Technical notes on using Analog Devices DSPs, processors and development tools
Visit our Web resources http://www.analog.com/ee-notes and http://www.analog.com/processors or e-mail processor.support@analog.com or processor.tools.support@analog.com for technical support.

Running Two Network Interfaces with ADSP-BF537 Blackfin® Processors

Contributed by Jiang Wu

Rev 1 – December 12, 2006

Introduction

This EE-Note describes how to simultaneously run two network interfaces with ADSP-BF537 and ADSP-BF536 Blackfin® processors.

This description and all testing are based on a hardware setup consisting of an ADSP-BF537 EZ-Kit Lite® evaluation system and a Blackfin USB-LAN EZ-Extender card.

Hardware/Software Prerequisite

ADSP-BF537 processors are equipped with an Ethernet MAC controller that complies to IEEE 802.3[1]. The processors use a standard MII/RMII interface for easy connection to a wide range of network physical layer devices (PHYs). The ADSP-BF537 EZ-Kit Lite board includes an SMSC LAN85C183 Ethernet physical layer transceiver, which connects to the MII interface of the processor[2]. With this, the EZ-Kit Lite board readily provides one network interface.

The SMSC LAN91C111 single-chip Ethernet controller on the Blackfin USB-LAN EZ-Extender card provides another network interface[3]. It includes a MAC controller and a PHY. The extender is plugged onto the EZ-Kit Lite board via the expansion socket. The ADSP-BF537 processor controls the LAN91C111 Ethernet controller through its asynchronous memory interface. The overall configuration is illustrated in Figure 1.

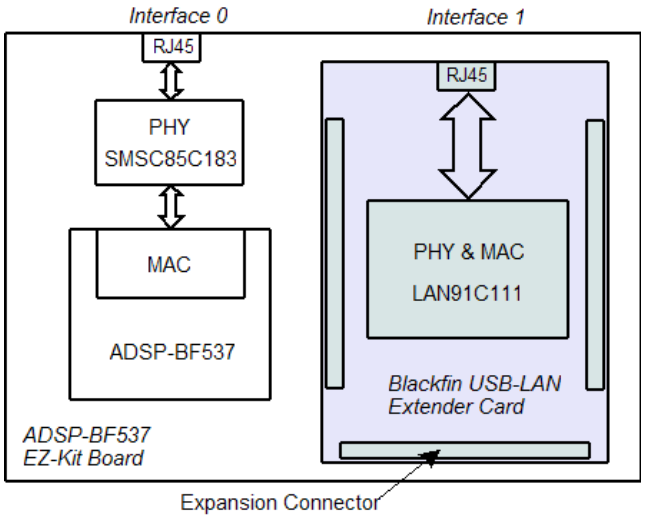

Figure 1. Hardware Configuration

One way to develop a network-capable application with Blackfin processors is to use the VisualDSP++® 4.5 development tools. This tools suite includes the drivers for the Ethernet controller on the ADSP-BF537 EZ-Kit Lite board and the USB-LAN extender card. The tools also provide a built-in TCP/IP application template, which is based on the VisualDSP++ 4.5 Kernel (VDK) and the open source TCP/IP light-weight IP (LwIP) stack. By means of the development tool's project wizard, the framework of a network application, including all the necessary C/C++ source code, can be built with a few mouse clicks. However, this framework is created for one network interface only: the EZ-Kit Lite board or the extender card.

The following section discusses how to change the single interface code to make it work with a second interface.

Ensure that both network devices can work separately by following the instructions and examples in the hardware user's manuals[2][3] and the *Getting Started with ADSP-BF537 EZ-KIT Lite*[4] manual.

Single Network Interface

VisualDSP++ 4.5 can be used to generate the startup source code for a TCP/IP application with one network interface support through its project wizard tool by specifying the project type as "`TCP/IP Stack application using lwIP and VDK`". The wizard walks you through selecting the processor type, specifying the hardware target (network devices, either from the ADSP-BF537 EZ-Kit Lite or from the USB-LAN extender card), the memory settings, processor clock/power settings, and run-time initialization settings. This skeletal code creates a new thread type: `lwIP_sysboot_threadtype` (implemented in the `lwIP_sysboot_threadtype.c` and `lwIP_sysboot_threadtype.h` files), which acts as the boot thread and initializes the entire system at boot time. All of the network-related initialization is done in the run function, `lwIP_sysboot_threadtype_RunFunction()`. Users then create their own application threads and use the network APIs supported by LwIP without having to consider the lower-level networking hardware.

To bring up a network interface and its related TCP/IP stack, the following sequence of actions must be taken (as described in *LwIP User Guide*[5]). Figure 2 shows how these steps are implemented inside the run function, `lwIP_sysboot_threadtype_RunFunction()`, except step 2, which is done with the kernel property setting of the project.

1. Specify and use the header file for the socket API.
2. Ensure that sufficient VDK semaphores are configured.
3. Initialize the System Services Library (SSL) interrupt and device driver managers.
4. Initialize and set up the kernel API library.
5. Open the device driver for the Ethernet MAC controller and pass the handle of the driver to the *LwIP* stack.
6. Configure the EBIU controller to give DMA priority over the processor core (required only for the MAC controller of the ADSP-BF537 processor).
7. Supply the MAC address to the device driver.
8. Supply memory to the device driver to enable it to support the appropriate number of simultaneous reads and writes.
9. Notify the device driver library that the Ethernet driver will use the `Dataflow` method.
10. Initialize and configure the LwIP stack, supplying memory for the stack to use as its internal heap.
11. Notify the Ethernet driver that it should now start to operate.
12. Wait for the physical link to be established.

Once these steps are executed successfully, the network function is ready to use via the socket API.

Dual Network Interfaces

VisualDSP++ 4.5 does not provide a project wizard for two network interfaces. Most of the work has to be done manually, but the code generated for the single network interface can be reused. Additionally, the overall procedure is the same as in the single interface case (as shown in Figure 2).

Initializing Network Devices

Both of the two physical network devices must be initialized separately; however, the LwIP TCP/IP stack only needs to be initialized once because the two devices will share the same TCP/IP stack. It is the stack's responsibility to distinguish between the two devices and present each to the application correctly.

In Figure 2, the shaded blocks are the initialization of the individual physical network devices. These steps must be executed for each network device. A practical way to do it is:

1. Use the project wizard of VisualDSP++ to generate two TCP/IP application projects, one for each device. Test them separately to ensure that both work correctly[2][3][4][5].
2. Locate all device initialization codes in `lwIP_sysboot_threadtype_RunFunction` of `lwIP_sysboot_threadtype.c` in the two projects (the shaded blocks of Figure 2).
3. Rename the global variables assigned the same name in both projects to distinguish between the two devices. For example, `ADI_ETHER_SUPPLY_MEM memtable` can be differentiated by simply adding a suffix, such as `_LAN` in the extender card project and `_BF` in the EZ-Kit Lite project. Note that all references to these elements throughout the project must be changed accordingly.
4. Choose one project as the final main project, copy the device initialization code from the other project, and paste it just below where their counterparts reside in the main project. One exception is that the USB-LAN network device does not need "Set MAC address", since its driver module will set the MAC address internally when the device is started.

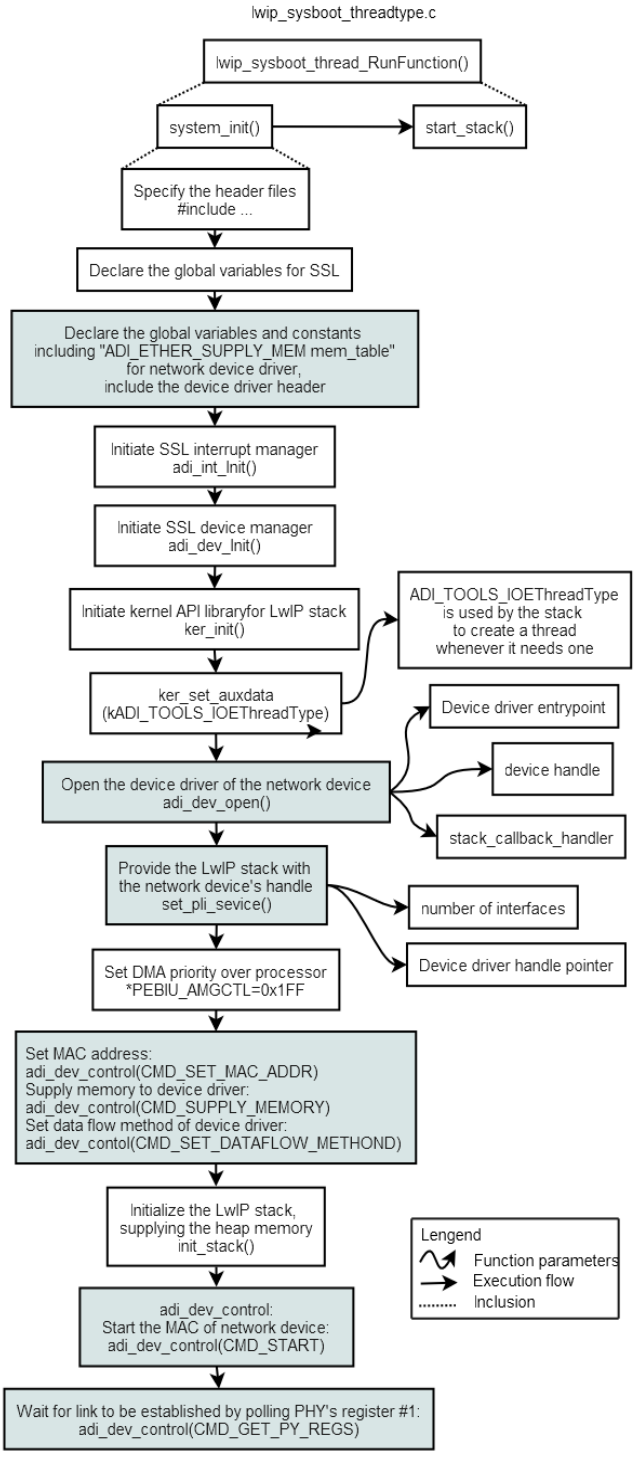

Figure 2. Steps to Bring Up One Network Interface

Providing Device Handles to the TCP/IP Stack

Both of the device handles returned by `adi_dev_open()` must be provided to the LwIP TCP/IP stack. This is done through the function call `set_pli_service()`, which takes two parameters, the number of device handles and the pointer to them. The handles must be arranged as an array, with the first device's handle in the zero location and the second handle in the one location. Listing 1 shows example code.

```
ADI_DEV_DEVICE_HANDLE handles[2];
#define handle_LAN ( handles[0] )
#define handle_BF ( handles[1] )
…
adi_dev_open( …, &handle_LAN, … );
adi_dev_open( …, &handle_BF, … );
…
set_pli_services( 2,handles );
```

Listing 1. Providing Two Device Handles to the LwIP Stack

Configuring Network Interfaces

Compiling the TCP/IP stack requires configuration information for each network interface. By default, this information is provided in the source file `default_user_config.c` when applications do not explicitly define another one. The configuration parameters are organized as a `struct net_config_info`. An array of this structure type (`user_net_config_info[]`) is defined to supply the settings of all interfaces, and the number of interfaces is derived from the size of the array. The default configuration code `default_user_config.c` defines only one interface.

To enable two network interfaces, a new configuration source file must be added to supply two interface settings. Although this file can be composed manually by expanding the default configuration code and adding it to the project, it can also be generated with the "TCP/IP Setting Plug-in" of VisualDSP++ 4.5 by following the steps below (details for this plug-in are described in *LwIP User Guide*[5]). If the plug-in is not listed in the `Settings` menu, it must be enabled by choosing `Settings -> Preferences` and then selecting `TCP/IP Configuration Manager`: on the `Plugins` page of the `Preferences` dialog box.

1. Choose `Settings -> TCPIP Configuration` to open the configuration dialog box.
2. On the `General` page, select the configuration file name.
3. On the `IP` page, specify `Number of Networks` as 2.
4. On the `Network 0` page and the `Network 1` page, set the necessary properties for both interfaces.
5. Click `OK` to close the configuration dialog box.

Two configuration source files are then generated and assigned the name of the configuration file (specified in Step 2) with the extensions `.c` and `.h`, respectively. They are added to the project under `Source Files` and `Header Files` in the Project window. In addition, a configuration file (`.tcp`) is created and added under `TCP/IP Configuration` in the Project window.

Linking to Libraries

The drivers for both network devices (`ADI_ETHER_BF537.dlb` for the ADSP-BF537 EZ-KIT Lite board's MAC controller and `lan91c111bf537.dlb` for the USB-LAN extender card's MAC Controller) are included in VisualDSP++ 4.5 as library files. Both files must be present in the project's library list, which can be set in the `Project Options` dialog box. In addition, the VDK library (`Kervdkbf537.dlb`), LwIP stack library (`liblwIPbf537.dlb`), and TCP/IP wrapper library (`Tcpipbf537.dlb`) are also necessary; the Project wizard usually adds these files to the library list automatically.

However, to enable the two network interfaces to work together, you must change the Tcpipbf537.dlb before it is linked to the project. The following steps describe how to make the modification:

1. Open the VisualDSP++ project file tcpip_wrapper-BF537.dpj under the directory:

   <install_path>\Analog Devices\VisualDSP 4.5\Blackfin\lib\src\lwIP\contrib\ports\ ADSP-Blackfin\proj\wrapperlib\

2. In the Project window under Source Files, locate and open the file pkthandler.c.

3. In the implementation of the function dhcp_configure(), locate the following instruction at the beginning of the function.

```
for(i=0; i<netinfo.num_if; i+=1) {
```

4. Change the code to:

```
for(i=netinfo.num_if-1; i>=0; i-= 1) {
```

5. Save the file and rebuild the project as a Release project configuration.

6. Copy the generated library file .\Release537\tcpip_wrapper-BF537.dlb to the library directory:

   <install_path>\Analog Devices\VisualDSP 4.5\Blackfin\lib

7. Replace the library file Tcpipbf537.dlb with tcpip_wrapper-BF537.dlb in the application project's library list.

The modified TCP/IP wrapper library is included in the associated ZIP file [6] for this EE-Note. It can be used directly to replace the original one.

Identify Network Interfaces

Since two network interfaces are running simultaneously, the application must be able to identify them.

The function int gethostaddr (int nwifce_no, char *host_addr) declared in cglobals.h under the directory <install path>\Analog Devices\VisualDSP 4.5\Blackfin\include\lwip can be used to get the IP address of either interface.

This function takes the number of the interface as its first parameter and fills its IP address as a string in the memory space pointed to by the second parameter. The two interfaces are numbered 0 and 1, according to the order of their handles in the handle array used by the set_pli_service() call. Using the code in Listing 1 as an example, interface number 0 corresponds to handle_LAN, and interface number 1 corresponds to handle_BF.

After acquiring the IPs of the two interfaces, applications can use the bind()socket API to select either one for their connections.

Example Code

The example code that accompanies this document brings up two network interfaces and runs four servers simultaneously in four threads: web server, echo server, character generation (CharGen) server, and discard server. Each network interface supports two servers. Most of the work described in previous sections is reflected in lwIP_sysboot_threadtype.c. Please follow the instructions in Readme.txt to run the example.

Conclusion

Two network interfaces can work simultaneously with the ADSP-BF537 EZ-Kit Lite board and USB-LAN Extender card to provide hardware support, and the VDK/LwIP combination provides the software support. Modifications must be made to the source code that is automatically generated by the VisualDSP++ 4.5 TCP/IP application Project wizard. The change affects the initialization of network devices, the configuration of network interfaces, and the LwIP library. The example code shows the success of this approach.

References

[1] *ADSP-BF537 Blackfin Processor Hardware Reference.* Rev 2.0, December 2005. Analog Devices, Inc.

[2] *ADSP-BF537 EZ-KIT Lite Evaluation System Manual.* Rev 2.0, June 2006. Analog Devices, Inc.

[3] *Blackfin USB-LAN EZ-Extender Manual.* Rev 2.0, April 2006. Analog Devices, Inc.

[4] *Getting Started with ADSP-BF537 EZ-KIT Lite.* Rev 1.1, April 2006. Analog Devices, Inc.

[5] *LwIP User Guide* (<install_path>\VisualDSP 4.5\Blackfin\lib\src\lwIP\docs\ LWIP_UserGuide.doc). Analog Devices, Inc.

[6] *Associated ZIP File.* Rev 1, December 2006. Analog Devices, Inc.

Document History

Revision	Description
Rev 1 – December 12, 2006 *by Jiang Wu*	Initial Release