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

User's Guide

Literature Number: SPRUES4
January 2007

TEXAS INSTRUMENTS

Contents

List of Tables

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 —***TMS320C6000DSP 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 are trademarks of Texas Instruments.

User's Guide
SPRUES4–January 2007

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

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 Support Package
- `<example/network/cfgdemo/DSK6455>` CCStudio project files for cfgdemo example
- `<example/network/client/DSK6455>` CCStudio project files for client example
- `<example/network/helloWorld/DSK6455>` CCStudio project files for helloWorld example
- `<example/tools/DSK6455>` Common tools used by Support Package
- `<lib/hal/DSK6455>` Pre-built HAL libraries for DSK6455
- `<src/hal/DSK6455/eth_c6455>` Source code for C6455 EMAC driver
- `<src/hal/DSK6455/userled_c6455>` Source code for DSK6455 LED driver

1.3 Rebuilding HAL Libraries

Included with the Support Package is a new batch file that resides in the `<NDK_INSTALL_DIR>/packages/ti/ndk` directory. This batch file is called MAKEHAL_DSK6455.BAT.

Before using MAKEHAL_DSK6455 from a command prompt, the NDK batch file DOSRUN_BIOS.BAT must be run from the `<NDK_INSTALL_DIR>/packages/ti/ndk` to set up the correct environment for running the TI code generation tools from a command prompt. Make sure that the TI_DIR environment variable is set to point to your Code Composer Studio Development Tools installation.

The form of the MAKEHAL_DSK6455 command is:

```
makehal_DSK6455 [library] (noclean)
```

The library name is required in this command. The value of library determines what device library (or libraries) to build. The value of library can be either of the following:

- ETHERNET: Texas Instruments C6455 Ethernet MAC Driver
- USERLED: DSK6455 User LED Driver

The final parameter *noclean* can be added to the command line to suppress cleaning old object files from the target directory. This is only useful when rebuilding the same driver for the same platform baseboard.

The batch file does not perform stringent argument checking, so incorrect calling arguments may result in a incorrect build.

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: LLLED.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 the 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 full NDK Ethernet packet driver API is documented in the Appendix D of *TMS320C6000 Network Developer's Kit (NDK) Programmer's Reference Guide* (SPRU524).

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
	LLPACKET.C	Hardware independent portion of the Low-Level Ethernet Packet Driver
	LLPACKET.H	Private include file for LLPACKET 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
<SRC\HAL\DSK6455\ETH_C6455\CSLR>		
	C6455_COMMON.H	Common peripheral register structures and definitions
	CSLR.H	Central register layer -- contains field-manipulation macro definitions
	CSLR_DEV.H	Register Desciptions for DEV
	CSLR_ECTL.H	Register Desciptions for ECTL
	CSLR_EMAC.H	Register Desciptions for EMAC
	CSLR_MDIO.H	Register Desciptions for MDIO

5.2 Ethernet Driver

The NDK packet driver API is discussed in Appendix D of the *TMS320C6000 Network Developer's Kit (NDK) Programmer's Reference Guide* (SPRU524), 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: LLPACKET.C

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 whether to use the device independent LLPACKET.C module.

The standard API to access the packet device as defined in Appendix D of the *TMS320C6000 Network Developer's Kit (NDK) Programmer's Reference Guide* (SPRU524) is implemented by the LLPACKET.C module. This module can also 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 module calls 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 to 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) that implements the llPacket API described in the *TMS320C6000 Network Developer's Kit (NDK) Programmer's Reference Guide* (SPRU524), Appendix D, 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 llPacket 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 llPacket 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)
} 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 *STKEVENT_signal()* 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, 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.
Filter	Current Rx Filter. The receive filter determines how the packet device should filter incoming packets. This field is set by LLPACKET.C and used by the mini-driver to program the MAC. Legal values include: • 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 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 *HwPktTxNext()* 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 *HwPktTxNext()* function and posts them to the Ethernet MAC for transmit. Once the packet has been transmitted, the packet buffer is freed by calling *PBM_free()*.

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.C module. There is one RX queue for all Ethernet devices. 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.

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 will place 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 will call 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 — *Initialize Packet Driver Environment*

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 — *Shutdown Packet Driver Environment*

Syntax void HwPktShutdown();

Parameters None

Return Value None

Description Called to indicate that the packet driver environment should be completely shut down.

HwPktOpen — *Open Ethernet Packet Device Instance*

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 — *Close Ethernet Packet Device Instance*

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 *PBM_free()*.

HwPktTxNext — *Transmit Next Buffer in Transmit Queue*

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 believes the transmitter to be free (TxFree set to 1). The mini-driver uses this function to start the transmission sequence.

HwPktSetRx — *Set Ethernet Rx Filter*

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.

_HwPktPoll — *Mini-Driver Polling Function*

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 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).

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
Low Power Wireless	www.ti.com/lpw	Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless