as shown in the figure below.

```
char *address;
char b;
int main(){
    b=0;
    address=&b;
    *address=5;
    while (1==1){
    }
}
```

Build the project after you have done the changes, step through the code, and after each step answer the following questions.

- What is the value of the variable “b”? _____
- What is the value of the variable “address”? _____

Check the contents of the memory at the address pointed by the variable “address”.

- What is stored in the memory at that memory address? _____
- What is the value of the variable “b”? _____
- What is the memory address of the variable “address”? _____

Assignment 7.

Comment the contents of the “main.c” file, and add the source code, as shown in the figure below.

```
char *address;
int b;
int main(){
    b=0xFFFFFFFF;
    address=(char *)&b+3;
    *address+=0xF0;
    while (1==1){
    }
}
```

Build the project after you have done the changes, and then step through the code.

Explain what this code does.

I/O handling

In many computer systems, the processor (CPU) communicates with other devices. These devices may be input devices (keyboard, mouse, scanner), in which case the processor receives data from these devices, or output devices (printer, monitor) where the processor sends data to these devices. Furthermore, it is possible to have devices that at different points in time operate as input or output devices. Think of a printer/scanner device. When used for printing, this device operates as an output device. However, when used for scanning, the same device operates as an input device. Such devices are called I/O (input/output) devices. To communicate with other devices, the processor needs to address the different devices that are connected to it. This is similar to how the processor addresses different memory locations in the memory. However, when it comes to peripheral devices, there are two main methods on how the processor communicates with these devices, i.e. memory-mapped I/O and isolated I/O. In the case of memory-mapped I/O, the same address space is used to address the memory and the peripheral devices. In such case, when the CPU accesses an address, it may refer to a portion of physical RAM, but it can also refer to memory (registers) of an I/O device. Thus, the entire memory space is shared between memory and peripheral devices. The advantage of memory-mapped I/O is that the processor can use the same instructions when accessing memory or I/O devices. In case of memory-mapped I/O the different registers (control, status and data) of the device are mapped to different memory addresses. Given that these addresses are known in advance, by using pointers, we can write software such that we can configure and send/receive data, and check the status of these devices.

Assignment 8.

In this assignment, we show how we can use the 16 switches and the 16 LEDs on the Nexys 4 board. Both the switches and the LEDs are interfacing the microprocessor through memory-mapped I/O devices. The *SWITCHES* device is a configurable I/O device that interfaces the 16 switches on the Nexys 4 board to the MicroBlaze microprocessor and the device *LEDS* is a configurable I/O device that interfaces the 16 LEDs on the Nexys 4 board to the MicroBlaze microprocessor. Each device has one 32-bit control and one 32-bit data register. The control register is used to control the data flow direction for each bit of the data register, i.e. each bit of the data register can be configured either as an input or output. The *SWITCHES* device has to be configured as an input device, such that each bit in the data register of this device is configured as an input, i.e. the state of each switch on the board is input information that the processor can read. On the

other hand, the *LEDS* device has to be configured as an output device, such that each bit of the data register of this device is configured as output, i.e. the processor will control the state of the LEDs on the board. The devices are configured by accessing and writing to their corresponding control registers. Since each of the I/O devices has a control and a data register that are already mapped to known memory addresses, for the ease of use, we create a header file where we define labels (macros) to refer to these particular registers (memory addresses).

To create a new header file, in the Project Explorer locate the “src” folder, right click, and from the popup menu select “New->Header File”. Use “address_mapping.h” as a name for the new header file that you are about to create.

In the newly created file “address_mapping.h”, copy and insert the following lines between the “#define” and “#endif” directives:

```
#define SWITCHES_DATA      (unsigned int *) 0x40020000
#define SWITCHES_CONTROL  (unsigned int *) 0x40020004
#define LED_DATA           (unsigned int *) 0x40010000
#define LED_CONTROL        (unsigned int *) 0x40010004
```

The header file contains definitions for different pointers. For example, to access the control register of the *LEDS* device, we can use the pointer “LED_CONTROL” which points to a memory address “0x40010004” and accesses a 32 bit data (the size of the control register is 32 bits). Each of these 32 bits configures the corresponding bit of the data register of the *LEDS* device as either input or output. When a particular bit in the control register is set to “0”, the corresponding bit in the data register is configured as output, i.e. the data flows from the processor to the device. When a particular bit in the control register is set to “1”, the corresponding bit in the data register is configured as input, i.e. the data flows from the device to the processor. Since the LEDs are used to output information, we need to ensure that the control register for the *LEDS* device, i.e. “LED_CONTROL”, is configured such that each of its bits is set to “0”. Remember that “LED_CONTROL” is a pointer that points to the known memory address “0x40010004”. To write to this register, we need to *dereference* the pointer, i.e. we use “*LED_CONTROL”.

The following code shows a simple example on how we can use the definitions from the header file to enable access to the memory-mapped devices.

```

#include "address_mapping.h"

unsigned int state;
unsigned int counter;
int main(){
    *SWITCHES_CONTROL=0xFFFF;
    *LED_CONTROL=0x0;
    counter=10;
    while (1==1){
        while (counter>=0){
            state=*SWITCHES_DATA;
            *LED_DATA=counter;
            counter--;
        }
    }
}

```

Debug the program by stepping through the code. After each step, modify the state of the switches. Follow the changes for each of the variables, and the memory mapped registers of the two I/O devices. Answer the following questions:

At which memory address is the variable “counter” stored? _____

At which memory address is the variable “state” stored? _____

Does the variable “state”, at every point in time, have the same value as the data register of the switches? Explain the reasoning behind your answer.

Assignment 9.

Study the code below. Do you expect any difference? _____

```

#include "address_mapping.h"

unsigned int state;
int main(){
    *SWITCHES_CONTROL=0xFFFF;
    *LED_CONTROL=0x0;
    *LED_DATA =10;
    while (1==1){
        while (*LED_DATA>=0){
            state=*SWITCHES_DATA;
            *LED_DATA=*LED_DATA-1;
        }
    }
}

```

Comment the contents of the “main.c” file, and add the provided source code. Build the project and debug the program by stepping through the code.

What do you observe on the board? _____

Open the “Expression” view in the debug environment, and check the contents of the expression “*LED_DATA”. How is the expression evaluated? _____

What happens when the program reads a write-only memory location? _____

Assignment 10.

Write a sample code that writes to the data register of the SWITCHES device. Check in the memory window to see the contents of the data register (“*SWITCHES_DATA”).

Do you observe any changes? _____

What happens when the program writes to a read-only memory location? _____

Assignment 11.

Write a program that based on the state of the switches, turns on/off the corresponding LED. For example, if the right-most switch is on, the right-most LED is turned on.

Assignment 12.

Write a program that based on the state of the switches, will turn on a number of LEDs that corresponds to the number of switches that are turned on. For example, if three switches are turned on, then only the rightmost three LEDs should be turned on. **Hint:** the decimal number $2^n - 1$ has a binary representation that contains n ‘1’s. Note that you are not allowed to use C library functions to implement the function that computes power of a number.