ERIKA Enterprise LWIP Tutorial

for the Altera Nios II platform

About Evidence S.r.l.

Evidence is a spin-off company of the ReTiS Lab of the Scuola Superiore S. Anna, Pisa, Italy. We are experts in the domain of embedded and real-time systems with a deep knowledge of the design and specification of embedded SW. We keep providing significant advances in the state of the art of real-time analysis and multiprocessor scheduling. Our methodologies and tools aim at bringing innovative solutions for next-generation embedded systems architectures and designs, such as multiprocessor-on-a-chip, reconfigurable hardware, dynamic scheduling and much more!

Contact Info

Address:
Evidence Srl,
Via Carducci 56
Località Ghezzano
56010 S.Giuliano Terme
Pisa - Italy
Tel: +39 050 991 1122, +39 050 991 1224
Fax: +39 050 991 0812, +39 050 991 0855

For more information on Evidence Products, please send an e-mail to the following address: info@evidence.eu.com. Other informations about the Evidence product line can be found at the Evidence web site at: http://www.evidence.eu.com.

Contents

1 Introduction

This tutorial explains in detail how to use the Lightweight IP (from now on, LWIP) TCP/IP software stack [1] together with RT-Druid and Erika Enterprise for the Altera Nios II platform.

The tutorial is based on the Nios II Standalone LWIP Port available for download in the Nios Community Forum, and it covers LWIP version 0.7.1. Support for newer versions of LWIP will be added in the next revision of this document.

At the end of the tutorial, the developer will be able to run a modified version of the LWIP web server demo on top of an Altera evaluation board.

The demo fully supports the LWIP RAW API. The rationale behind the choice is that the RAW API is an event based interface to the TCP/IP stack that perfectly integrates with the non-blocking task model provided by Erika Enterprise. In particular, the LWIP timers and service routines are mapped to a set of tasks; LWIP and Application tasks can share a common stack reducing the overall RAM usage.

The socket interface is currently not supported because of bigger memory requirements, and because of the need of blocking primitives (e.g., `select`) that would not take advantage of the stack sharing mechanisms of Erika Enterprise.

If compared to the single task LWIP standalone version for the Altera HAL[1], this porting of the LWIP stack fully supports the multithread environment of Erika Enterprise, meaning that application task can run together with LWIP tasks.

For more information about the LWIP example and on the LWIP internals, please refer to the original `readme.txt` included in the distribution.

The structure of this tutorial is the following: Chapter 2 contains the requirements for installing the LWIP port for Erika Enterprise; Chapter 3 is a step-by-step guide for the installation, configuration and run of the demo; finally, Chapter 4 contains advanced configuration issues.

---

[1]The LWIP standalone version for the Altera HAL is available for free in the Nios II Community Forum Download area.

2 Requirements

The LWIP demo presented in this tutorial requires the following software to be installed on the host machine:

- Altera Quartus II version 5.0;

- Altera Nios II version 5.0;

- RT-Druid and Erika Enterprise for Nios II version 1.2.5. The evaluation version of the tools will *not* work straightforward because they require a 2-CPU demo, whereas this tutorial is based on the `standard` Altera examples.

- Standalone Lightweight IP version 1.1 (available on the Nios II Community Forum).

- This example requires an Ethernet cable connected to the development board's RJ-45 jack, and a JTAG connection with the development board. If the host communication settings are changed from JTAG UART (default) to use a conventional UART, a serial cable between board DB-9 connector and the host is required.

The demo will work on the standard and full featured demos available for the Altera Evaluation boards.

3 Running the LWIP Demo

The following basic steps will guide you in running a small web server using the LWIP TCP/IP stack:

1. Open the `standard` example for your evaluation board from Altera Quartus II.

2. Open SOPCBuilder, by double clicking on the SOPCBuilder Block. Then, open the Nios II IDE.

3. Create a new Altera System Library Project. Call it `standard_syslib`. See Figure 3.1 for a screenshot.

4. Then, select "New Project..." from the File menu. Choose "RT-Druid Nios Project" from the Evidence tab of the New Project Dialog box. A dialog box appears. Name the project `evidence_lwip` and press the Finish button.

5. Select "Import..." from the File menu of the Nios II IDE. Select "File System" from the dialog box. In the "From directory" textbox, type the name of the `src` directory included the example you downloaded from the Evidence website (you can also select it using the "Browse" button). Choose the directory name in the tree view on the left, selecting all the files on the right side. The "Into Folder" text box should point to the `evidence_lwip` demo. Finally, select the checkbox "Overwrite existing resources without warning" and press Finish. See Figure **??** for a screenshot of the Import window.

6. Among the files that have been imported in the RT-Druid Project, find the file named `altera_avalon_lan91c111.c`. You have to copy it into the System Library Project `standard_syslib`. To copy it, right click on the file name, and choose "Copy"; then, right click on the project name, and select "Paste". The file you just copied will replace the file provided by default by the Standalone LWIP package. The new file is equal to the previous one apart for a callback that has been added to the IRQ handler function.

7. Right click on the System Library name, and choose "Build Project" to build the System Library.

8. Set up the RT-Druid project build directory. To do that, open the properties of the project, and inside the "C/C++ Make Project" tab, in the Build directory textbox, specify `\<projectname>\Debug`, where `<projectname>` is the name of the project. See Figure 3.4 for details.
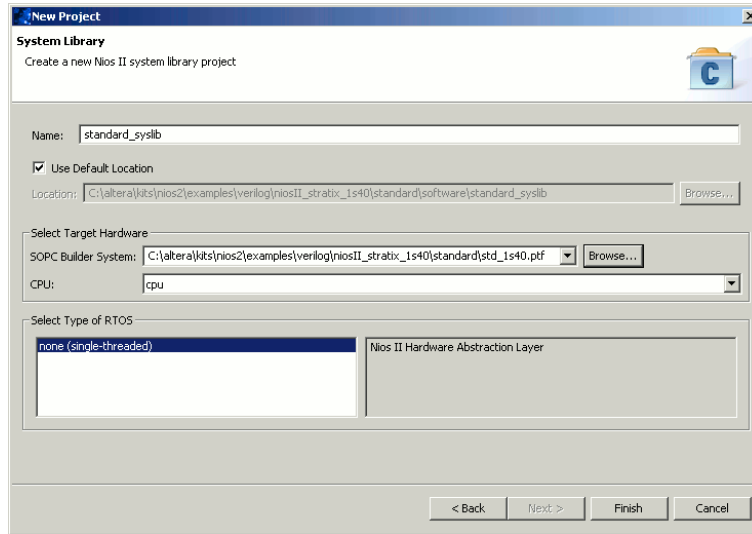
6

Figure 3.1: This screenshot shows the dialog box for the creation of the `standard_syslib` project.

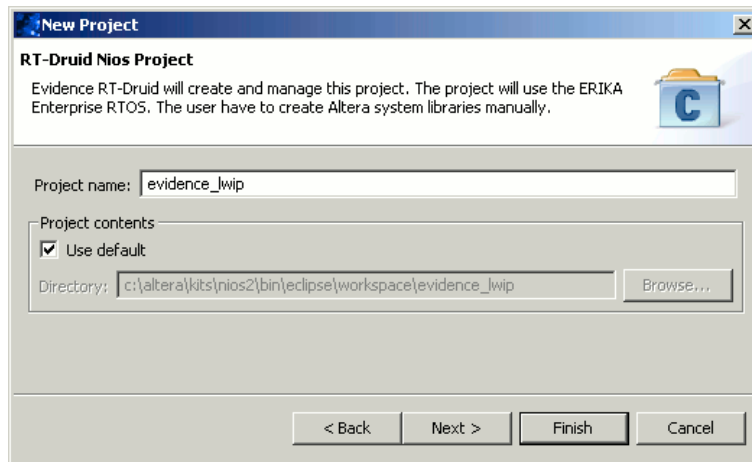Figure 3.2: This screenshot shows the dialog box for the creation of the RT-Druid Project.
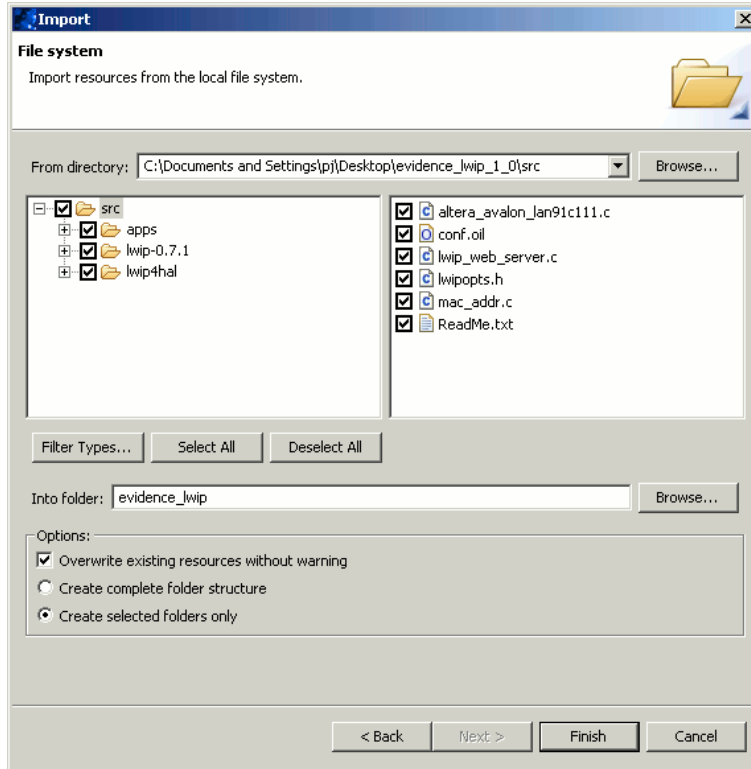
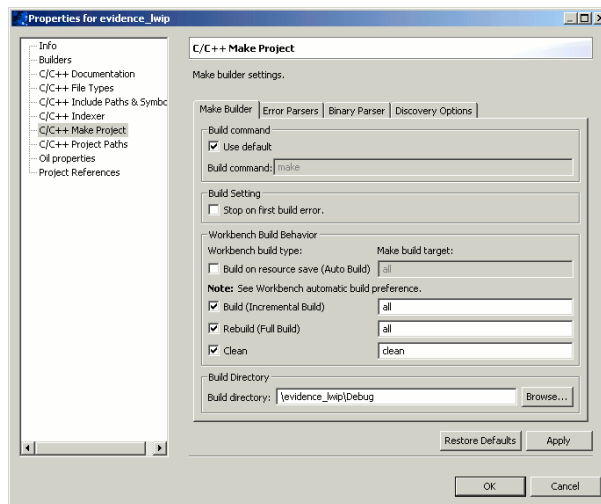Figure 3.3: This screenshot shows the dialog box for the import of the demo inside the current project.

Figure 3.4: Changing the Project Build Directory. You can insert the value pressing the Browse button.
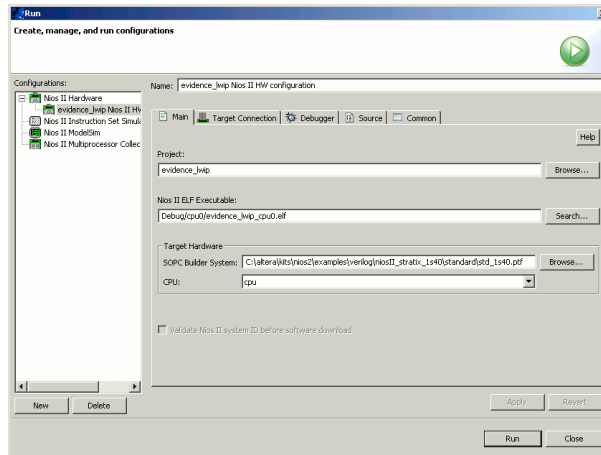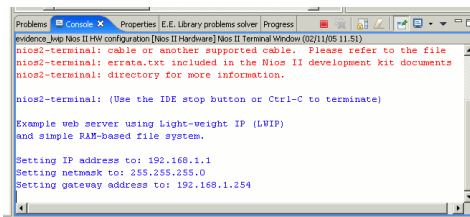
Figure 3.5: You need to create a Debug/Run configuration to be able to run the application on the target.

9. Find and edit the IP address, network mask, and gateway address inside the file `lwip_web_server.c`. You will use this address from your web browser to test the application.

10. After that, right click on the RT-Druid project name, and select "Build Project". The demo application will be compiled, and an ELF binary file will be produced.

11. Open the Altera Quartus II Programmer from the "Tools" menu. Select the right `standard.sof` file, and program it to the FPGA. The hardware design is now programmed to the FPGA.

12. In the "Run" menu, click on the "Run..." command.

13. Create a Run configuration as shown in Figure 3.5.

14. Run the application by clicking on the "Run" button in Figure 3.5. As a result, the application starts printing some text on the console like in Figure 3.6.

15. Obtaining the messages in Figure 3.6 means the web server is up and running. At that point, you can do the following actions:

   - Browse the server using this IP address in the address bar of your browser.
   - Use a Telnet client to connect to TCP port 7 to exercise the echo function.
   - Use Ping with a length greater than 1500 and less than 5500 bytes to exercise the IP reassembly and fragmentation capability.

Figure 3.6: The messages printed on the JTAG UART console.

4 Configuration of the RTOS parameters for the LWIP stack

4.1 Application structure

When using the LWIP TCP/IP stack together with **Erika Enterprise**, the application design have to be designed accordingly to a set of common sense rules that help handling the TCP/IP with the least overhead maintaining the possibility of having concurrent tasks running in the system. These rules are shortly described below (see Figure **??** for a graphical reference), and are used in the demo distributed together with this tutorial.

- The `main()` function is used for **Erika Enterprise** initialization purposes *only*. In general, that function covers the system startup and implements at its end the background processing. For that reason, it should not call any LWIP timer function after system initialization (please note that the standalone example distributed in the Nios II Community forum does all the processing inside the `main()`). The rationale for this choice is that the application designer should be able to specify the precedence the application tasks should have over the TCP/IP processing.

- The initialization phase is carried out by a task named `LWIP_startup_task`, that basically contains all the initialization routines of the LWIP stack. The task is automatically activated by the `StartOS()` primitive (the `AUTOSTART` property of the task is set to `TRUE` in the OIL file.

- There are other two tasks, named `LWIP_service_task` and `LWIP_timer_task`, that are responsible to the execution of the LWIP timer functions. The rationale behind this choice is that the RAW API exports an event-based API for the handling of TCP/IP. There are two kind of events in the LWIP code: periodic and asynchronous events, with corresponding periodic and asynchronous LWIP timer functions. Periodic events are handled every given amount of time to process TCP, ARP, and reassembly timeouts. Asynchronous events occur in correspondence of the process of incoming packets from the network interface.

- The `LWIP_service_task` task simply calls the `lan91c111if_service` that is used by the LWIP stack to process an incoming packet. In principle, the `LWIP_service_task` task is activated every interrupt from the Ethernet interface[1]. Please note that the

---

[1] That is the reason for the patch to `alt_avalon_lan91c111_irq()`.The actual implementation slightly differ because there is to consider a race condition between task activation and the service process of the packet.
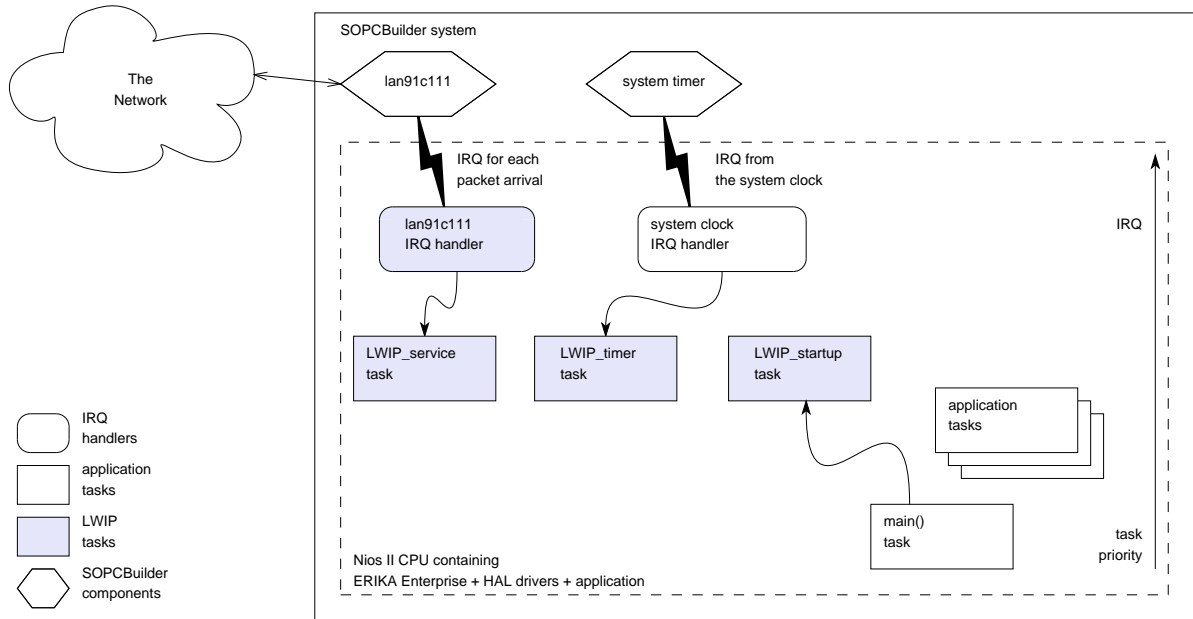
Figure 4.1: This Figure displays the architecture of the LWIP port, showing the timer and LAN interrupts, the LWIP tasks, and the main task.

standalone version called `lan91c111if_service` in the main loop, which typically resulted in a lot of computation time spent inside the call without any packet arrived, or, worse, since real application in the standalone version have to be implemented inside the main loop, having a lot of computation inside `main` simply decrease the polling frequency of the network, with a result of delays in the packet handling.

- The `LWIP_timer_task` task is activated periodically using an HAL alarm. The alarm mechanism of the Altera HAL is usually attached to the system timer. The content of he task is basically a call to all the periodic timeouts that handle the TCP/IP stack.

- The priorities of the three LWIP tasks are all the same, to avoid preemption between different LWIP tasks.

- Processing into the ISRs is reduced to the minimum possible, to avoid long interval of time executed with interrupt disabled: de facto, they contain only a call to `ActivateTask()`. Once activated, the LWIP tasks are scheduled like other application tasks, eventually sharing the stack with them to reduce the overall RAM consumption while maintaining multithread capabilities.

As a result, the application architecture depicted in the previous points allow multitasking support without loosing the efficiency of the RAW API, and enabling stack sharing between application tasks and LWIP tasks.

4.2 Handling concurrency between LWIP calls

Application tasks typically need to use the LWIP API only to create, bind, and listen on connections. All the other processing will be done on packet arrivals and on the periodic timers, that are handled inside the `LWIP_service_task` and `LWIP_timer_task` LWIP tasks.

The developer have also to take care of a race condition that appears when passing from the standalone version of the LWIP stack to a multitask system. In particular, some care have to be taken to avoid that the LWIP tasks `LWIP_service_task` and `LWIP_timer_task` preempts the running task during a LWIP call.

To avoid that situation, one out of the following four strategies can be implemented:

1. The LWIP tasks can be assigned the highest priority in the system. Then, a Resource is allocated, that is shared among the LWIP tasks and all the application tasks that needs to call the LWIP API. All the calls to the LWIP API done by application tasks have to be protected by the usage of the resource.

   As a result, the two LWIP tasks have the same priority, and so they are in mutual exclusion *without* the need of calling `GetResource()` and `ReleaseResource()`. Then, when a task uses a LWIP API function, the call of `GetResource()` before the usage of the API functions has the result of increasing the running task priority to the ceiling of the resource[2], making the task mutually exclusive with the LWIP tasks. If a packet arrives or a LWIP timeout expires while the running task executes the API function, then the LWIP task will be activated, and will then wait the release of the resource done by the running task with a call to `ReleaseResource()`.

   Please note that the demo distributed with this tutorial implements this scheme.

2. The LWIP tasks have not the highest priority in the system. In this case *all* the calls to the LWIP API, and the code of the LWIP tasks have to be protected using a shared resource.

3. All the task using the LWIP API can be defined in the OIL file as non preemptive. In this case, no explicit call to `GetResource()` and `ReleaseResource()` is needed. Note that other tasks with a priority higher than the LWIP tasks may suffer a blocking time also if they do not use the LWIP stack.

4. All the tasks using the LWIP API can be defined in the OIL file to share an internal resource. Also in this case, no explicit call to `GetResource()` and `ReleaseResource()` is needed. Other higher priority tasks can preempt the LWIP and the application tasks using the LWIP API.

---

[2]Erika Enterprise uses the Immediate Priority Ceiling Protocol, which says that the ceiling of a resource is the highest priority of the tasks potentially using the resource. Resource usages are taken from the OIL file.

5 History

Version	Comment
1.0.0	Initial revision.
1.0.1	Updated text, corrected typos, added history.
1.0.2	Added new versioning mechanism.

Bibliography

[1] Adam Dunkels. The lwip tcp/ip stack. http://www.sics.se/~adam/lwip/, 2005.