

TMS320C6000 Network Developer's Kit (NDK) Support Package for DSK6455

User's Guide



Literature Number: SPRUES4A
January 2007–Revised June 2008

Preface	5
1 Getting Started	7
1.1 Introduction	7
1.2 Installing the Support Package	7
1.3 Rebuilding HAL Libraries	15
1.4 Required Terms and Concepts	17
2 User LED Driver	17
3 Timer Driver	17
4 Serial Driver	18
5 Ethernet Driver	18
5.1 Introduction	18
5.2 Ethernet Driver	19
5.3 Ethernet Packet Mini-Driver	20
Appendix A Revision History	24

List of Figures

1	HelloWorld Project Open in Code Composer Studio Window	9
2	Configuring Build Options Through Code Composer Studio	10
3	Enabling NIMU for Build Through Code Composer Studio	11
4	DSK6455 Connect Procedure	12
5	DSK6455 Connect	13
6	helloWorld Project Running Successfully on DSK6455	14
7	Source File Addition to Existing Project	15
8	Ethernet HAL Build From Sources	16

List of Tables

1	Ethernet Packet Driver Source Files.....	18
2	Structure Entries.....	20
A-1	Document Revision History	24

Read This First

About This Manual

This document contains information about the Network Developer's Kit (NDK) Support Package for DSK6455. The package includes source code to HAL drivers, and examples to reuse or modify for customer designed platforms. Pre-built HAL libraries are also delivered with the package.

How to Use This Manual

The information presented in this document is divided into the following sections:

- **Section 1 – Getting Started:** Introduces the NDK Support Package, which is designed to run the NDK on DSK6455 platform.
- **Section 2 – User LED Driver:** Describes the user LED driver for DSK6455.
- **Section 3 – Timer Driver:** Describes the timer driver for DSK6455.
- **Section 4 – Serial Driver:** Describes the serial driver for DSK6455.
- **Section 5 – Ethernet Driver:** Describes the EMAC driver for DSK6455.

Notational Conventions

This document uses the following conventions:

- Program listings, program examples, and interactive displays are shown in a special typeface.
- In syntax descriptions, the function or macro appears in a bold typeface and the parameters appear in plainface within parentheses. Portions of a syntax that are in bold should be entered as shown; portions of syntax that are within parentheses describe the type of information that should be entered.
- Macro names are written in uppercase text; function names are written in lowercase.

Related Documentation from Texas Instruments

The following books describe the TMS320C6x™ devices and related support tools. To obtain a copy of any of these TI documents, call the Texas Instruments Literature Response Center at (800) 477–8924. When ordering, please identify the book by its title and literature number. Many of these documents can be found on the Internet at <http://www.ti.com>.

[SPRU189](#) — TMS320C6000 DSP CPU and Instruction Set Reference Guide. Describes the CPU architecture, pipeline, instruction set, and interrupts for the TMS320C6000™ digital signal processors (DSPs).

[SPRU190](#) — TMS320C6000 DSP Peripherals Overview Reference Guide. Provides an overview and briefly describes the peripherals available on the TMS320C6000™ family of digital signal processors (DSPs).

[SPRU197](#) — TMS320C6000 Technical Brief. Provides an introduction to the TMS320C62x™ and TMS320C67x™ digital signal processors (DSPs) of the TMS320C6000™ DSP family. Describes the CPU architecture, peripherals, development tools and third-party support for the C62x™ and C67x™ DSPs.

[SPRU198](#) — TMS320C6000 Programmer's Guide. Reference for programming the TMS320C6000™ digital signal processors (DSPs). Before you use this manual, you should install your code generation and debugging tools. Includes a brief description of the C6000 DSP architecture and code development flow, includes C code examples and discusses optimization methods for the C code, describes the structure of assembly code and includes examples and discusses optimizations for the assembly code, and describes programming considerations for the C64x™ DSP.

[SPRU509](#) — **Code Composer Studio Development Tools v3.3 Getting Started Guide** introduces some of the basic features and functionalities in Code Composer Studio™ to enable you to create and build simple projects.

[SPRU523](#) — **TMS320C6000 Network Developer's Kit (NDK) Software User's Guide**. Describes how to use the NDK libraries, how to develop networking applications on TMS320C6000™ platforms, and ways to tune the NDK to fit a particular software environment.

[SPRU524](#) — **TMS320C6000 Network Developer's Kit (NDK) Software Programmer's Reference Guide**. Describes the various API functions provided by the stack libraries, including the low level hardware APIs.

Trademarks

TMS320C6x, TMS320C6000, TMS320C62x, TMS320C67x, C62x, C67x, C64x, Code Composer Studio, DSP/BIOS are trademarks of Texas Instruments.

TMS320C6000 Network Developer's Kit (NDK) Support Package for DSK6455

1 Getting Started

This section introduces the NDK support package for DSK6455.

1.1 Introduction

The TMS320C6000 NDK Support Package for DSK6455 includes:

- Source codes and pre-built libraries for NDK Hardware Adaptation Layer (HAL) drivers
- NDK examples

There are four basic HAL drivers required to operate the NDK: timer, user LED, serial port, and Ethernet. The Support Package provides the Ethernet, and user LED drivers specific to DSK6455 platform. The timer driver and serial stub driver are used from the NDK. The timer driver is implemented by using the DSP/BIOS™ PRD module from NDK.

1.2 Installing the Support Package

1. Install the Code Composer Studio software using the Code Composer Studio CD, if not installed already; it must be installed before the NDK Support Package software. Make sure that you install a version of Code Composer Studio 3.3 or higher.
2. Install the NDK software, if not already installed.
3. Copy the *ti.ndk.platforms.dsk6455.tar* under `<NDK_INSTALL_DIR>\packages` directory, and untar it. The support package installs over the NDK installation. Once installed, the following directories are created under the `<NDK_INSTALL_DIR>/packages/ti/ndk` directory:
 - `<docs/DSK6455>` Documentation files for the support package
 - `<example/network/cfgdemo/DSK6455>` Code Composer Studio project files for `cfgdemo` example
 - `<example/network/client/DSK6455>` Code Composer Studio project files for `client` example
 - `<example/network/helloWorld/DSK6455>` Code Composer Studio project files for `helloWorld` example
 - `<example/tools/DSK6455>` Common tools used by the support package
 - `<lib/hal/DSK6455>` Pre-built HAL libraries for DSK6455
 - `<src/hal/DSK6455/eth_c6455>` Source code for the C6455 EMAC driver
 - `<src/hal/DSK6455/userled_c6455>` Source code for DSK6455 LED driver
4. The network management interface unit (NIMU) is introduced in NDK 1.94.00.001 as an alternative to LL packet architecture to add support for multiple instances of drivers. Several builds of the Ethernet driver libraries are provided with NIMU and LL architectures for Big and Little Endian architectures under `<lib/hal/DSK6455>`. The libraries are named as `<lib_name>.lib` for little endian and `<lib_name>.e.lib` for big endian. The libraries built using NIMU are named as `<lib_name>_nimu.lib` and the ones built using LL architecture are named as `<lib_name>_ll.lib`. Please choose the most appropriate version of the library based on the endianness and whether NIMU/LL is required and rename the library to just `<lib_name>.lib` for using it with any example projects.
For example, If an application requires NIMU little endian HAL ethernet driver, rename `<lib/hal/dsk6455/hal_eth_c6455_nimu.lib>` to `<lib/hal/dsk6455/hal_eth_c6455.lib>`.

5. Choose the appropriate version of the user LED driver and rename it, based on endianness. The little endian version of the library is named as <lib_name>.lib and the big endian counterpart is named as <lib_name>e.lib.
For example, If the target board is big-endian platform then copy over the <lib/hal/dsk6455/hal_userled_c6455e.lib> to <lib/hal/dsk6455/hal_userled_c6455.lib>.
6. Copy the correct NDK libraries for testing based on endianness and NIMU/LL architectures.
For example, to compile your application with NIMU for a little endian target, change the directory to <NDK_INSTALL_DIR>/packages/ti/ndk/lib/C64plus> and execute the following commands at the command prompt:

```
copy netctrl_nimu.lib netctrl.lib
copy nettool_nimu.lib nettool.lib
copy all_stk\stk_nimu.lib stack.lib
```

 For more details on choosing and building in appropriate NDK libraries, see the *NDK Libraries* and *Building in NIMU* sections of the *TMS320C6000 Network Developer's Kit (NDK) User's Reference Guide* ([SPRU523](#)).
7. Configure the wnvionment variables; for DSK6455 the following variables should be configured
NDK_INSTALL_DIR = <ndk_1_94 installation directory>
8. Make sure that the PC is connected through USB/JTAG to the DSK6455 correctly. Click on the Code Composer Studio icon typically placed on the Desktop of the PC to open Code Composer Studio as shown below.



The Code Composer enumerates the USB and should pop up. If there are any errors and the USB enumeration fails, check the USB connectivity with the board and refer to the Quick Start Guide which is a part of the Digital Spectrum CD.

- Open Code Composer. Click on Project → Open and browse to the <NDK_INSTALL_DIR>/packages/ti/ndk/example/network/helloWorld/dsk6455>. Select the helloWorld.pjt. It should open up the project as shown in Figure 1:

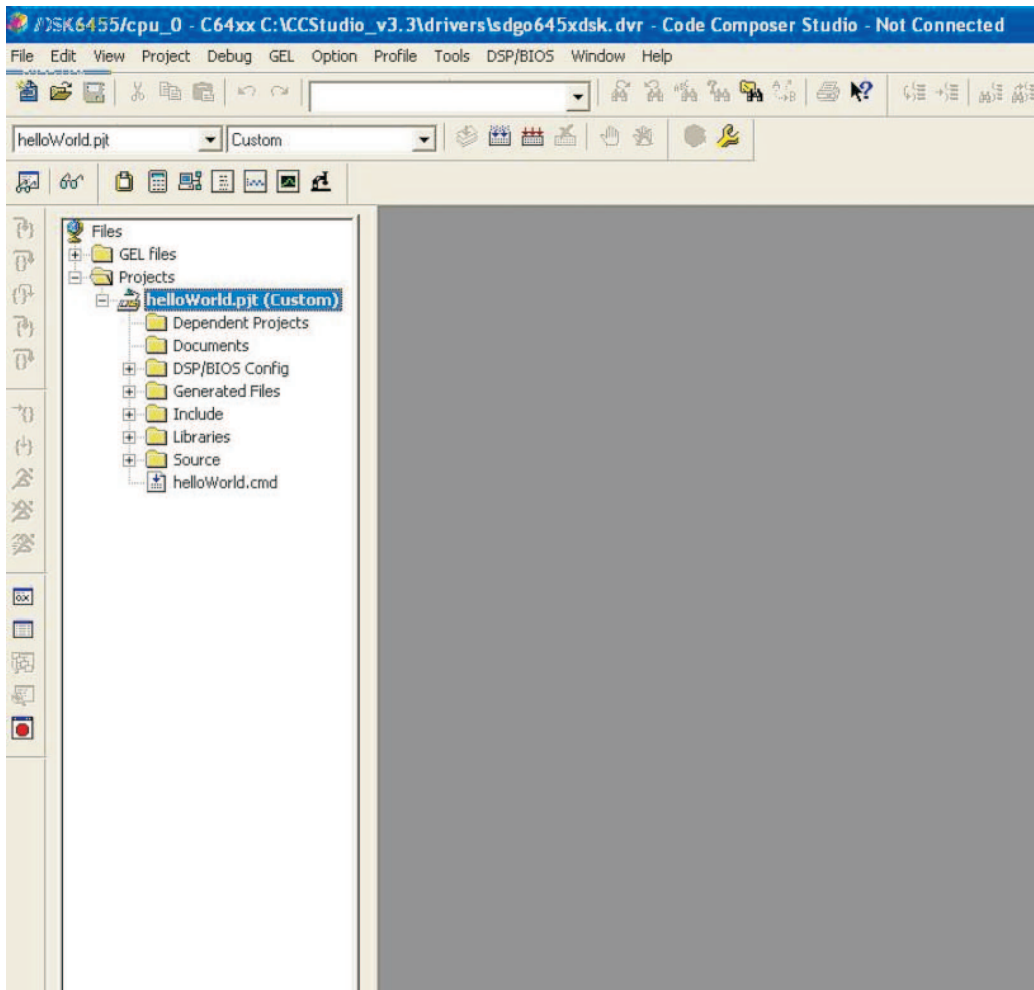


Figure 1. HelloWorld Project Open in Code Composer Studio Window

10. To build the example project with NIMU enabled using Code Composer Studio (CCS) IDE, please follow the following steps:
- a. Open the application project in Code Composer Studio IDE. Go to Project → Build Options and click on it as shown in [Figure 2](#):

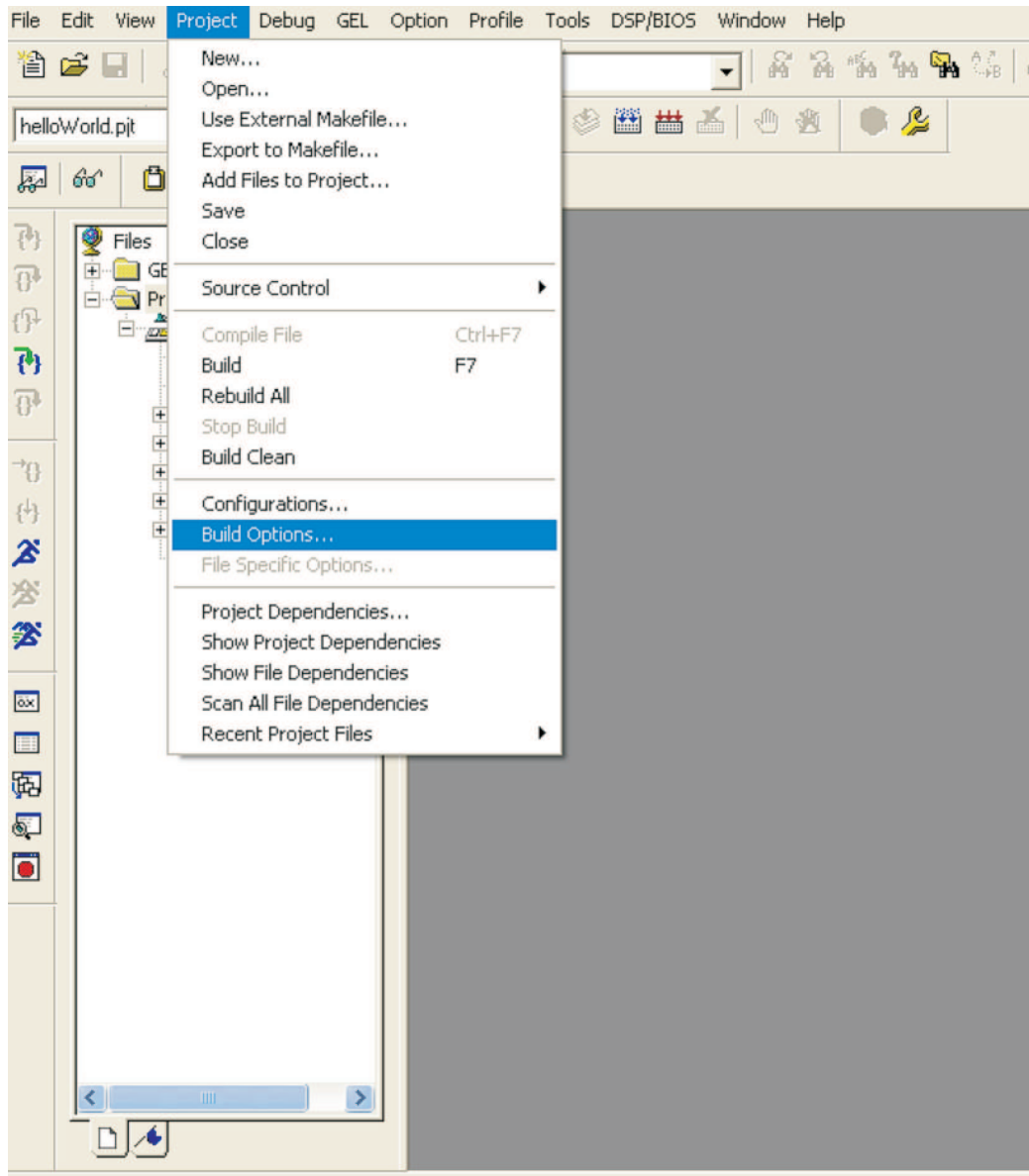


Figure 2. Configuring Build Options Through Code Composer Studio

2. Click on *Preprocessor* in the *Category* box of the *Compiler* tab on the Build Options tab that appears, as shown in [Figure 3](#):

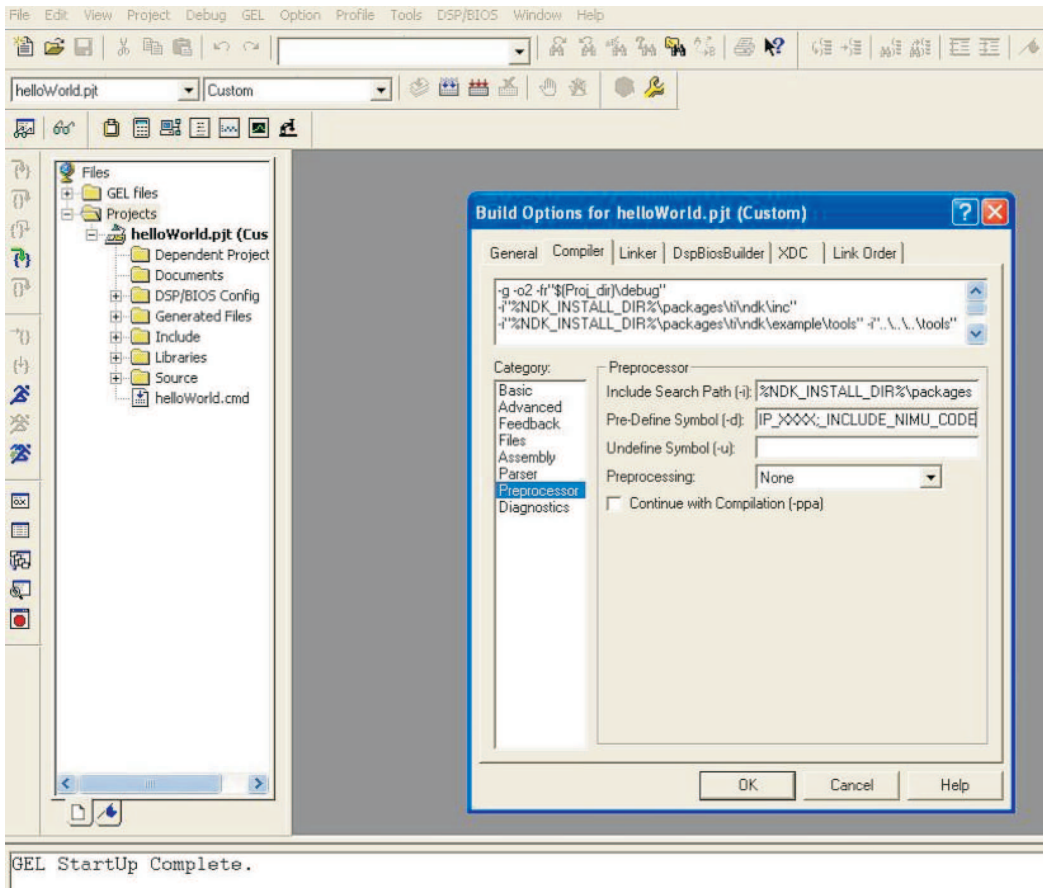


Figure 3. Enabling NIMU for Build Through Code Composer Studio

- c. Add the following constant at the end and click the *OK* button to apply the settings in the *Pre-Define Symbol (-d)* box as shown in [Figure 3](#):
`_INCLUDE_NIMU_CODE;`
- d. Finally, build the application project by following the steps mentioned next, to make sure NIMU is built into the final executable.

11. Connect to the debugger on the DSK6455 by either entering *Alt + C* or by clicking on Debug → Connect if not already connected, as shown in Figure 4.

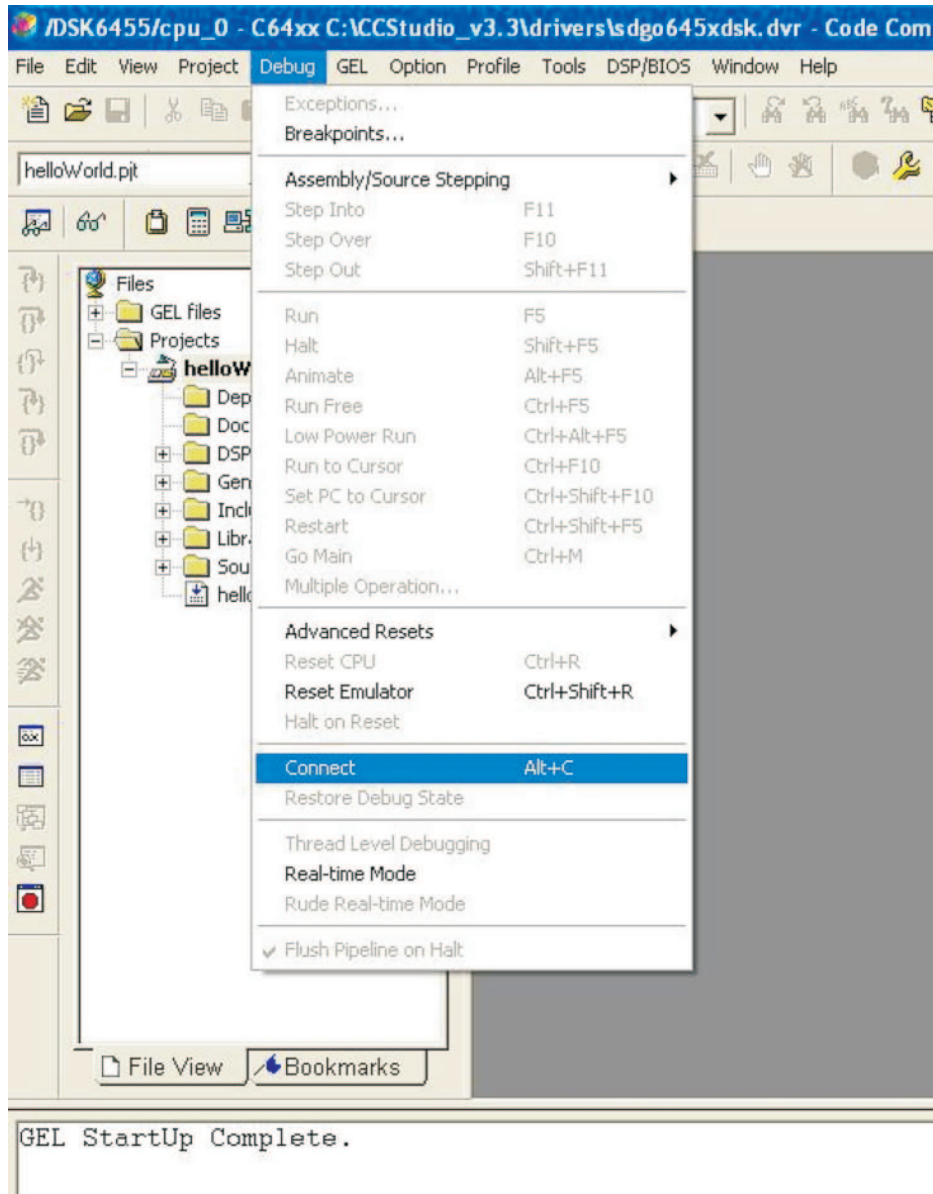


Figure 4. DSK6455 Connect Procedure

- Once connected, [Figure 5](#) shows up displaying that the Code Composer Studio debugger is now connected to the target.

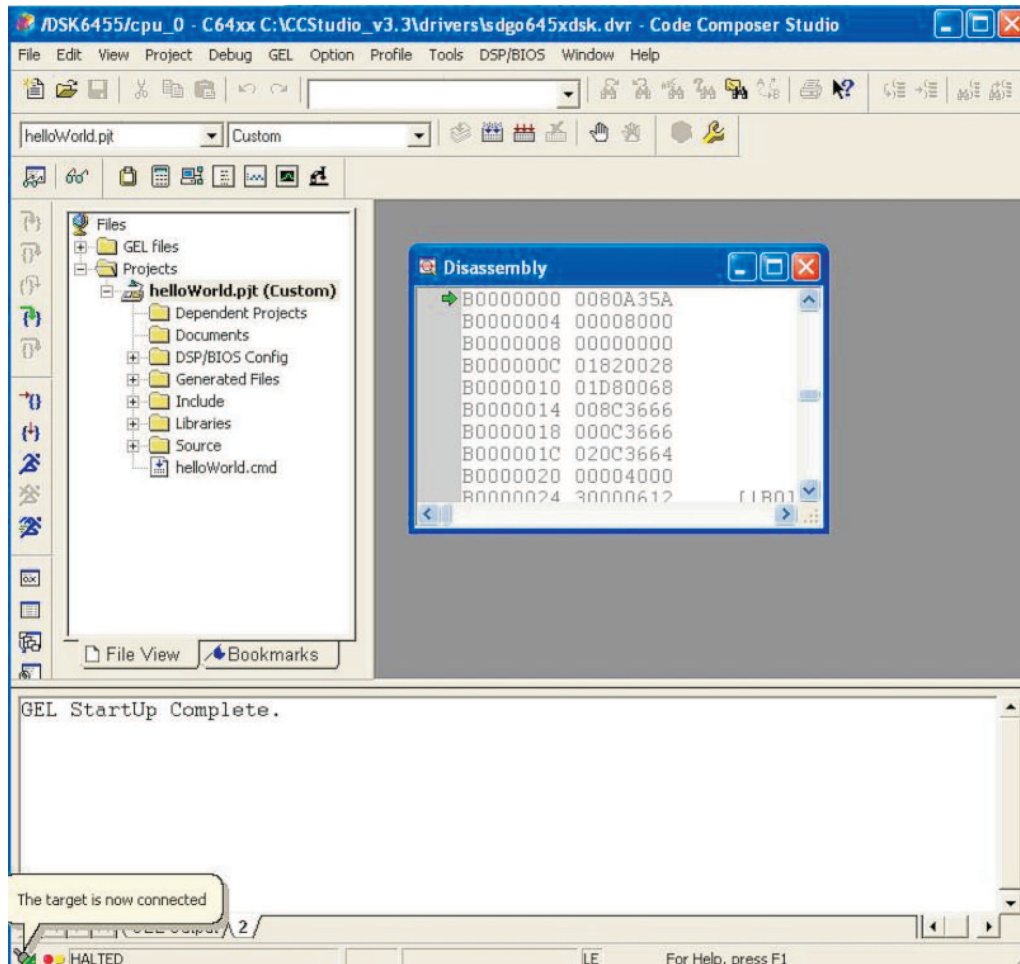
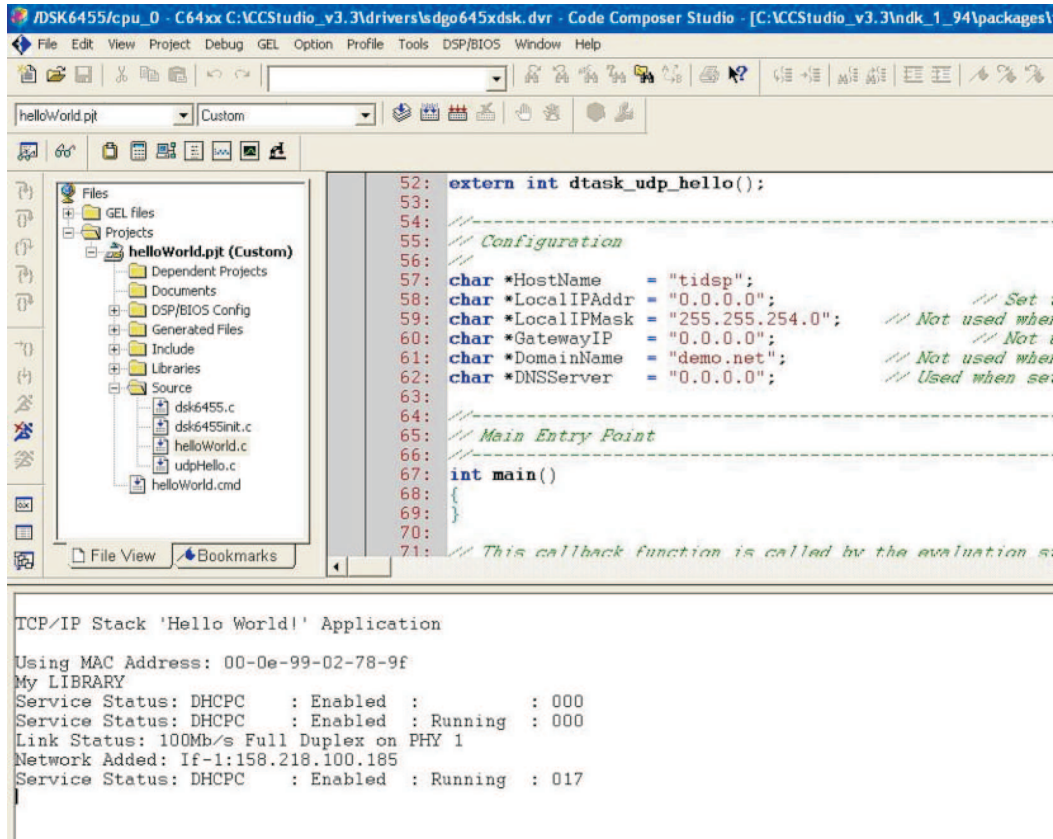


Figure 5. DSK6455 Connect

13. Enter *F7* or go to Project → Build and click on it to build the project. Once the project is built without any errors, load the project onto the DSK6455 platform by clicking on File → Load Program and browsing to bin folder of the helloWorld project and select helloWorld.out. To start the helloWorld application on the target, enter *F5* or click on Debug → Run as shown in [Figure 6](#)



```

52: extern int dtask_udp_hello();
53:
54:
55: // Configuration
56:
57: char *HostName = "tidsp";
58: char *LocalIPAddr = "0.0.0.0"; // Set
59: char *LocalIPMask = "255.255.254.0"; // Not used when
60: char *GatewayIP = "0.0.0.0"; // Not u
61: char *DomainName = "demo.net"; // Not used when
62: char *DNSServer = "0.0.0.0"; // Used when se
63:
64: // Main Entry Point
65:
66:
67: int main()
68: {
69:
70:
71: // This callback function is called by the evaluation s
    
```

```

TCP/IP Stack 'Hello World!' Application
Using MAC Address: 00-0e-99-02-78-9f
My LIBRARY
Service Status: DHCP      : Enabled : Running : 000
Service Status: DHCP      : Enabled : Running : 000
Link Status: 100Mb/s Full Duplex on PHY 1
Network Added: If-1:158.218.100.185
Service Status: DHCP      : Enabled : Running : 017
    
```

Figure 6. helloWorld Project Running Successfully on DSK6455

1.3 Rebuilding HAL Libraries

Code Composer Studio is the recommended tool to rebuild the HAL libraries, if needed. To build any specific HAL library from Code Composer Studio, add the relevant HAL library sources to the Code Composer Studio application project and recompile as illustrated in the steps below:

1. Right click on the project file in the Project window, as shown in Figure 7, and select *Add Files to Project*, to add any HAL library sources to the Code Composer Studio project. This opens up a window so that you can browse to the required HAL sources directory and select the appropriate source code files and add them.

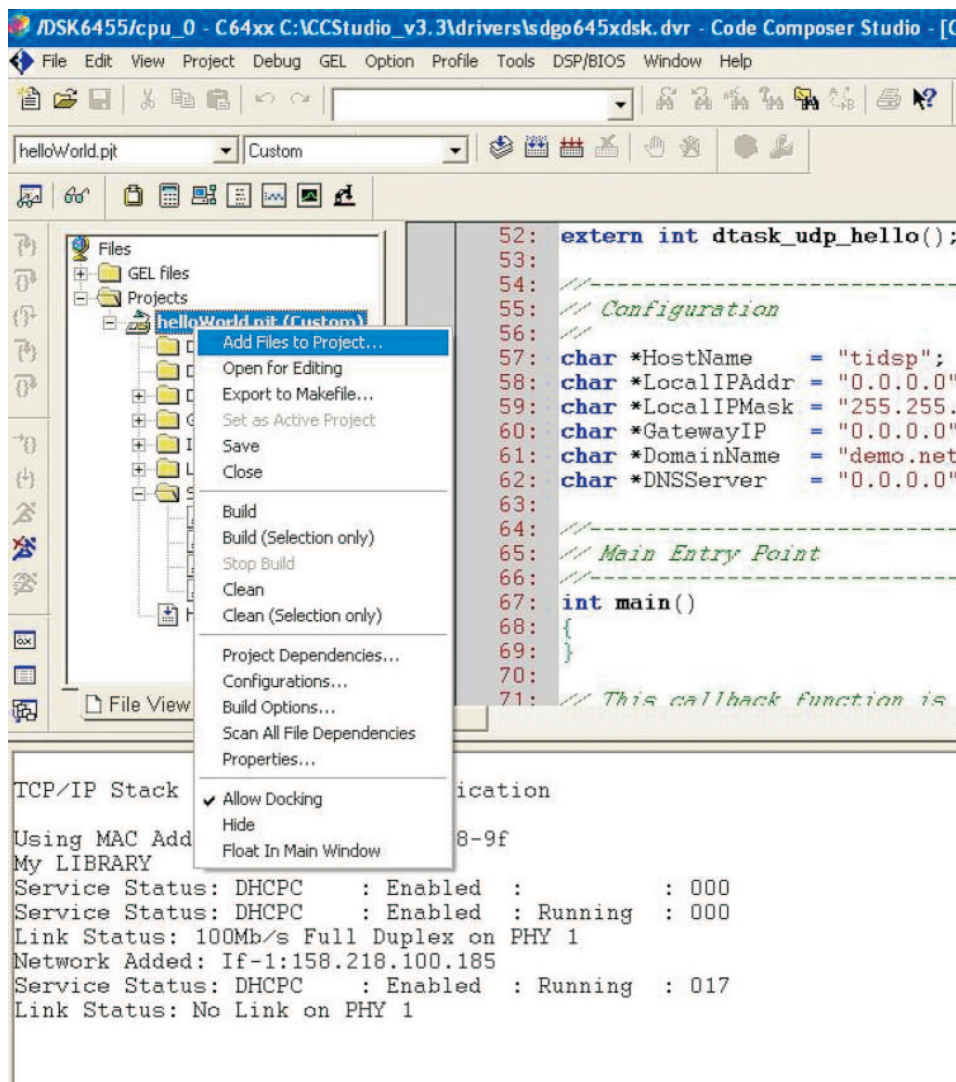


Figure 7. Source File Addition to Existing Project

For example, to add Ethernet driver source files, browse to
 <NDK_INSTALL_DIR>/packages/ti/ndk/src/hal/dsk6455/eth_c6455>
 and select the required .c files as shown in Figure 8.

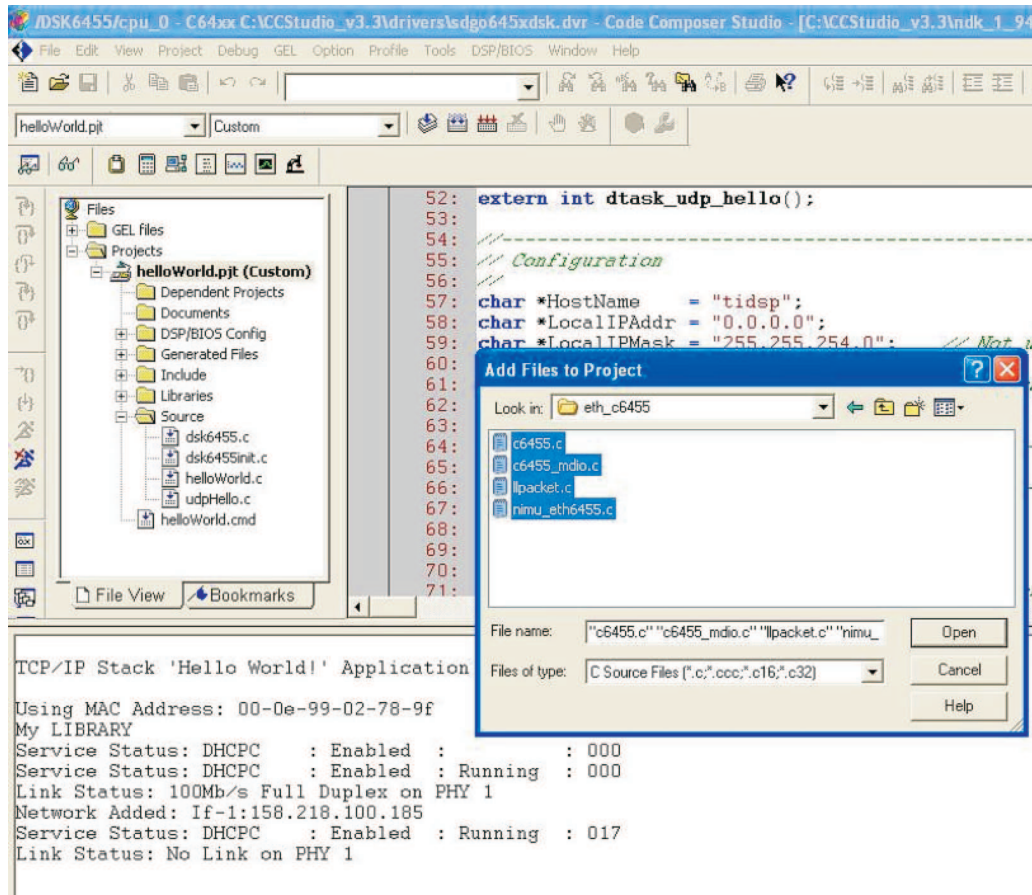


Figure 8. Ethernet HAL Build From Sources

To rebuild the DSK6455 Ethernet driver from sources, ensure the following files are added to project:

With NIMU add:

<NDK_INSTALL_DIR>/packages/ti/ndk/src/hal/dsk6455/eth_c6455/c6455.c>

<NDK_INSTALL_DIR>/packages/ti/ndk/src/hal/dsk6455/eth_c6455/c6455_mdio.c>

<NDK_INSTALL_DIR>/packages/ti/ndk/src/hal/dsk6455/eth_c6455/nimu_eth6455.c>

With LL add:

<NDK_INSTALL_DIR>/packages/ti/ndk/src/hal/dsk6455/eth_c6455/c6455.c>

<NDK_INSTALL_DIR>/packages/ti/ndk/src/hal/dsk6455/eth_c6455/c6455_mdio.c>

<NDK_INSTALL_DIR>/packages/ti/ndk/src/hal/dsk6455/eth_c6455/llpacket.c>

To rebuild the DSK6455 user LED driver from sources, ensure the following file is added to project:

<NDK_INSTALL_DIR>/packages/ti/ndk/src/hal/dsk6455/userled_c6455/llled.c>

For further explanation on the user LED driver and Ethernet driver files and their contents, see [Section 2](#) and [Section 5](#) of this document, respectively.

2. Rebuild the project once the necessary HAL library source files have been added to the project.

1.4 Required Terms and Concepts

To port the NDK Support Package device drivers, you should be familiar with the following concepts.

1.4.1 HAL Driver Source Files

[Section 1.3](#) described how to build different HAL drivers for DSK6455.

1.4.2 Network Control Module (NETCTRL)

The network control module (NETCTRL) is at the center of the NDK and controls the interface of the HAL device drivers to the internal stack functions.

The NETCTRL module and its related APIs are described in both the *TMS320C6000 Network Developer's Kit (NDK) Software Programmer's Reference Guide* ([SPRU524](#)) and the *TMS320C6000 Network Developer's Kit (NDK) Software User's Guide* ([SPRU523](#)). To write device drivers, you must be more familiar with NETCTRL. The description given in the *TMS320C6000 Network Developer's Kit (NDK) Software User's Guide* ([SPRU523](#)) is more appropriate for device driver work.

1.4.3 Stack Event (STKEVENT) Object

The STKEVENT event object is a central component to the low-level architecture. It ties the HAL layer to the scheduler thread in the network control module (NETCTRL). The network scheduler thread waits on events from various device drivers in the system, including the Ethernet, serial, and timer drivers.

The device drivers use the STKEVENT object to inform the scheduler that an event has occurred. The STKEVENT object and its related API are described in the *TMS320C6000 Network Developer's Kit (NDK) Programmer's Reference Guide* ([SPRU524](#)). Device driver writers need to be familiar with STKEVENT.

1.4.4 Packet Buffer (PBM) Object

The PBM object is a packet buffer that is sourced and managed by the Packet Buffer Manager (PBM). The PBM is part of the OS adaptation layer. It provides packet buffers for all packet based devices in the system. Therefore, the serial port and Ethernet drivers both make use of this module.

The PBM object and its related API are described in the *TMS320C6000 Network Developer's Kit (NDK) Programmer's Reference Guide* ([SPRU524](#)). The *TMS320C6000 Network Developer's Kit (NDK) User's Guide* ([SPRU523](#)) also includes a section on adapting the PBM to a particular included software.

2 User LED Driver

This section describes the User LED software. The User LED driver is a collection of functions that turn on and off LED lights on the DSK6455 platform. There is only one C file for the User LED: LLLLED.C LED driver, located in the subdirectory SRC\HAL\DSK6455\USERLED_C6455.

3 Timer Driver

This section discusses the software that drives event timing. The timer driver determines the timing for all time driven events in the NDK. The DSK6455 platform uses the NDK provided *timer_bios* driver, which is implemented using a DSP/BIOS PRD object.

4 Serial Driver

There are no serial drivers supported on the DSK6455 platform. The NDK delivered serial stub driver is used instead.

5 Ethernet Driver

This section describes the operational theory of the low-level Ethernet driver, including instructions on the use and porting of the device driver source code.

5.1 Introduction

The Ethernet packet driver, provided in NDK, is broken down into two parts: a device independent upper layer and a device dependent layer. The device dependent layer is called a mini-driver because it only implements a subset of the full driver functions; the mini-driver API is documented at the end of this section. The device independent upper layer of the Ethernet packet driver can be implemented in two ways: using the old-style LL packet driver architecture and using the new NIMU architecture. The NIMU architecture enables the NDK stack to control and communicate with multiple instances of a driver. This is more beneficial when compared to the old LL packet driver architecture; originally the NDK stack could communicate with only one instance of the LL driver at a time. An NDK Ethernet driver developer needs to make a design choice of either going with the LL packet driver or NIMU architectures and develop the driver accordingly. The LL packet driver architecture API is documented in the *Hardware Adaptation Layer (HAL)* section of the *TMSC320C6000 Network Developer's Kit (NDK) Programmer's Reference Guide (SPRU524)*. The NIMU architecture and API are described in detail in the *Network Interface Management Unit* section of the *TMS320C6000 Network Developer's Kit (NDK) Programmer's Reference Guide (SPRU524)*. Also, the NIMU NDK core stack changes and build instructions can be found in the *Stack Library Design and Building in NIMU* sections of the *TMS320C6000 Network Developer's Kit (NDK) User's Guide (SPRU523)*, respectively.

5.1.1 Ethernet Driver Source Files

The Ethernet packet driver source files are located in various subdirectories according to their function.

Table 1. Ethernet Packet Driver Source Files

Directory	File	Function
<SRC\HAL\DSK6455\ETH_C6455>		Source code of the Texas Instruments C6455 Ethernet driver
	NIMU_ETH6455.C	Hardware independent portion of Ethernet packet driver using NIMU architecture
	LLPACKET.C	Hardware independent portion of the low-level Ethernet packet driver
	LLPACKET.H	Private include file for LLPACKET and NIMU drivers
	C6455.C	Packet mini-driver for C6455 EMAC
	C6455_MDIO.C	MDIO control functions for C6455 EMAC
	C6455_MDIO.H	Header file of MDIO file
	C6455_COMMON.H	Common peripheral register structures and definitions
	CSLR.H	Central register layer - contains field-manipulation macro definitions
	CSLR_DEV.H	Register descriptions for DEV
	CSLR_ECTL.H	Register descriptions for ECTL
	CSLR_EMAC.H	Register descriptions for EMAC
	CSLR_MDIO.H	Register descriptions for MDIO

5.2 Ethernet Driver

The NDK LL packet driver API and the NIMU API are discussed in the *Hardware Adaptation Layer (HAL)* and *Network Interface Management Unit* sections of the *TMS320C6000 Network Developer's Kit (NDK) Programmer's Reference Guide* ([SPRU524](#)), respectively, which includes how to implement the individual API functions. The sections below discuss the implementation of an Ethernet packet mini-driver.

5.2.1 Important Note on Data Alignment

The NDK libraries have been built with the assumption that the IP header in a data packet is 16-bit aligned. In other words, the first byte of the IP packet (the version/length field) must start on an even 16-bit boundary. In any fixed length header protocol, this requirement can be met by backing off any odd byte header size, and adding it to the header padding specified to the stack. For Ethernet and peer to peer protocol (PPP), the only requirement is that the Ethernet or PPP packet not start on an odd byte boundary.

In addition, all drivers in the NDK are set up to have a 22 byte header. This is the header size of a PPPoE packet when sent using a 14 byte Ethernet header. When all arriving packets use the 22 byte header, it guarantees that they can be routed to any egress device with a header requirement up to that size. For Ethernet operation, this requires that a packet has 8 bytes of pre-pad to make its total header size 22 bytes.

The value of this pre-pad is #defined as `PKT_PREPAD` in the file `LLPACKET.H`.

5.2.2 Hardware Independent Low-Level Ethernet Driver

The low-level Ethernet packet driver is similar to the low-level serial port driver. It consists of two parts: a hardware independent module and a hardware specific module, which makes the hardware specific portion of the driver easier to port. When deciding how to port the packet driver, you must choose for the device independent module either the `LLPACKET.C` or the NIMU enabled counterpart `NIMU_ETH6455.C`. Currently, the choice between NIMU and `LLPacket` modules is based on whether the pre-processor symbol `_INCLUDE_NIMU_CODE` is defined or not. If this constant is not defined, `LLPACKET.C` is compiled in, otherwise `NIMU_ETH6455.C` is compiled in. For instructions to build in NIMU using Code Composer Studio, see Step 10 in [Section 1.2](#) of this document, and the *TMS320C6000 Network Developer's Kit (NDK) User's Guide* ([SPRU523](#)).

The standard API to access the packet device, as defined in the *Hardware Adaptation Layer (HAL)* section of the *TMS320C6000 Network Developer's Kit (NDK) Programmer's Reference Guide* ([SPRU524](#)), is implemented by the `LLPACKET.C` module. The NIMU architecture and API, described in detail in the *Network Interface Management Unit* section of the *TMS320C6000 Network Developer's Kit (NDK) Programmer's Reference Guide* ([SPRU524](#)), is implemented by the `NIMU_C6455.C` module. The NIMU enabled module can handle multiple device instances, and is charged with handling the queuing for all received packet data.

To implement the low-level packet API in a device independent manner, the `LLPACKET.C` or `NIMU_ETH6455.C` modules call down to a hardware specific module.

The interface functions to this module are defined in the `LLPACKET.H` include file. They are documented to some degree in the example source code of the hardware specific modules. The `LLPACKET.H` file also contains the specifications for the buffering of packets.

5.2.3 Hardware Specific Low-Level Ethernet (Mini) Driver

The mini-driver module is a device driver specific to its target hardware. Its basic function is to communicate with the Ethernet MAC hardware. It also must interface to any other hardware specific to the target platform. For example, it can set up interrupts, cache control, and the EDMA controller.

The interface specification is capable of handling multiple devices, but the example implementations mostly only support a single device instance. Notes are made in the source code as to where alterations can be made to support multiple devices.

5.3 Ethernet Packet Mini-Driver

5.3.1 Overview

As mentioned in the previous section, the low-level Ethernet packet driver is broken down into two distinct parts: a hardware independent module (LLPACKET.C/NIMU_ETH6455.C) that implements the IIPacket/NIMU APIs and a hardware specific module that interfaces to the hardware independent module. This section describes this small hardware specific module, or mini-driver.

Note that this module is purely optional. A valid packet driver can be developed by directly implementing the IIPacket/NIMU API described in the *TMS320C6000 Network Developer's Kit (NDK) Programmer's Reference Guide* ([SPRU524](#)). Even if the mini-driver is used, you may change any of the internal data structures as long as the IIPacket/NIMU interface remains unchanged.

5.3.2 Global Instance Structure

Nearly all the functions in the mini-driver API take a pointer to a packet driver instance structure called PDINFO. This structure is defined in LLPACKET.H:

```
//
// Packet device information
//
typedef struct _pdinfo {
    uint                PhysIdx;                // Physical index of this device
    (0 to -1)
    HANDLE              hEther;                // Handle to logical driver
    STKEVENT_Handle    hEvent;
    UINT8               bMacAddr[6];          // MAC Address
    uint                Filter;                // Current RX filter
    uint                MCastCnt;             // Current MCast Address Countr
    UINT8               bMCast[6*PKT_MAX_MCAST];
    uint                TxFree;                // Transmitter "free" flag
    PBMQ                PBMQ_tx;              // Tx queue (one for each PKT
device)
#ifdef _INCLUDE_NIMU_CODE
    PBMQ                PBMQ_rx;              // Rx queue (one for all PKT
devices)
#endif
} PDINFO;
```

Only some of these fields are used in a mini-driver. The structure entries are defined as follows:

Table 2. Structure Entries

Field	Description
PhysIdx	Physical Index of This Device (0 to –1). The physical index of the device determines how the device instance is represented to the outside world. The mini-driver is not concerned about the physical index.
hEther	Handle to Ethernet Driver. This is a handle NDK Ethernet instance that is bound to the physical Ethernet driver. When a packet is received, it is tagged with this Ethernet handle before being placed on the global PBMQ_rx queue. This allows the Ethernet module to identify the ingress device.
hEvent	Handle to Scheduler Event Object. The handle hEvent is used with the STKEVENT function <i>STKEVENT_signal()</i> to signal the system whenever a new packet is received.
bMacAddr	Ethernet MAC Address. This is a byte array that holds the Ethernet MAC address. It is set to a default value by LLPACKET.C/NIMU_ETH6455.C, but can be used or altered by the mini-driver when the device opens. If the MAC contains its own unique MAC address, this value is written to bMacAddr. If the MAC does not have a MAC address, the value bMacAddr programs the MAC device.

Table 2. Structure Entries (continued)

Field	Description
Filter	Current Rx Filter. The receive filter determines how the packet device should filter incoming packets. This field is set by LLPACKET.C/NIMU_ETH6455.C and used by the mini-driver to program the MAC. Legal values include: <ul style="list-style-type: none"> ETH_PKTFLT_NOTHING: No Packets ETH_PKTFLT_DIRECT: Only directed Ethernet ETH_PKTFLT_BROADCAST: Directed plus Ethernet Broadcast ETH_PKTFLT_MULTICAST: Directed, Broadcast, and selected Ethernet Multicast ETH_PKTFLT_ALLMULTICAST: Directed, Broadcast, and all Multicast ETH_PKTFLT_ALL: All packets
MCastCnt	Number of Multicast Addresses Installed. The field holds the current number of multicast addresses stored in the multicast address list (also in this structure). The multicast address list determines what multicast addresses (if any) the MAC is allowed to receive.
bMCast	Multicast Address List. This field is a byte array of consecutive 6 byte multicast MAC addresses. The number of valid addresses is stored in the MCastCnt field. The multicast address list determines what multicast addresses (if any) the MAC is allowed to receive.
TxFree	Transmitter Free Flag. The TxFree flag is used by LLPACKET.C/NIMU_ETH6455.C to determine if a new packet can be sent immediately by the mini-driver, or if it should be placed on the transmit pending queue for later. If the flag is not zero, the mini-driver function <i>HwPktTxNext()</i> is called when a new packet is queued for transmission. This flag is maintained by the mini-driver.
PBMQ_tx	Transmit Pending Queue. The transmit pending queue holds all the packets waiting to be sent on the Ethernet device. The mini-driver pulls PBM packet buffers off this queue in its <i>HwPktTxNext()</i> function and posts them to the Ethernet MAC for transmit. Once the packet has been transmitted, the packet buffer is freed by calling <i>PBM_free()</i> .

5.3.3 Mini-Driver Operation

The Ethernet packet mini-driver maintains the device hardware, and services any required communications interrupts. It is built around a simple open/close concept. When open, the driver is active, and when closed, it is not. In general, it must implement the mini-driver API described in the following section. The following sections provide additional information on its internal operation.

5.3.3.1 Receive Operation

The mini-driver receives packets when the device is open. When an Ethernet packet is received, it is placed in a PBM packet buffer. Empty packet buffers are allocated by calling *PBM_alloc()*.

Once the packet buffer is filled, it should be placed onto the receive pending queue (PBMQ_rx) defined in the LLPACKET.H. For LLPacket style devices, there is only one RX queue for all Ethernet devices, therefore, the mini-driver must set the RX IF device to the value of hEther in the instance structure before placing it on the RX queue. Contrary to this, the NIMU style devices have one RX queue each for each Ethernet device instance; therefore, the mini-driver can simply enqueue the packet buffer onto the RX queue of that Ethernet instance.

After the data frame buffer has been pushed onto the Rx queue, the mini-driver signals an Ethernet event to the STKEVENT handle supplied in the driver instance structure.

5.3.3.2 Transmit Operation

When the transmitter is idle, the mini-driver must set the TxFree field of its instance structure to 1. When a new packet is ready for transmission, LLPACKET.C/NIMU_ETH6455.C places the PBM packet buffer on the PBMQ_tx queue of the mini-driver's instance structure.

Once a new packet has been written to the transmit pending queue, if TxFree is set, LLPACKET.C/NIMU_ETH6455.C calls the mini-driver *HwPktSendNext()* function. At this time, the mini-driver should clear the TxFree field, and start transmission of the packet. Once the packet has been sent, the packet buffer is freed by calling *PBM_free()*. This call can be made at interrupt time.

5.3.4 Ethernet Packet Mini-Driver API

The following API functions must be provided by a mini-driver.

HwPktInit	<i>Initialize Packet Driver Environment</i>
Syntax	uint HwPktInit();
Parameters	None
Return Value	The number of Ethernet packet devices in the system
Description	Called to initialize the packet mini-driver environment, and enumerate the number of devices in the system. A device instance may be opened for each device represented in the return count. If the function returns zero, no devices are supported.
HwPktShutdown	<i>Shutdown Packet Driver Environment</i>
Syntax	void HwPktShutdown();
Parameters	None
Return Value	None
Description	Called to indicate that the packet driver environment should be completely shut down.
HwPktOpen	<i>Open Ethernet Packet Device Instance</i>
Syntax	uint HwPktOpen(PDINFO *pi);
Parameters	pi- Pointer to Ethernet packet device instance structure
Return Value	Returns 1 if the driver was opened, or 0 on error.
Description	Called to open a packet device instance. When called, PDINFO structure is valid. The device should be opened and made ready to receive and transmit Ethernet packets.
HwPktClose	<i>Close Ethernet Packet Device Instance</i>
Syntax	void HwPktClose(PDINFO *pi);
Parameters	pi- Pointer to Ethernet packet device instance structure
Return Value	None
Description	Called to close a packet device instance. When called, any outstanding packet buffers held by the instance should be freed using <i>PBM_free()</i> .

HwPktTxNext	<i>Transmit Next Buffer in Transmit Queue</i>
Syntax	void HwPktTxNext(PDINFO *pi);
Parameters	pi - Pointer to Ethernet packet device instance structure
Return Value	None
Description	Called to indicate that a packet buffer has been queued in the transmit pending queue contained in the device instance structure, and LLPACKET.C/NIMU_ETH6455.C believes the transmitter to be free (TxFree set to 1). The mini-driver uses this function to start the transmission sequence.
HwPktSetRx	<i>Set Ethernet Rx Filter</i>
Syntax	void HwPktSetRx(PDINFO *pi);
Parameters	pi - Pointer to Ethernet packet device instance structure
Return Value	None
Description	Called when the values contained in the PDINFO instance structure for the Rx filter or multicast list are altered. The mini-driver should update its filter settings at this time.
HwPktIoctl	<i>Execute Driver Specific IOCTL Command</i>
Syntax	uint HwPktIoctl(PDINFO *pi, uint cmd, void *arg);
Parameters	pi - Pointer to Ethernet packet device instance structure cmd - Device specific command arg - Pointer to command specific argument
Return Value	This function returns 1 on success.
	Description Execute driver specific IOCTL command. There is no command defined for existing driver.
_HwPktPoll	<i>Mini-Driver Polling Function</i>
Syntax	void _HwPktPoll(SDINFO *pi, uint fTimerTick);
Parameters	pi - Pointer to serial device instance structure fTimerTick - Flag indicating the 100 ms have elapsed
Return Value	None
Description	Called by LLPACKET.C/NIMU_ETH6455.C at least every 100 ms, but calls can come faster when there is network activity. The mini-driver is not required to perform any operation in this function, but it can be used to check for device lockup conditions. When the call is made due to the 100 ms time tick, the fTimerTick calling parameter is set. Note that this function is not called in kernel mode (hence, the underscore in the name). This is the only mini-driver function called from outside kernel mode (done to support polling drivers).

Appendix A Revision History

Table A-1 lists the changes made since the previous version of this document.

Table A-1. Document Revision History

Reference	Additions/Modifications/Deletions
Section 1.2	Replaced contents with new source
Section 1.3	Replaced contents with new source
Section 5.1	Replaced contents with new source
Table 2	Replaced contents of the table
Section 5.2	Replaced contents with new source
Section 5.2.2	Changed title of the section
Section 5.2.2	Replaced contents with new source
Section 5.3.1	Replaced contents with new source
Section 5.3.2	Replaced contents with new source
Section 5.3.2	Replaced text in table - bMacAddr
Section 5.3.2	Replaced text in table - Filter
Section 5.3.2	Replaced text in table - TxFree
Section 5.3.3.1	Replaced contents with new source
Section 5.3.3.2	Replaced contents with new source
Section 5.3.4	Replaced text in module - HwPktTxNext
Section 5.3.4	Replaced text in module - _HwPktPoll

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of TI information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation. Information of third parties may be subject to additional restrictions.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

TI products are not authorized for use in safety-critical applications (such as life support) where a failure of the TI product would reasonably be expected to cause severe personal injury or death, unless officers of the parties have executed an agreement specifically governing such use. Buyers represent that they have all necessary expertise in the safety and regulatory ramifications of their applications, and acknowledge and agree that they are solely responsible for all legal, regulatory and safety-related requirements concerning their products and any use of TI products in such safety-critical applications, notwithstanding any applications-related information or support that may be provided by TI. Further, Buyers must fully indemnify TI and its representatives against any damages arising out of the use of TI products in such safety-critical applications.

TI products are neither designed nor intended for use in military/aerospace applications or environments unless the TI products are specifically designated by TI as military-grade or "enhanced plastic." Only products designated by TI as military-grade meet military specifications. Buyers acknowledge and agree that any such use of TI products which TI has not designated as military-grade is solely at the Buyer's risk, and that they are solely responsible for compliance with all legal and regulatory requirements in connection with such use.

TI products are neither designed nor intended for use in automotive applications or environments unless the specific TI products are designated by TI as compliant with ISO/TS 16949 requirements. Buyers acknowledge and agree that, if they use any non-designated products in automotive applications, TI will not be responsible for any failure to meet such requirements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products

Amplifiers	amplifier.ti.com
Data Converters	dataconverter.ti.com
DSP	dsp.ti.com
Clocks and Timers	www.ti.com/clocks
Interface	interface.ti.com
Logic	logic.ti.com
Power Mgmt	power.ti.com
Microcontrollers	microcontroller.ti.com
RFID	www.ti-rfid.com
RF/IF and ZigBee® Solutions	www.ti.com/lprf

Applications

Audio	www.ti.com/audio
Automotive	www.ti.com/automotive
Broadband	www.ti.com/broadband
Digital Control	www.ti.com/digitalcontrol
Medical	www.ti.com/medical
Military	www.ti.com/military
Optical Networking	www.ti.com/opticalnetwork
Security	www.ti.com/security
Telephony	www.ti.com/telephony
Video & Imaging	www.ti.com/video
Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments, Post Office Box 655303, Dallas, Texas 75265
Copyright © 2008, Texas Instruments Incorporated