C Programming on MSL Nios II System Tutorial

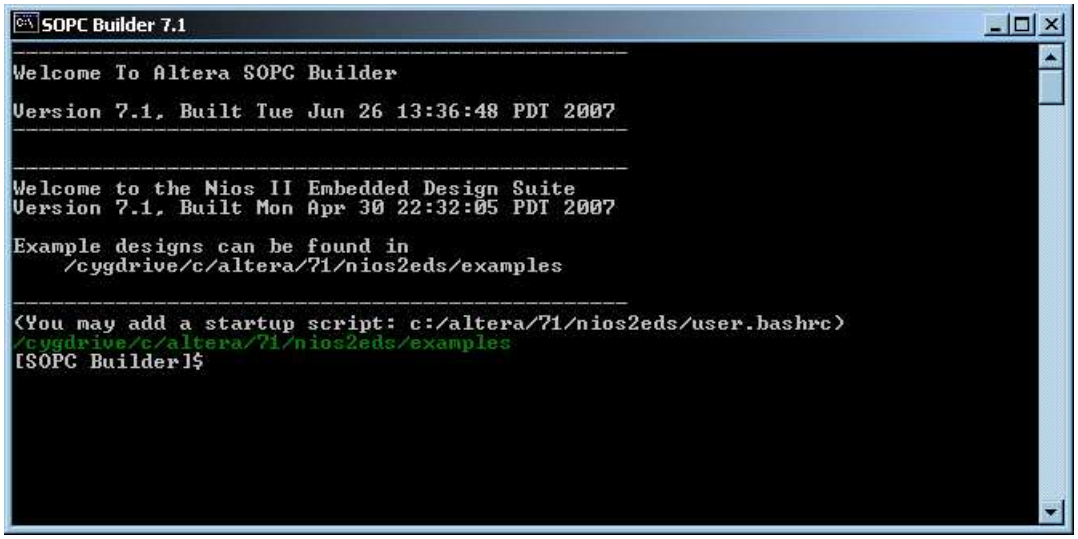
Frank Franjo Plavec

September 2007

University of Toronto

Running Your first Program on Nios II

1. Click on Start->Programs->Altera->Nios II EDS 7.1->Nios II 7.1 Command Shell
A new command line window will open. This is Cygwin window, a Unix-like interface for windows. You can use many common Unix commands in this window. We will use it to compile and run the programs.



2. Type `cd w:`
This takes you to the W drive, which is where your UGSPARC account is mounted, and where you have personal space of approximately 300MB.

3. Create a folder named TUTORIAL, or any other name you choose. The rest of the tutorial assumes the folder name is TUTORIAL, so if you use a different name, keep that in mind. The command to create the folder is `mkdir TUTORIAL`
Enter the newly created folder: `cd TUTORIAL`

4. Open Internet Explorer. The MSL web-site should pop-up immediately (If it doesn't go to the following address: http://www-ug.eecg.toronto.edu/msl/)
Click on "Nios II" on the left side, and then on "Running" (also on the left side). There is a link to Makefile on that page. Right-click on the link and select "Save Target As". Locate W: drive and save the file in the folder you created in the previous step (TUTORIAL).
This Makefile will be used to compile and run all the programs and to start the debugger.

5. Turn on the DE2 board (two switches: power supply and the red button on the board)
Go back to the command prompt and type `make test`
If you get an error message like this:
`make: *** No rule to make target `test'. Stop.`
you probably saved the Makefile as text file. Go back to the Internet Explorer and select "Save Target As" again. When the "Save As" dialog box pops up, go to the folder where you want to save it, and enclose Makefile into quotations like this `"Makefile"` (keep quote marks). This assumes that you deleted .txt part from the name.
Go back to the command prompt and type `make test`
After short time the test program should be downloaded to the board, and the 7-segment displays should start counting and LEDs flashing.

6. Type
```
make help
```
into the Command line window. Examine the various options offered.
Don't worry if you don't understand them all yet. We will cover all you need.

Compiling Your First Program

The program you downloaded in the previous exercise was already pre-compiled for you. If you check the command line window, you will see that a file test.elf was downloaded to the board. ELF stands for "Executable and Linkable Format", which is a common standard file format used for executables. In this exercise you will compile your first program and download it to the board. You will now learn how to compile and run a C program starting from its source.

1. Go to the MSL web-site. Click on Nios II and then Devices on the left side. Click on "Slider Switches". This page describes how to use the slider switches (surprise, surprise :)) on the DE2 board. The table contains various information, including the base address where the register corresponding to the switches resides. Default Nios II system maps each device on the board to a register in the Nios II address space. This means that reading from this memory address will read the state of the switches. All you need in C is a pointer that points to that memory location.

2. Below the table there are two pieces of code, one written in assembly, the other one in C. For this tutorial, always use the C code sample. Select all text below the “C Example” title, right click and select Copy. Open Notepad and paste the copied text. Save the file in the TUTORIAL folder and call it `"switches.c"` Once again, be careful to enclose the name in quotes, so that it does NOT get saved with .txt extension.

3. Go back to the command-line window and type
```
make SRCS="switches.c" run
```
If you get an error message like this:
```
make: *** No rule to make target `switches.o', needed by `prog.elf'.
Stop.
```
you probably didn't save the file `switches.c` appropriately (might have a `.txt` extension).

If you did everything right, the program will be compiled and downloaded onto the board. The state of the red LEDs will correspond to the state of the input switches. Try flipping a few switches to see what happens.
Does LED4 behave as expected? Can you explain its behaviour based on the source code you typed into Notepad?

Basic Debugging

The previous exercise taught you how to compile and run a program. In the following exercise we will show how to debug programs.

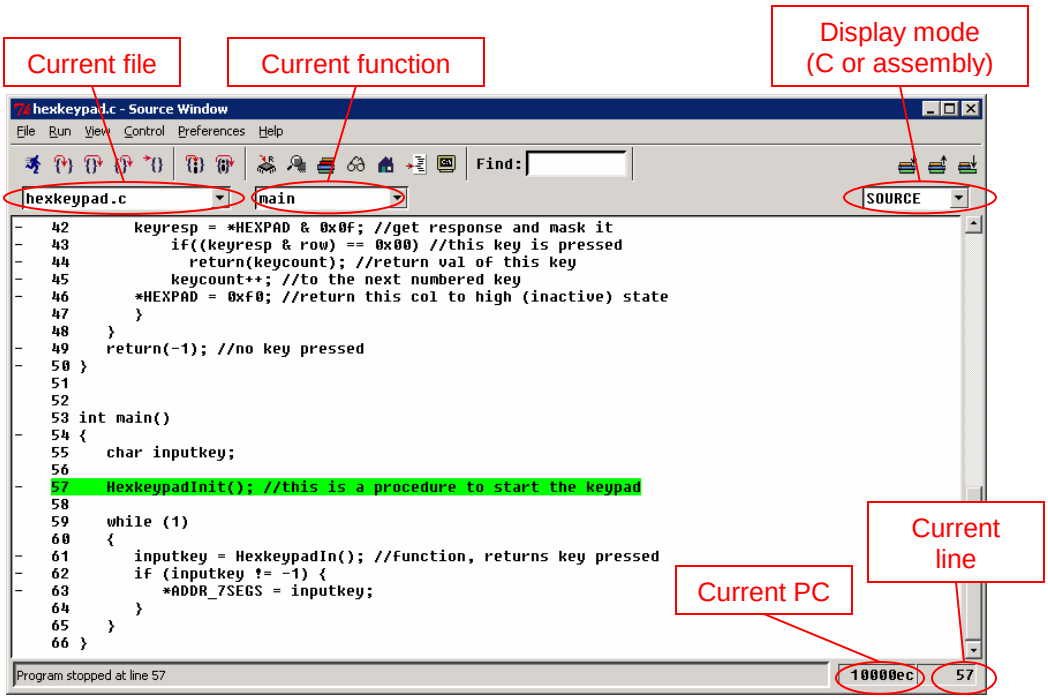
1. At the MSL web-site locate the documentation for Hexkeypad. Copy the C example code into Notepad and save the file into TUTORIAL folder with file name `hexkeypad.c`.

2. Before using this file, you will have to attach the hex keypad to the DE2 board. The code you will be using requires the keypad to be connected to the JP1 expansion header (the left one) on the DE2 board. To attach the keyboard, you will require a 40-pin ribbon cable from the lab kit. The cable can be plugged into the keypad and into the board only one way, because of a notch on one side of the header.

BE CAREFUL to connect the keypad to the correct header, and to plug the cable into the headers the right way.

3. In the command-line window type
   `make SRCS="hexkeypad.c" debug`

The program will be compiled and debugger window will pop-up. The debugger window that opens is a GUI (Graphical User Interface) for the standard GDB debugger. The user interface is called Insight Debugger, and should look like this:



The debugger starts with the Source Window open and pauses the execution at the entry to the main() function in your program. Pay attention to the three drop-down boxes on top of the window.

The first box from the left is used to select one of the source files for the currently running program. In this case, there are only two files: `hexkeypad.c` and `crt0.s`. The latter is a start-up routine, which is a part of any C program. It is needed to set up the environment in which a C program can run (setting up stack and global pointers, etc.)

The second box is used to select one of the functions in the currently selected file. When `hexkeypad.c` is selected in the first box, you can select `HexkeypadIn`, `HexkeypadInit`, or `main` function in this box, which are all the functions in the current file. Clicking on one of the functions takes you to the source code of that function.

Finally, the third box allows you to select the display mode for the currently displayed file. You should only use the SOURCE display mode, which displays the source code for the program being executed. Other modes allow you to display assembly of the generated machine instructions for the program, or mix of source code and assembly instructions. If you don't know what these are, do not change this setting.

Toolbar

The toolbar consists of three functional sections: execution control buttons, debugger window buttons, and stack frame control buttons.

Execution Control Buttons

These convenience buttons provide on-screen access to the most important debugger execution control functions. Some of the important icons you will be using are listed below. For other functions, please consult Insight Debugger's help system.

Step
: Step the program until it reaches a different source line

Next
: Step the program, proceeding through subroutine calls

Finish
: Execute until the current stack frame (function) returns

Continue
: Continue the program being debugged, after signal or breakpoint

Stop
: The Stop Button will interrupt execution of the program. It is also used as an indication that the debugger is busy.

Run
: The Run Button is NOT USED for Nios II debugging. If you wish to restart the execution, quit the debugger (File→Exit, and then answer Yes in the dialogue box that pops-up), and run it again.

Window Buttons

The Debugger Window buttons give instant access to the Debugger's auxiliary windows:

Local Variables
: Open a Locals Window, which allows you to see the values of local variables

Memory
: Open a Memory Window, which allows you to inspect content of system memory

Stack
: Open a Stack Window, which displays the list of functions on the stack

Watch Expressions
: Open a Watch Window, which allows you to monitor custom expressions

Breakpoints
: Open a Breakpoint Window that lists all the breakpoints

Console
: Open a GDB Console Window (This is NOT a terminal window)

4. Try to step through the program using the  Step and  Next commands until you notice the difference between the two. Notice that you can do this without using the mouse, by using keyboard shortcuts **S** and **N**.

IMPORTANT NOTE!!!
The debugger sometimes appears frozen: i.e. all of the buttons are greyed, and you cannot press any of them. The resolution of this problem depends on the situation

1. You have just started the debugger and it appears frozen
Click on the area with source code (white part of the window) and press S on the keyboard. The debugger will single-step through your code and the user-interface should become responsive.

2. The debugger is unresponsive after you pressed Finish or Continue
Your program is probably stuck in a loop that takes a long time (possibly an infinite loop, whether intentional or not). Press the Stop button to interrupt the program execution.

5. In this step, we will set-up a breakpoint. Breakpoints allow you to run the program until a certain point in the program has been reached. In our program, the hex keypad is scanned until a key press is detected. At that point `HexkeypadIn()` function will return a number that is not `-1`. We will run our program until this happens. To do this, locate the line of code inside `main()` that looks like this
```
    *ADDR_7SEGS = inputkey;
```
Note that there is a "-" character at the beginning of that line. This indicates that this line is "executable". Breakpoints can be set only at executable lines of code.
Position your mouse over the dash at the beginning of that line. The mouse pointer will turn into a circle. Left-clicking will cause a red dot to appear in that spot, which means a breakpoint has been placed there.

6. Now that you have set the breakpoint, click on the  Continue button. The program will continue execution until it reaches the breakpoint. This should not happen until you press a button on the hex keypad. Of course, the keypad should be attached to the board's JP1 extension header, as previously specified

7. Press one of the keys on the keypad. Program execution should now stop at the breakpoint. If this does not happen, double check the connection between the keypad and the DE2 board.

8. Step through the program once using the  Step to see the key you pressed appear on the 7-segment displays.

9. A very useful tool in debugging is to examine the content of local variables. If you click on the  Local Variables button, you should normally get a listing of all local variables. However, in this case the window is empty, because the compiler performed some optimizations and debugger cannot find the `inputkey` variable. One way to avoid this situation is to modify the Makefile to turn off the optimizations

10. Open Windows Explorer and go to the `W:\TUTORIAL` folder. Double-click on the `Makefile`. When asked which program to use to open the file, choose `WordPad` at the end of the list. Once the file is open, locate the following text:

```
%.o: %.c
   nios2-elf-gcc -g -mno-cache-volatile -mno-hw-mulx -mhw-mul -mhw-
   div –O1 -ffunction-sections -fdata-sections -fverbose-asm -fno-
   inline $(UPINCLUDES) -
   I$(NIOS2EDSPATH)/components/altera_nios2/HAL/inc -
   DSYSTEM_BUS_WIDTH=32 -DALT_SINGLE_THREADED -
   D_JTAG_UART_BASE=0x00ff10f0 -c $?
```

    Change the `-O1` into `-O0`. That is, change "minus, capital letter O, one" into "minus, capital letter O, zero". This will change the optimization level from 1 to 0 (i.e no optimizations).

11. Select File→Save. You may get a dialogue box asking you if you really want to save the file in a Text-Only format. Answer Yes.
    If the debugger is still running exit it. Go to the Command line window and type
    `make clean`
    This operation will eliminate all object, elf and programming files generated by previous compiles. This is necessary to force re-compilation of the code with new optimization options[1].

> **IMPORTANT NOTE!!!**
> In general, it is a good idea to do `make clean` if your code behaves "weird", just to make sure that something didn't go wrong during compilation. Recompiling the code from scratch sometimes (though not often) removes the weird behaviour.

12. Type `make SRCS="hexkeypad.c" debug`. Repeat steps 5 through 7 above. Open the Local Variables window and observe the value of inputkey variable. Right-click on the value of inputkey in the Local Variables and explore the various options that can be set there.

> **IMPORTANT NOTE!!!**
> In case you run into unexpected problems after modifying the Makefile, you might have inadvertently modified a section of the Makefile you shouldn't have. Delete the Makefile, and download it again using the procedure described in steps 4 and 5 on page 1 of this tutorial.

Using terminal

In this exercise, we will add some `printf()` statements to the `hexkeypad.c` file created above. You will also learn how to compile the program to support the terminal output, and open a terminal window where you can see the output. printf's can be useful for printing program status to the terminal.

1. Go to the Makefile that was opened in WordPad. In case you closed it, open it again like in step 10 above. Change the optimizations back to `-O1` (see step 10 above). You will probably be able to use the WordPad undo function for this. Select File→Save.

2. If not already opened, open the `hexkeypad.c` using Notepad. Add the following directive to the top of the file:

```
#include <stdio.h>
```

[1] You may be aware that make utility checks the "last modified" dates of all its input files to decide whether to regenerate the targets. However, the make utility does not check the "last modified" date of the Makefile, so in this case it would not regenerate the targets without first running `make clean`.

3. Inside the `main()` function add a `printf` statement that prints the key pressed on the hexkey to the terminal. You can use `%X` printf format to avoid having to convert raw numbers into their hex representation. Where should you place the printf statement so that it only gets printed when a key is pressed?

4. Open another command line window (so you should have two of them open now). In the newly opened window go to the w: drive and TUTORIAL folder and type the following: `make terminal`. This command-line window will now serve as an output for printf's in your program, and you may also use it as an input (reading keyboard) if your application requires it.

5. Go back to the other command line window and type the following
   `make SRCS="hexkeypad.c" JTAGOBJ=jtag.o run`
   The `JTAGOBJ=jtag.o` part specifies that standard I/O should be redirected through JTAG UART, which is a serial link that utilizes USB port to support terminal services. In case you forget the exact syntax of this (or any other) make command, type
   `make help`

6. Once the program is compiled and loaded, the keys you press should be printed to the terminal window (the other command line). What happens when multiple keys are pressed on the hex keypad at the same time?

Debugging and Developing Your Own Program

In this exercise you will do some simple debugging and you will extend an existing program to add more functionality to it.
This program demonstrates usage of switches and LEDs on the digital protoboard (the green board below the power supply) in combination with the DE2 board. These switches and LEDs can be used in addition to the switches and LEDs available on the DE2 board. In addition, the protoboard can be used to test the behaviour of GPIO for other purposes.

1. Download the start-up kit (tutorial.tar.gz) from the same place where you obtained this tutorial.

2. Go to the command line window and type
   `tar xzf tutorial.tar.gz`
   This will extract several programs from the archive which will be used in this and following exercises. One of the programs is called `protoboard.c`.

3. Connect the JP1 port of the DE2 board to the digital protoboard using the 40-pin ribbon cable. Turn the power to the switchboard on (the power switch is on the breadboard – white board with many holes).

4. Try to compile the `protoboard.c` program using the following command:
   `make SRCS="protoboard.c" compile`
   The program should NOT compile because of an error. Open the source file and find the cause of the error. Fix the error, and compile the program again using the above command until the compilation succeeds.
   While looking at the program, you will notice a `for` loop terminated with a semicolon. The purpose of this loop is to create a primitive time delay.

5. Run the program. Please note that you cannot run previously compiled program by simply typing `make run`. You have to type in the full command:
   `make SRCS="protoboard.c" run`

6. Once the program starts running, you should see the “travelling light” effect on the LEDs. Open the source code for the program and analyze it to understand how the effect is achieved.

7. **BONUS 1:** Add the following functionality to the program: Let the light travel in one direction when switch 1 on the protoboard is **on**, and in the opposite direction when that switch is **off**. If you don’t understand all the details of how the protoboard communicates with the DE2 board, check the protoboard documentation on the MSL web-site (Under Devices).
Once your program is working, modify the program so that it uses JP2 expansion header on the DE2 board

8. **BONUS 2**: The delay in the current program is implemented using a for loop. This creates an imprecise delay and could be optimized away by a smart compiler. Study the documentation for the Timer available in the default Nios II system. Change the program to use the timer in such a way that the delay between the steps is precisely set to 0.5 seconds.

Using LEGO kits

In this exercise you will learn how to use C programs to control the LEGO kit. The principle is the same as in the previous exercise, just the device is different. This is only a short introduction to using the LEGO kit. For detailed documentation, refer to the MSL web-site (Nios II→Devices→Lego Controller)

1. Attach the LEGO Controller to the JP1 port of DE2 board. Attach one of the motors to the Motor1 connector on the Lego Controller. Connect the power to the LEGO Controller.

2. Compile and run the program `lego.c` that you extracted in the previous exercise. Once the program runs, the motor should turn on. If the motor doesn’t turn on, check the motor override switch and all the connections.

3. Change the code to turn the motor off. This can be done by writing `0x00` to the memory location pointed to by the `LegoMotors` pointer (instead of `0x01`)

4. Change the code so that the motor is on for a period of time and then it turns off for a period of time, and repeat this periodically. By fine-tuning the times the motor is on and off, you may be able to control the speed of the motor. That is, if the motor is turned on and off periodically, it will move the LEGO piece it is controlling (e.g. a gear) less than if the motor was on all the time.

5. Connect two sensors to Sensor1 and Sensor2.

6. Open the file `lego_adv.c` in Notepad. Analyze the code and try to understand what the code will do. For this code to work, you may have to calibrate the sensors using potentiometers. For details, see Lego documentation on the MSL web-site.

7. Compile and run the program lego_adv.c. Shine light on the sensors. Does the code behave how you expected? If not, analyze the code again and try to understand its behaviour.

List of Potential Problems

In your time during the lab you will likely encounter countless problems. In this section, you will find descriptions of some common mistakes and problems we are aware of. Hopefully, this knowledge will help you solve new problems when they occur.

Wrong Command-Prompt

If you open the regular Windows Command Prompt instead of the Cygwin command prompt, the Makefile will not work or will behave weirdly.

Disk Space

You have limited disk space on the W: drive. Since you probably use the same account for multiple courses, you may run out of disk space. The make utility will usually fail with an error message that will not reveal the problem. Common error messages are “Cannot write file <filename>.<ext>”. Check the amount of free space on the disk (In Windows Explorer right-click on the W: drive and select Properties). If disk space is the problem, free up some space on the drive by deleting unnecessary files. `make clean` may also help, but only temporarily. It will remove all files generated previously, but next make will create more files.

UNIX files on Windows

If you open your source file in Notepad and find that it looks all garbled (e.g. all squeezed into a few lines), try opening it with another program, such as WordPad.

Conflicts between Assembler and C

If you use assembly, you should be aware of the following issue. If there is a c program (.c extension) and an assembler program (.s extension), with the same base name in the same folder, assembler code will get compiled even when the C source code is specified.

For instance, if a folder contains files `program.s` and `program.c`, both of the following commands will compile and run the assembler program:

```
make SRCS=″program.s″ run
make SRCS=″program.c″ run
```

WORKAROUND: Change one of the files’ names.

Frozen Debugger

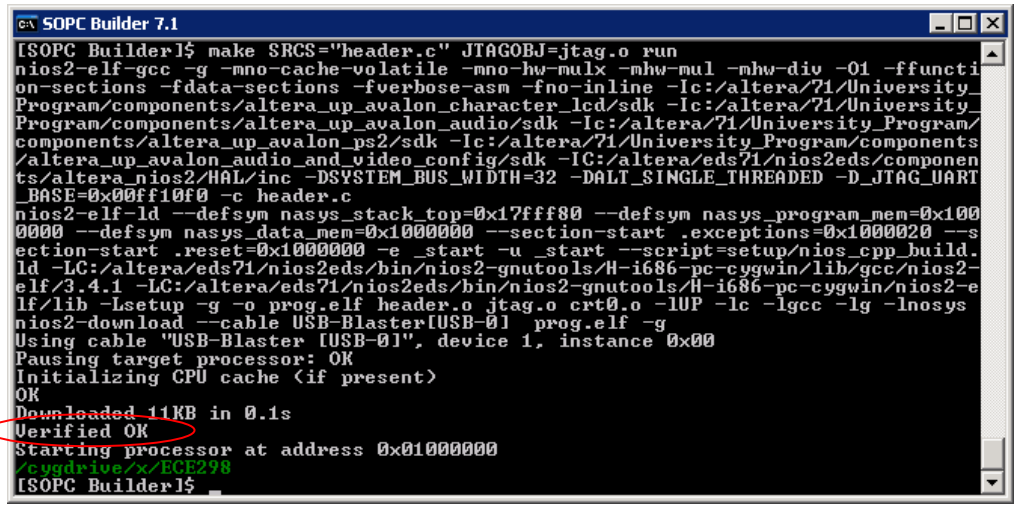
The debugger sometimes appears frozen: i.e. all of the buttons are greyed, and you cannot press any of them. The resolution of this problem depends on the situation

1. You have just started the debugger and it appears frozen
Click on the area with source code (white part of the window) and press S on the keyboard. The debugger will single-step through your code and the user-interface should become responsive.

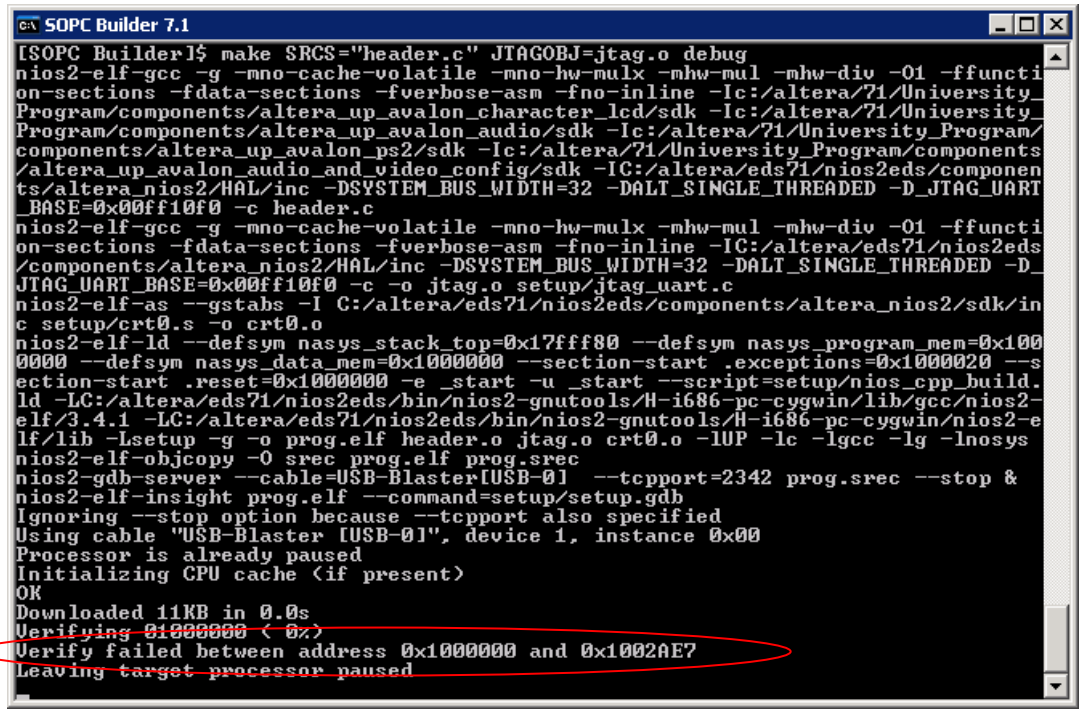
2. The debugger is unresponsive after you pressed Finish or Continue
Your program is probably stuck in a loop that takes a long time (possibly an infinite loop, whether intentional or not). Press the Stop button to interrupt the program execution.

Configuration/Hardware Problem

If the debugger is not responding or the program is not running, check if the program was downloaded to the board successfully. To do this, look into the command line window where you ran your `make` commands and check for error messages. If everything was OK, the window should look like this:



If there are problems, there might be an error message that is hard to spot, like in the following figure:



In the above figure the debugger was started without any problems (not shown), but debugging was impossible. By careful inspection of the output of the make utility, we observe the line: `Verify failed between address 0x1000000 and 0x1002AE7.` This means that the verification process failed.

The verification process tries to read the program back from the board's memory after it has been downloaded to ensure that the download was successful. If the program read is not equivalent to the original program that was downloaded, an error is reported. This is usually an indicator of a hardware problem. Follow this procedure to resolve it:

1. Power the DE2 board off. Wait for three seconds. Power it back on. This ensures that the default Nios II system has been loaded into the FPGA. Try downloading the program again.
2. If verification still fails after doing step 1, close all the programs and restart the PC. Try downloading the program again.
3. If verification still fails, type `make configure` (make sure you are in the folder with the Makefile). This should reprogram the FPGA chip with the default design. If this helps, the default configuration on the board has been corrupted. Inform your TA about this, so that they can fix it permanently.

Miscellaneous

- printf's always go to the JTAG UART. However, there are actually two UARTS on the board. If you want o use RS232 UART (which connects to the COM1 serial port on the computer), you will have to open a different terminal window. Check the instructions on the MSL web-site (under Nios II→Devices→ RS-232 UART) for instructions on how to do that.

- Other than turning the board off and back on, the processor can be reset (master reset signal) by pressing all 4 push buttons (blue buttons in the bottom right corner of the board) simultaneously. This doesn't always work reliably, so try until you succeed. It is IMPORTANT to be aware of this in case you build your design in such a way that pressing all 4 buttons means something for your program.

- There is no Winzip or a similar utility installed on the Windows system in the lab. You can still unzip their files by logging into one of the UGSPARC machines, by following these steps:

    1. Store your zip file somewhere on the `W:` drive
    2. On the Windows desktop double-click on TeraTermSSH
    3. A window will pop-up asking you which machine you wish to connect to. Keep the default machine name. (You can select another valid name if you know one, of course)
    4. Log-in using your ECF login and password
    5. Once logged-in, you have access to the same files that are on the `W:` drive (`W:` is a network mapped drive). Therefore, go to the folder where you saved your zip file and assuming it is called `file.zip`, type
       `unzip file.zip`
    6. You can now access the unzipped files either through Windows Explorer or SSH. If the files do not appear immediately in your Windows Explorer window, press F5 (Refresh)