DS3131DK
Bit-SynchronouS (BoSS) HDLC Controller Demo Kit

www.maxim-ic.com

GENERAL DESCRIPTION

The DS3131 bit-synchronous (BoSS) HDLC controller can handle up to 40 channels of high-speed, unchannelized, bit-synchronous HDLC. The on-board DMA has been optimized for maximum flexibility and PCI bus efficiency to minimize host processor intervention in the data path. Diagnostic loopbacks and an on-board BERT remove the need for external components.

APPLICATIONS

Routers
xDSL Access Multiplexers (DSLAMs)
Clear-Channel (unchannelized) T1/E1
Clear-Channel (unchannelized) T3/E3
SONET/SDH Path Overhead Termination
High-Density V.35 Terminations
High-Speed Links such as HSSI

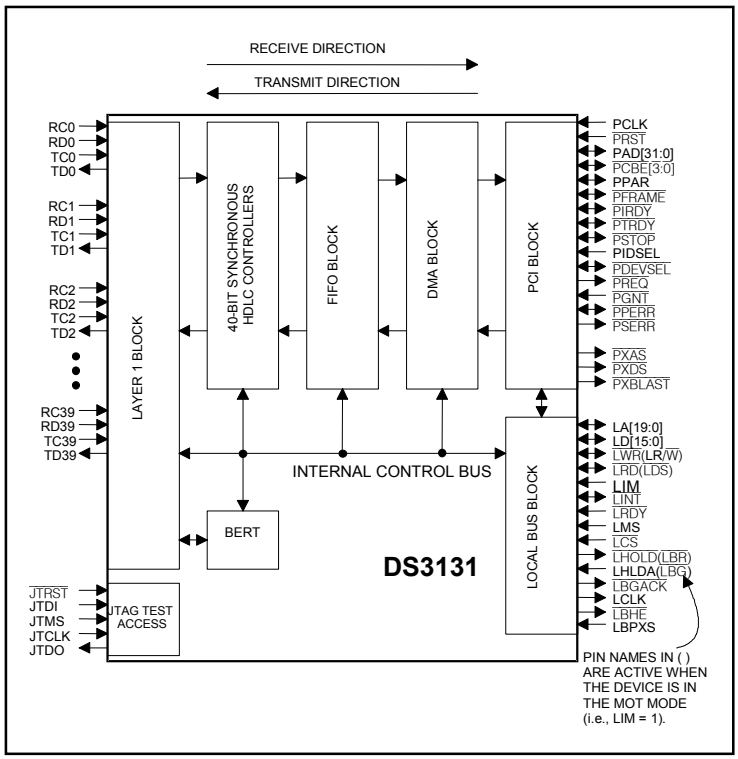
FEATURES

- 40 Timing Independent Ports
- 40 Bidirectional HDLC Channels
- Each Port Can Operate Up to 52Mbps
- Up to 132Mbps Full-Duplex Throughput
- On-Board Bit Error-Rate Tester (BERT)
- Diagnostic Loopbacks in Both Directions
- Local Bus Supports PCI Bridging
- 33MHz 32-Bit PCI Interface
- Full Suite of Driver Code

ORDERING INFORMATION

PART	TEMP RANGE	PIN-PACKAGE
DS3131	0°C to +70°C	272 PBGA

FUNCTIONAL DIAGRAM



__Note:__ Some revisions of this device may incorporate deviations from published specifications known as errata. Multiple revisions of any device may be simultaneously available through various sales channels. For information about device errata, click here: www.maxim-ic.com/errata.

TABLE OF CONTENTS

1. GENERAL OVERVIEW

The DS3131DK is a demonstration and evaluation kit for the DS3131 BoSS bit-synchronous HDLC controllers. The DS3131DK is intended to be used in a full-size PC platform, complete with PCI. The DS3131DK operates with a software suite that runs under Microsoft Windows®95/98/NT. The PC platform must be at least a 200MHz+ Pentium II class CPU with 32MB of RAM. Figure 1-1 details an outline of the PCI board for the DS3131DK.

The DS3131DK was designed to be as simple as possible but provides the flexibility to be used in a number of different configurations. The DS3131DK has all of the port pins and the local bus pins from the DS3131 that are easily accessible through headers on top of the card. A second DS3131DK can also be loaded into the PC in an adjacent PCI slot to add additional functions such as:

- Multiple T1/E1 framers
- T3 line interface
- HSSI interface
- V.35 interfaces

An Altera 9000 series PLD device is connected to all of the port pins on the DS3131. The PLD is capable of being loaded with various configurations through a programming port (J4) that resides on the DS3131DK. This PLD generates clocks and frame syncs as well as routes data from one port to another in a daisy-chain fashion to allow testing the device under worst-case loading (Figure 1-2). Two oscillators provide the port timing.

The transmit side of a port is derived from one clock and the receive side from another, so that they can be asynchronous to one another. If the PLD is not needed, it can be three-stated to remove it (electrically) from the board. Signals can then be sent to the DS3131 by the pin headers.

The board is intended to be a full-size PCI card that can only be plugged into a 5V PCI system environment. There is a 256-pin plastic BGA socket on the board for the DS3131.

Only the DS3131 is operated at 3.3V. Since it cannot be guaranteed that a 3.3V supply exists in a 5V PCI system environment, the DS3131DK has a linear regulator on it (U4: LT1086) to convert from 5V to 3.3V. All of the other logic, including the PLD and oscillators, operate at 5V. If 3.3V exists on the PCI bus, the linear regulator can be removed and a 0Ω jumper can be installed at R97 (Figure 1-1).

The JTAG pins on the DS3131 are not active on the DS3131DK. Therefore, the JTCLK, JTDI, and JTMS signals are wired to 3.3V and $\overline{\text{JTRST}}$ is wired low.

The DS3131DK was designed to use the device's 28-port mode rather than the 40-port mode, so the local bus can be used.

Windows is a registered trademark of Microsoft Corp.

Figure 1-1. PCI Card Configuration

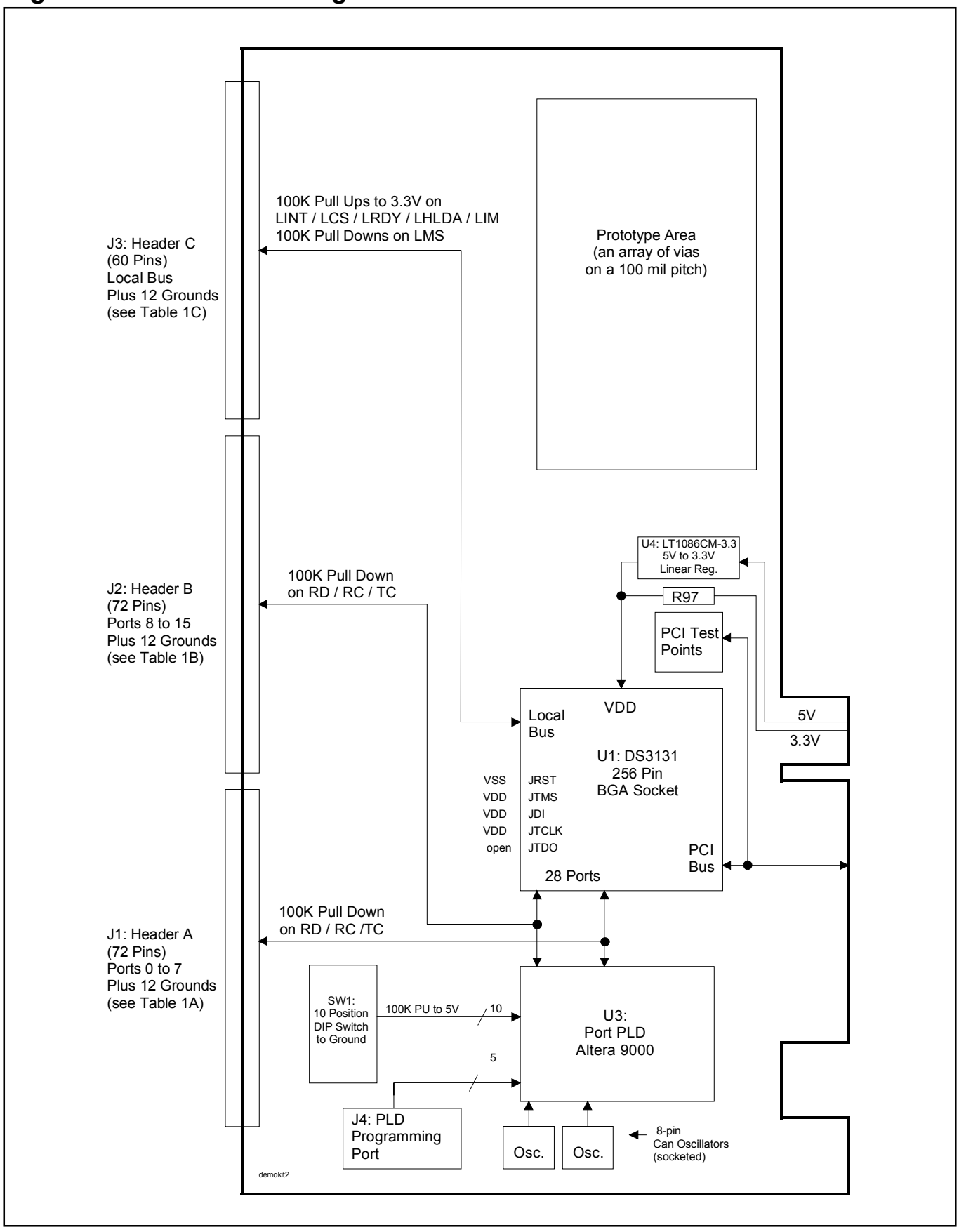
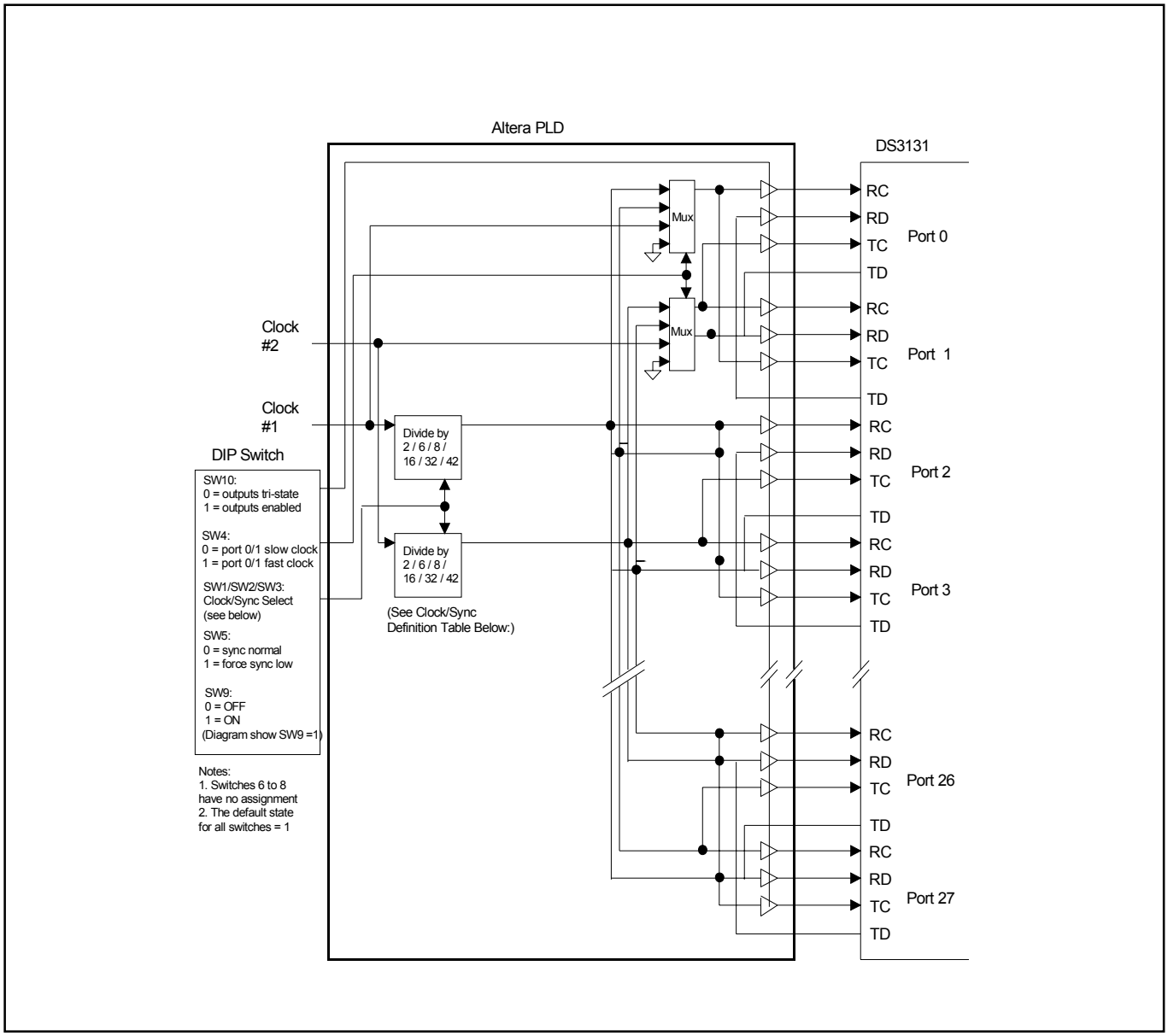


Figure 1-2. Port PLD Schematic



Clock/Sync Definitions

SW3	SW2	SW1	Clock Speed with OSC = 66MHz
0	0	0	66MHz / 2 = 33MHz
0	0	1	66MHz / 6 = 11.00MHz
0	1	0	66MHz / 8 = 8.25MHz
0	1	1	66MHz / 16 = 4.125MHz
1	0	0	66MHz / 32 = 2.0625MHz
1	0	1	66MHz / 42 = 1.572MHz

Note 1: Switch Open = Off = High (1)
Note 2: Switch Closed = On = Low (0)

Table 1-A. Header A Definition

1	RD0	2	TD0
3	RC0	4	TC0
5	RD1	6	TD1
7	RC1	8	TC1
9	GND	10	GND
11	RD2	12	TD2
13	RC2	14	TC2
15	RD3	16	TD3
17	RC3	18	TC3
19	GND	20	GND
21	RD4	22	TD4
23	RC4	24	TC4
25	RD5	26	TD5
27	RC5	28	TC5
29	GND	30	GND
31	RD6	32	TD6
33	RC6	34	TC6
35	RD7	36	TD7
37	RC7	38	TC7
39	GND	40	GND
41	RD8	42	TD8
43	RC8	44	TC8
45	RD9	46	TD9
47	RC9	48	TC9
49	GND	50	GND
51	RD10	52	TD10
53	RC10	54	TC10
55	RD11	56	TD11
57	RC11	58	TC11
59	GND	60	GND
61	RD12	62	TD12
63	RC12	64	TC12
65	RD13	66	TD13
67	RC13	68	TC13
69	GND	70	GND
71	GND	72	GND

Table 1-B. Header B Definition

1	RD14	2	TD14
3	RC14	4	TC14
5	RD15	6	TD15
7	RC15	8	TC15
9	GND	10	GND
11	RD16	12	TD16
13	RC16	14	TC16
15	RD17	16	TD17
17	RC17	18	TC17
19	GND	20	GND
21	RD18	22	TD18
23	RC18	24	TC18
25	RD19	26	TD19
27	RC19	28	TC19
29	GND	30	GND
31	RD20	32	TD20
33	RC20	34	TC20
35	RD21	36	TD21
37	RC21	38	TC21
39	GND	40	GND
41	RD22	42	TD22
43	RC22	44	TC22
45	RD23	46	TD23
47	RC23	48	TC23
49	GND	50	GND
51	RD24	52	TD24
53	RC24	54	TC24
55	RD25	56	TD25
57	RC25	58	TC25
59	GND	60	GND
61	RD26	62	TD26
63	RC26	64	TC26
65	RD27	66	TD27
67	RC27	68	TC27
69	GND	70	GND
71	GND	72	GND

Table 1-C. Header C Definition

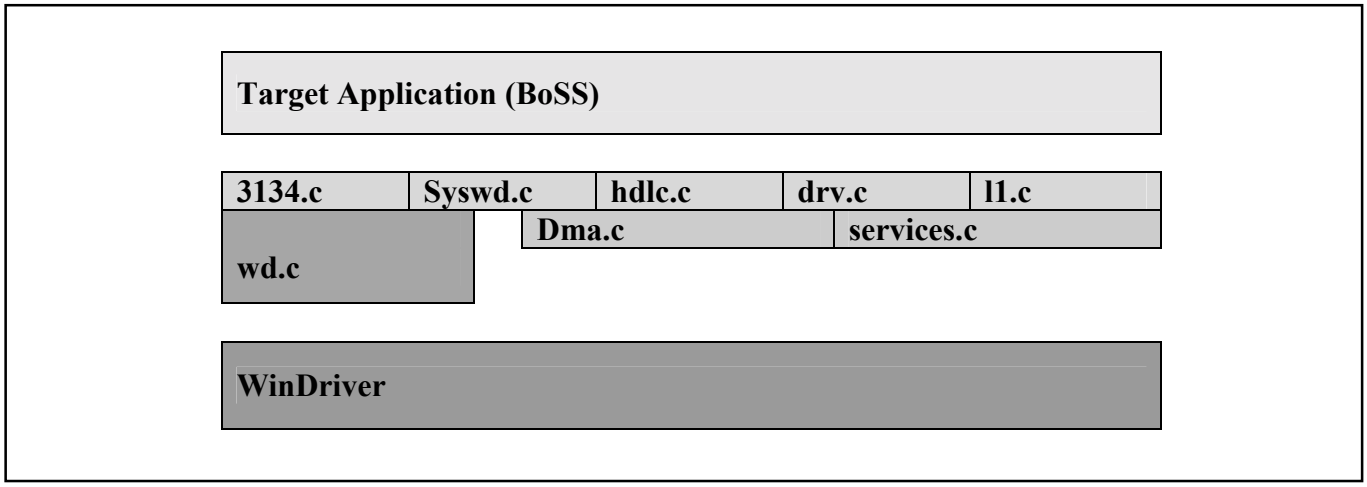
1	LD0	2	LD1
3	LD2	4	LD3
5	LD4	6	LD5
7	GND	8	GND
9	LD6	10	LD7
11	LD8	12	LD9
13	LD10	14	LD11
15	GND	16	GND
17	LD12	18	LD13
19	LD14	20	LD15
21	LIM	22	LMS
23	GND	24	GND
25	LHOLD	26	LHLDA
27	LBGACK	28	$\overline{\text{LINT}}$
29	$\overline{\text{LCS}}$	30	$\overline{\text{LRDY}}$
31	LCLK	32	LBHE
33	$\overline{\text{LWR}}$	34	$\overline{\text{LRD}}$
35	LA0	36	LA1
37	GND	38	GND
39	LA2	40	LA3
41	LA4	42	LA5
43	LA6	44	LA7
45	GND	46	GND
47	LA8	48	LA9
49	LA10	50	LA11
51	LA12	52	LA13
53	GND	54	GND
55	LA14	56	LA15
57	LA16	58	LA17
59	LA18	60	LA19

2. SOFTWARE

2.1 Architecture

The DS3131DK software consists of a high-level piece of reference software called "BoSS" that sits on top of a driver. This driver itself is composed of two discrete layers. The upper layer of the driver consists of various blocks of C code that are specific to the DS3131. These blocks contain an assortment of portable functions designed to serve as a low-level API for the BoSS. At the bottom level of the driver is the commercially available WinDriver, which interfaces with the Windows operating system to the DS3131DK's PCI hardware.

Figure 2-1. Software Architecture



2.2 Introduction to BoSS

The DS3131DK software (BoSS program) is written to run under a PC loaded with a Windows 95/98/NT operating system using the DS3131DK PCI card. The software includes two parts—the GUI interface (Figure 2-2) and driver code. It is developed by Visual C++ and using WinDriver to create the driver.

The software provides:

- a simple demonstration of the DS3131 with the ability to set the device into a number of different configurations
- software drivers for the DS3131
- the ability to explore and load new data into the BoSS registers
- a utility to dump the internal BoSS registers to a file and to load BoSS from a file
- user-configurable DMA parameters

The software does not implement all of the functions available in the DS3131. The user controls the software through a main GUI interface, as shown in Figure 2-2. The software implements 28 ports, coupled with 28 independent bidirectional HDLC channels. However, if a field in the main GUI is shaded gray, the function is not available.

HDLC Channel Assignment Table

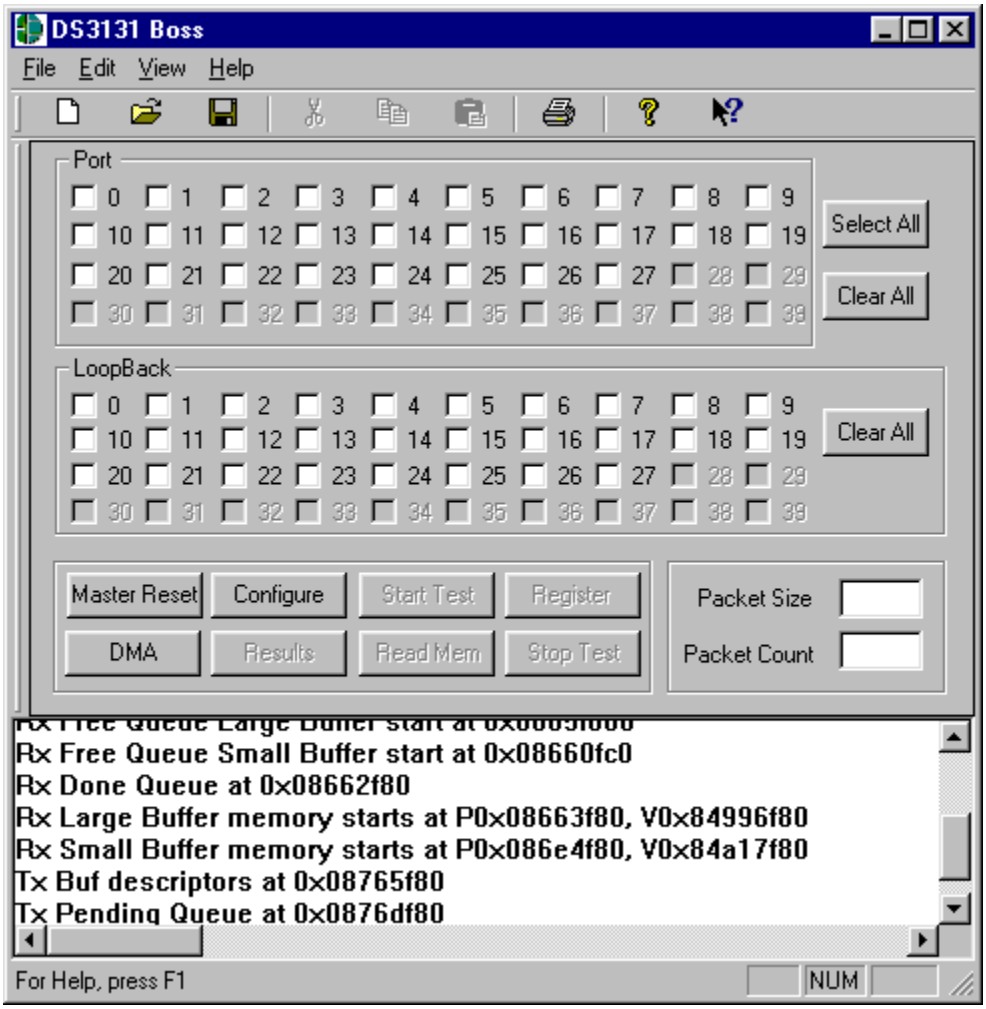
PORT NUMBER	HDLC CHANNEL NUMBER
0	1
1	2
: :	: :
26	27
27	28

When a test is run, the BoSS transmits data that is looped back to either the same port (if local loopback is used) or to an adjacent port (if the Altera PLD is used to loop the data). The software checks the receipt of packets to ensure they are received without error (i.e., the CRC is correct). For each HDLC channel that is enabled, the software also keeps track of the number of packets sent, number of packets received, number of packets received in error, and a variety of other statistics/counts.

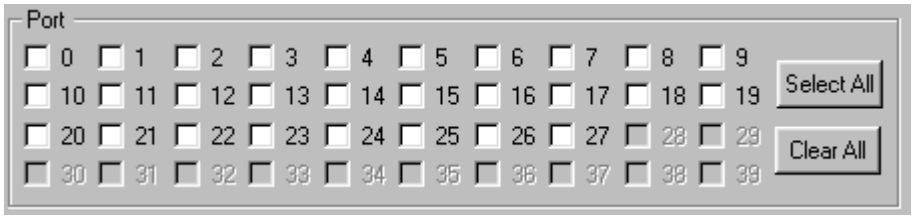
2.3 BoSS Software GUI Interface and Description

2.3.1 Main GUI Interface—Configuration

Figure 2-2. Software Main GUI



General Configuration

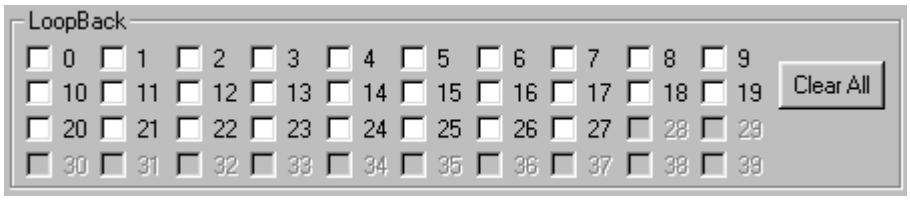


- The 28 ports on BoSS are handled through a set of 28 check boxes. The port number's box must be checked to be enabled. If this box is not checked, the software does not configure any of the RP[n]CR or TP[n]CR registers or any of the RH[n]CR] and TH[n]CR registers for that port.

- As a port is selected, the corresponding port's loopback check box is selected by default.

- All 28 ports and loopback are selected when this button is hit.

- All 28 ports and loopback are cleared (not selected) when this button is hit.



- The 28 ports on the BoSS are handled through a set of 28 check boxes. If the box is checked, then the software sets the LLBA bit (bit 10) in the RP[n]CR register to a 1, configuring the port in loopback mode. If this box is not checked, the software clears the LLBA bit.

- All 28 loopback check boxes are cleared when the button is hit.

- The user can input (in hex or decimal number) the desired packet size and packet count through the Packet Size and Packet Count edit boxes for the test. Both edit boxes default to 0x100 if the user does not input any entry.

Control Descriptions

Master Reset

- BoSS program sets the BoSS into a default state by issuing a software reset and then writing 0s into all indirect registers. The software also runs a register diagnostic to ensure it can correctly write and read all BoSS registers. The address of the buffer, descriptor, and queue are displayed in the message box when the process is done. If the diagnostic fails, the software creates an error message and displays it in the message box at the bottom of the main GUI interface.

Configure

- BoSS program loads the BoSS registers with the settings in the GUI interface when this button is hit. "Successfully configured port #" is displayed in the message box if successful.

Start Test

- The program transmits and receives packets based on the user's selections. "Test Done" is displayed in the message box when the test is done.

Stop Test

- The user can stop the test any time. "Test stopped by user" is displayed in the message box when the user hits this button.

Results

- This brings up a screen with detailed information about the packet results (Figure 2-3, Figure 2-4, and Figure 2-5).

Read Mem

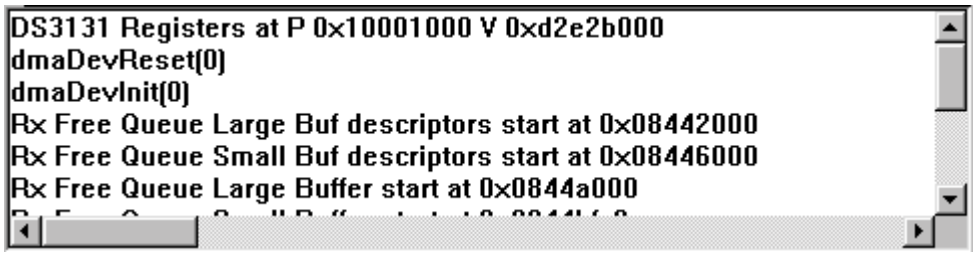
- This brings the Physical Memory Viewer screen up (Figure 2-6). The software prompts the user for an address, then dumps the next 32 dwords to the screen. The user manually cancels the screen. This button is only active when a test is not being run.

Register

- This brings up the BoSS Registers screen (Figure 2-7) for the user to read from or write into the register by hex number.

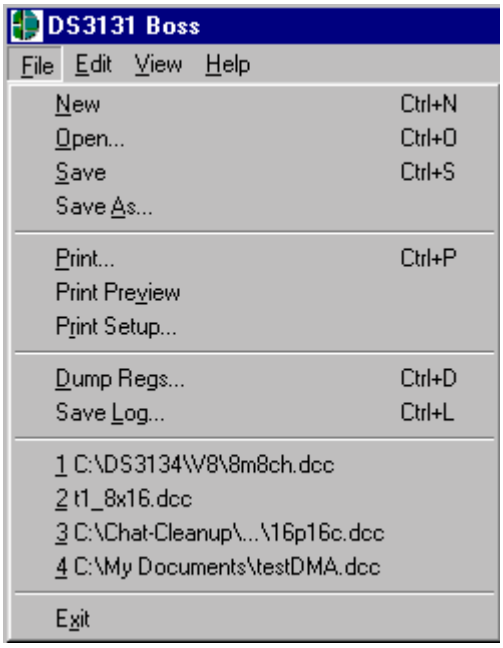
DMA

- This brings up the DMA Configuration screen (Figure 2-8) for the user to configure the DMA by their desired values. Otherwise, the software uses default values to configure the DMA.



- The message box at the bottom of the main GUI displays the status of the process.

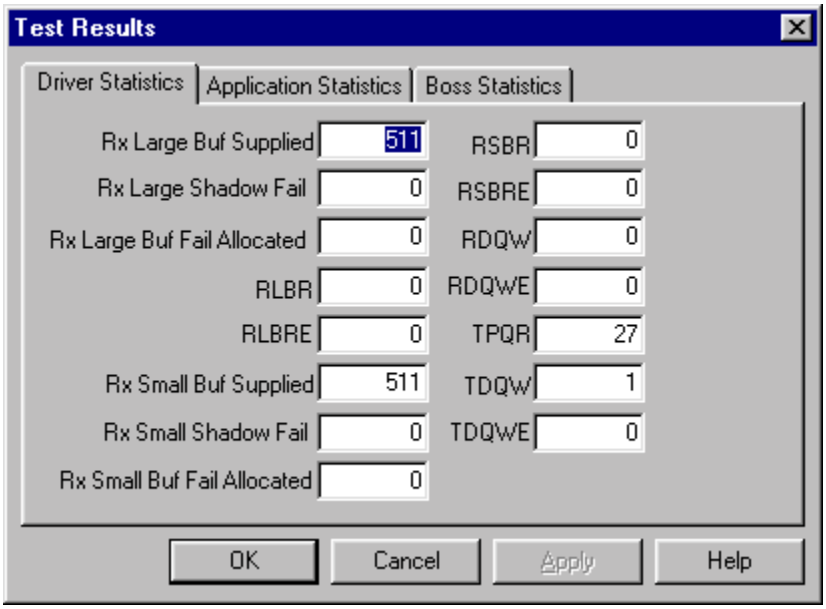
File Menu Descriptions



Open	All fields of the general configuration in the main GUI are filled from the file (that file should be saved by the software by Save under the File menu first).
Save	Copies all the selections from main GUI into a file when Save is selected.
Dump Regs	Saves the settings of all registers into a text file. The user can dump information at any time.
Save Log	Saves contents of the message box (at the bottom of the main GUI) into a text file.

2.3.2 Show Results

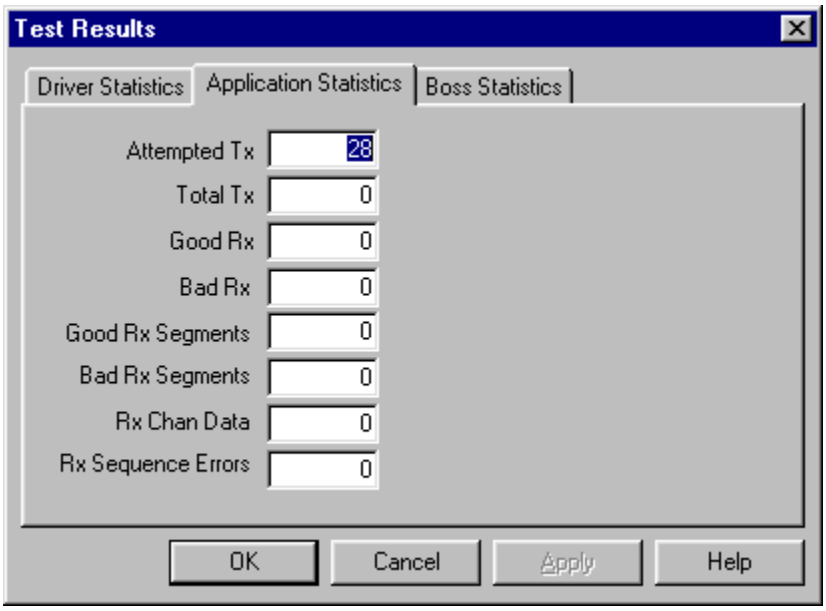
Figure 2-3. Show Results GUI (Driver Statistics)



Descriptions of Driver Statistics

Rx Large Buffer Supplied	Number of Rx large buffers used
Rx Large Shadow Failed	Number of Rx large shadow creation failures
Rx Large Buf Fail Allocated	Number of Rx large buffer allocation failures
RLBR	Read from SDMA Bit 6
RLBRE	Read from SDMA Bit 7
Rx Small Buffer Supplied	Number of Rx small buffers used
Rx Small Shadow Failed	Number of Rx small shadow creation failures
Rx Small Buf Fail Allocated	Number of Rx small buffers allocation failures
RSBR	Read from SDMA Bit 8
RSBRE	Read from SDMA Bit 9
RDQW	Read from SDMA Bit 10
RDQWE	Read from SDMA Bit 11
TPQR	Read from SDMA Bit 13
TDQW	Read from SDMA Bit 14
TDQWE	Read from SDMA Bit 15

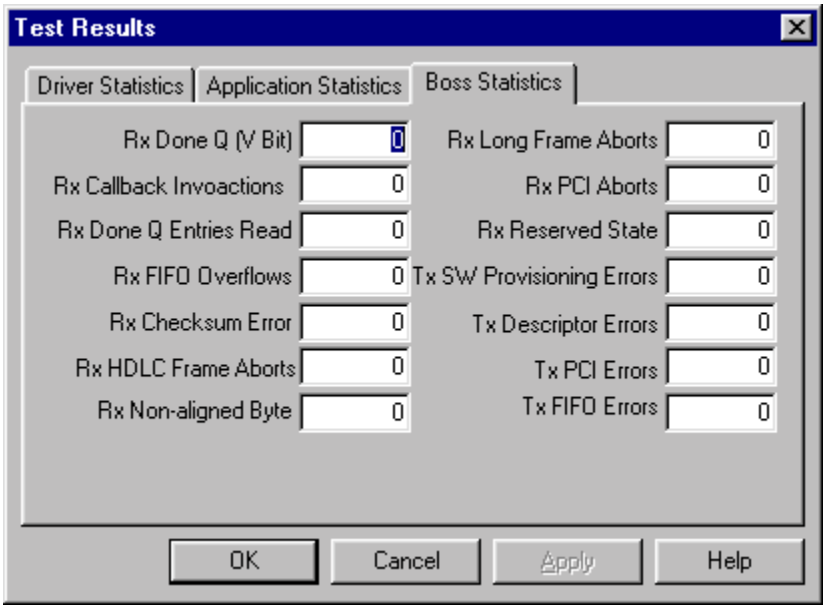
Figure 2-4. Show Results GUI (Application Statistics)



Descriptions of Application Statistics

Attempted Tx	Number of packet transmission attempted
Total Tx	Total number of packets transmitted
Good Rx	Number of packets received without error/problem
Bad Rx	Number of packets with errors
Good Rx Segments	For program debugging
Bad Rx Segments	For program debugging
Rx Chan Data Errors	Number of packets with an incorrect channel number
Rx Sequence Errors	Number of packets with an incorrect sequence number

Figure 2-5. Show Results GUI (BoSS Statistics)



Descriptions of BoSS Statistics

Rx Done Queue (V Bit)	Number of V set occurrences in receive done-queue descriptors
Rx Callback Invocations	For program debugging
Rx Done Queue Entries Read	Number of done-queue entries read

The following seven items are read from receive done-queue descriptor, dword0, bits 27–29, reported at the final status of an incoming packet:

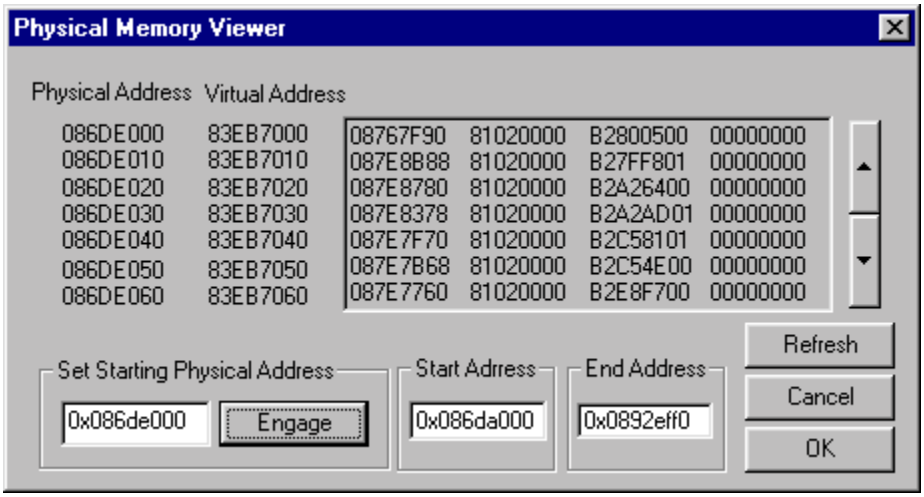
Rx FIFO Overflows	Remainder of the packet discarded
Rx Checksum Error	CRC checksum error
Rx Long Frame Aborts	Max packet length exceeded; remainder of the packet discarded
Rx HDLC Frame Aborts	HDLC frame abort sequence detected
Rx Non-Aligned Byte	Not an integral number of bytes
Rx PCI Aborts	PCI abort or parity data error
Rx Reserved State	Not a normal device operation event

The following four items are read from transmit done-queue descriptor, dword0, bits 26–28, reported at the final status of an outgoing packet:

Tx SW Provisioning Errors	Channel was not enabled.
Tx PCI Errors	PCI errors; abort
Tx Descriptor Errors	Either byte count = 0 or channel code inconsistent with pending queue
Tx FIFO Errors	Underflow events

2.3.3 Memory Viewer

Figure 2-6. Memory Viewer GUI



The physical memory viewer shows all the data within the start and end address space that is allocated by the BoSS program. The user can step through memory by the address box or the scroll up/down buttons on the right.

Fields Descriptions

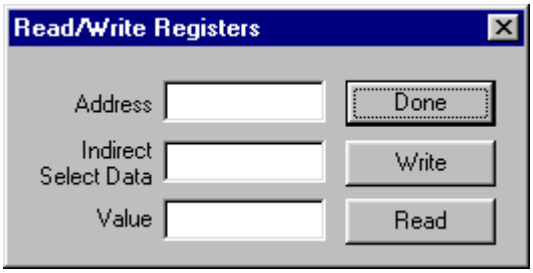
Start Address	Display the starting address of the DMA that the program allocated in the memory
End Address	Display the ending address of the DMA that the program allocated in the memory

Control Descriptions

Engage	User can look at any address within the range of Start Address and End Address through the edit box; (Input the desired physical address) and then hit the Engage button. All data displayed starts from the input physical address.

2.3.4 Register Access

Figure 2-7. Registers Access GUI



Field Descriptions

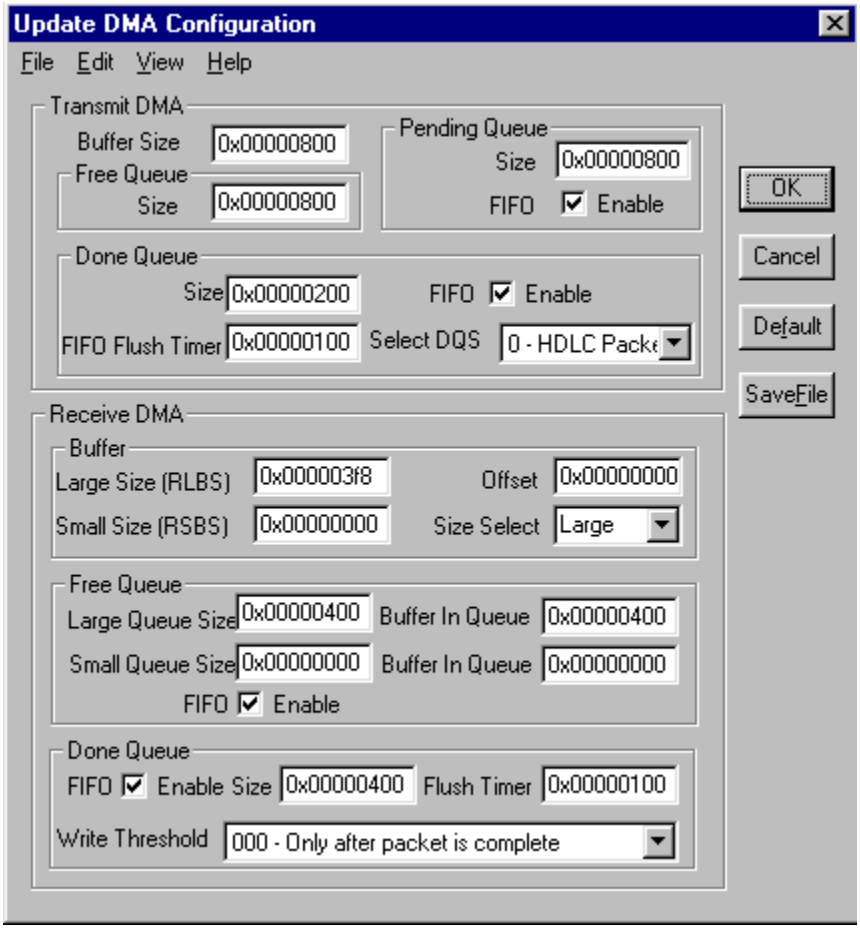
Register Address	Address of data register to read/write.
Indirect Select Data	If the address is the indirect select register, user needs to input the value.
Value	Display the value from the register when Read button is hit or specify the value to write into the register when Write button is hit.

Control Descriptions

Done	Close the screen.
Write	Prompt the user for which register address to write and the value to be written and then it writes to the register.
Read	Prompt the user for which register address to read and then it reads the register and return the value.

2.3.5 DMA Configuration

Figure 2-8. DMA Configuration GUI



The DMA configuration displays default values at first, then the user can change the desired value and hit the Master Reset. The DMA in BoSS can read from as well as write to the receive free queue and transmit pending queue. Therefore, each access of the descriptor queues are done one at a time, and sequentially.

Descriptions of Transmit DMA

Buffer Size	Size of the transmit buffer side; maximum value = 0x1fff
Free Queue Size	Number of free queues maximum value = 0xffff
Pending Queue	Size: Size of pending queue, maximum = 0x10000; FIFO: Enable/disable FIFO (TDMAQ bit 0)
Done Queue	Size: Size of done queue in transmit DMA, maximum value = 0x10000 FIFO: Enable/disable FIFO (TDMAQ bit 2) Flush Timer: TDQFFT, maximum value = 0xffff Select DQS: HDLC packet (transmit DMA configuration RAM dword1, bit 1)

Descriptions of Receive DMA

Buffer	Large: RLBS register; maximum value = 0x1fff Small: RSBS register; maximum value = 0x1fff Offset: Receive DMA configuration RAM, dword2, bits 3–6 Size Select: Receive DMA configuration RAM dword2, bits 1 and 2
Free Queue	Large: Size of free queue for large buffer Buffer in Queue: Number of buffers to put into the free queue Small: Size of free queue for large buffer Buffer in Queue: Number of buffers to put into the free queue Maximum value of the free queue = 0x10000 (large + small) FIFO: Enable/disable FIFO (RDMAQ, bit 0)
Done Queue	Size: Size of receive done queue, maximum value = 0x10000 FIFO: Enable/disable FIFO (RDMAQ, bit 4) Flush Timer: RDQFFT, maximum value = 0xffff Threshold: Receive DMA configuration RAM, dword2, bits 7–9

Control Descriptions

OK	The DMA settings are updated with the value from all fields.
SaveFile	Saves all the information from the GUI into a file.
Default	Restores all fields to BoSS default values.

2.4 Driver

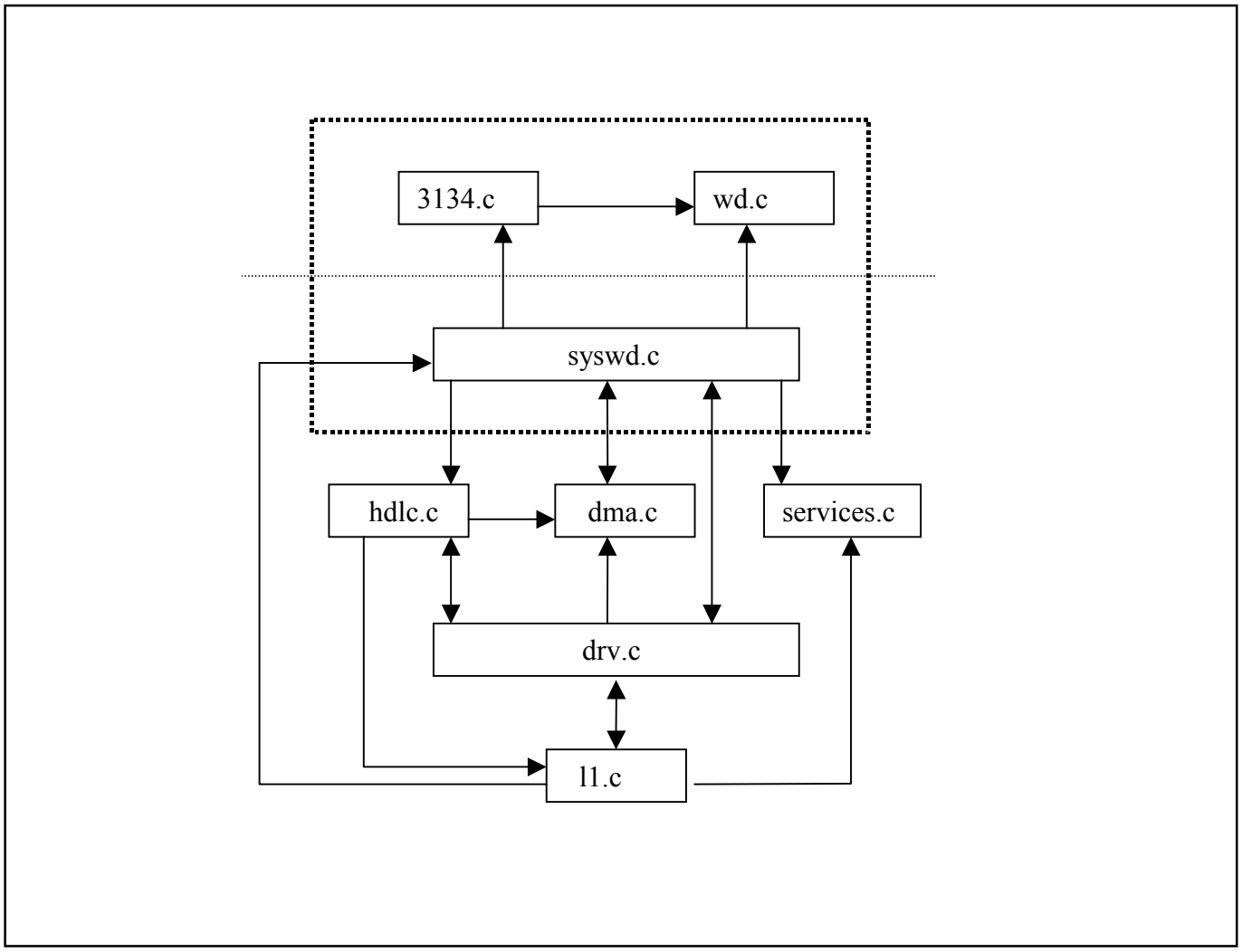
The low-level API, or driver, shown in layer 2 of Figure 2-1 may be used as a starting point in systems development to speed time to market. Low-level API source blocks are summarized in Table 2-A, and relate to one another structurally as shown in Figure 2-9. Note also in this figure a grouping of three particular source files: 3134.c, syswd.c, and wd.c. These are the files that must undergo modifications if the WinDriver package is not deployed in the target system.

Table 2-A. Low-Level API Source Block Contents

SOURCE FILE	CONTENT/PURPOSE
syswd.c	Interface code to WindRiver; system and memory management functions
hdlc.c	Channel management functions
l1.c	Port management and BERT functions
drv.c	Register management functions
services.c	Bit manipulation functions
3134.c	WinDriver-generated PCI management functions
wd.c	Generated WinDriver code

APPENDIX A contains reference tables listing all of the functions in each of the above code blocks. Note that some of the data structures are elaborate, and that they are defined in the header files. Usage examples can be found in the BoSS demonstration code.

Figure 2-9. Low-Level API Source Block Relationships



3. INSTALLATION AND GETTING STARTED

Please contact Telecom.Support@dalsemi.com or call 972-371-6555 if you have any technical questions, or visit our website at www.maxim-ic.com/telecom.

3.1 Card Installation

Separate instructions for Win95, Win98, and WinNT Systems.

3.1.1 Windows 95 Systems

1) Power-down the host computer system, and open its case. Follow ESD precautions while in contact with the card, the DS3131, and system components.
2) If not already seated, install the DS3131 chip into the BGA socket on the DK's PC board (Section 4).
3) Set the DIP switches on the card to configure the board and operational mode (Figure 1-2).
4) Plug the DS3131DK card into an empty PCI slot.
5) Reassemble the computer.
6) Boot the computer.
7) Insert the DS3131DK CD.
8) Open a DOS window to perform the following commands:
   - Change directory to c:\windows\system\vmm32.
   - Copy the file *windrvr.vxd* from the CD "Install\Win95" directory to c:\windows\system\vmm32.
   - Copy the file *wdreg.exe* from the CD "Install\Win95" directory to c:\windows\system\vmm32.
   - Run *wdreg -vxd install* from the DOS prompt in the current working directory.
   - Close the DOS shell and reboot the machine.

3.1.2 Windows 98 Systems

1) Power-down the host computer system, and open its case. Follow ESD precautions while in contact with the card, the DS3131, and system components.
2) If not already seated, install the DS3131 chip into the BGA socket on the DK's PC board (Section 4).
3) Set the DIP switches on the card to configure the board and operational mode (Figure 1-2).
4) Plug the DS3131DK card into an empty PCI slot.
5) Reassemble the computer.
6) Boot the computer and do not allow the system to search for or install drivers for the new hardware.
7) Insert the DS3131DK CD.
8) Open a DOS window to perform the following commands.:
   - Change directory to c:\windows\system\vmm32.
   - Copy the file *windrvr.sys* from the CD "Install\Win98" directory to c:\windows\system32\drivers.
   - Copy the file *wdreg.exe* from the CD "Install\Win98" directory to c:\windows\system32\drivers.
   - Run *wdreg install* from the DOS prompt in the current working directory.
   - Close the DOS shell and reboot the machine.

3.1.3 Windows NT Systems

1) Power-down the host computer system, and open its case. Follow ESD precautions while in contact with the card, the DS3131, and system components.
2) If not already seated, install the DS3131 chip into the BGA socket on the DK's PC board (Section 4).
3) Set the DIP switches on the card to configure the board and operational mode (Figure 1-2).
4) Plug the DS3131DK card into an empty PCI slot.
5) Reassemble the computer.
6) Boot the computer.
7) Insert the DS3131DK CD.
8) Open a DOS window to perform the following commands:
   - Change directory to c:\winnt\system.
   - Copy the file *windrvr.sys* from the CD "Install\WinNT" directory to c:\winnt\system32\drivers.
   - Copy the file *wdreg.exe* from the CD "Install\WinNT" directory to c:\winnt\system32\drivers.
   - Run *wdreg install* from the DOS prompt in the current working directory.
   - Close the DOS shell, and reboot the machine.

3.2 Software Installation

1) Make a directory on the system.
2) Copy BoSS.exe from the CD "Install\<OS_Type>" to the target directory.
3) Create a shortcut to the program, or set up a menu entry for it.

Note: The source code for BoSS and the underlying drivers is in the CD "Source" directory. If desired, the source directory can also be copied off of the CD to the host.

3.3 Operational Test

After performing the card and software installations as described above,

1) Ensure that the board's DIP switches are set as follows:

1	2	3	4	5	6	7	8	9	10
On	On	Off	On	On	Off	Off	Off	Off	Off

2) Execute the BoSS.exe program.
3) Click the ☐ On and ☐ LBK checkbox for Port 1.
4) Set both the Packet Size and Packet Count to 100.

5) Click the Master Reset button.
6) Click the Configure button. This results in a message stating "Successfully configured Port 1."
7) Click the Start Test button. The message "Starting test with 100 packets" appears. The message "Test Done" prints when complete.
8) Next, click the Results button. In the Application Statistics portion of the results window the following data (part of them) appear:

Atempted Tx	100
Total Tx	100
Good Tx	100
Bad Tx	0

4. PC BOARD LAYOUT

5. APPENDIX A

syswd.c **System Services (Generated Code; see WinDriver Developer's Guide)**

FUNCTION	PURPOSE	RETURNS
SysDevOpen	Open a particular card/device on PCI	int32
SysDevClose	Disable interrupts, unregister a card, and close the driver	int32
SysIntInit	Configure ISR	Nothing
SysCrash	System crash error handler	Nothing
SysFail	System failure handler	Nothing
SysRxBufAlloc	Allocate receive large buffer	drvRxBuf *
SysRxSmBufAlloc	Allocate receive small buffer	DrvRxBuf *
SysRxBufFree	Free receive large buffer	Nothing
SysRxSmBufFree	Free receive small buffer	Nothing
sysRxBufLastFree	Free the buffer in final	Nothing
sysTxBufAlloc	Allocate transmit buffer	drvTxBuf *
sysTxBufFree	Free transmit buffer	Nothing
sysDevWrReg16	Writes a word to an address space on the board	Nothing
sysDevRdReg16	Reads a word from an address space on the board	int32
sysIntDisable	Lock out interrupt thread	int32
sysIntEnable	Enable interrupt processing	Nothing
sysMemAlloc	Allocate virtual memory	void *
sysMemFree	Free virtual memory	Nothing
sysContAlloc	Allocate continuous memory block and map to phys. mem	void *
sysContFree	Release a continuous memory block	Nothing
locateAndOpenBoard	Locate the 3131 card on the PCI bus, open it, return handle	static DS3131_HANDLE
closeBoard	Close the 3131 board whose handle is passed	Nothing
sysIntHandler	Read SDMA register; then call ISR if it is not zero	Nothing
sysClearCrashMsg	Clear out the crash message buffer	Nothing
SysGetCrashMsg	System receive crash message	Nothing
sysGetVmemBase	Get virtual memory base address	unsigned long
sysGetPmemBase	Get physical memory base address	unsigned long
sysV2P	Convert virtual address to physical address	unsigned long
sysP2V	Convert physical address to virtual address	unsigned long

hdlc.c **HDLC Functions**

FUNCTION	PURPOSE	RETURNS
BosshdlcDevReset	Reset the device and its data (not called directly)	Nothing
hdlcDevOff	Turn the device off (not called directly)	Nothing
BosshdlcChanOpen	Open a channel with specified parameters	Nothing
hdlcChanClose	Close a channel	Nothing
BosshdlcChanGetState	Return the status of the channel	TRUE if open, FALSE otherwise
hdlcChanTrafficCtrl	Control a channel's traffic	Nothing

drv.c **Driver Level Functions**

FUNCTION	PURPOSE	RETURNS
drvGetVmemBase	Get virtual memory base address (calls to syswd.c)	unsigned long
drvGetPmemBase	Get physical memory base address (calls to syswd.c)	unsigned long
drvVAddr2Paddr	Convert virtual address to physical addr (calls to syswd.c)	unsigned long
drvPAddr2Vaddr	Convert physical address to virtual addr (calls to syswd.c)	unsigned long
drvWriteReg	Write a value into a device register	TRUE (success) FALSE (failure)
drvReadReg	Read a value from a device register	-1 on failure or the reg value on success
drvWriteIReg	Write a value into an indirect register	TRUE (success) FALSE (failure)
drvReadIReg	Read a value from an indirect register	-1 on failure or the int32 reg value on success
BossdrvDevInit	Reset and initialize the device	FALSE if device does not exist
drvDevOff	Put the device in reset	Nothing
drvWrIReg	Write to an indirect register	Nothing
drvRdIReg	Read from an indirect register	int32 Register value
BossdrvInitIRegs	Write a zero to an indirect register	Nothing
drvIntCallback	The interrupt callback from the ISR (only DMA int.s today)	Nothing
DrvGetIsrStats	Get a pointer to the interrupt service routine stats	drvIsrStats *
drvInitIRegs	Write a zero to all indirect registers of the device	Nothing
DrvGetdmaDesc	Get a pointer to the structure describing DMA configuration	DrvDmaDesc *
DrvUpdatedmaDesc	Get a pointer to the structure of DMA updated configuration	DrvDmaDesc *

L1.c Layer 1 Related Functions

FUNCTION	PURPOSE	RETURNS
Bossl1DevReset	Reset the device and its data (not called directly)	Nothing
l1DevOff	Turn Layer 1 port off (not called directly)	Nothing
Bossl1PortDisable	Disable a port	Nothing
Bossl1PortInit	Configure a port with static parameters	Nothing
Bossl1PortWriteDParam	Configure dynamic params of a port: copy from param	Nothing
Bossl1PortAllocateDs0	Set status specified by param, and HDLC channel number to all DS0s specified by tsMap and associate these ds0 with the port	Nothing
l1PortSetDs0Bits	Set status specified by nonzero bits of param to all DS0s specified by bitMap	Nothing
l1PortClearDs0Bits	Clear status specified by nonzero bits of param to all DS0s specified by bitMap	Nothing
l1PortFreeDs0	Disconnect the DS0 specified by bitMap from being associated with a port	Nothing
Bossl1PortReadStatus	Read port status and present it as a bitmap	Port status, int32 bitmap
l1BertWriteParam	Set miscellaneous BERT parameters	Nothing
l1BertSetParamBits	Set miscellaneous BERT parameters defined by nonzero bits of param	Nothing
l1BertClearParamBits	Set miscellaneous BERT parameters defined by nonzero bits of param	Nothing
l1BertLatchCounters	Get value of coutners into local storage and start a new count	Nothing
l1BertReadCounter	Read last latched value of counter from local storage	counter value (4 Bytes), uint32
l1BertSetPattern	Set pattern transmission	Nothing
l1BertSingleErrorInsertion	Insert single bit error	Nothing

dma.c	DMA Functions		
	FUNCTION	**PURPOSE**	**RETURNS**
	DmaDevReset	Reset the device and its data (not called directly)	Nothing
	DmaDevOff	Turn the device off	Nothing
	DmaDevInit	Initialize the device	TRUE/success, FALSE/not enough resources
	DmaDoRxReplenish	Give the Rx DMA as many receive large buffers as it can handle (not called directly)	Nothing
	MaDoSmRxReplenish	Give the Rx DMA as many receive small buffers as it can handle (not called directly)	Nothing
	DmaReplenishRxBuffers	Give the Rx DMA as many receive buffers as it can handle (not called directly)	Nothing(calls dmaDoRxReplinish)
	dmaCtrl	Enable or disable DMA	Nothing
	DmaChanSend	Submit packet chain to be transmitted	Nothing
	DmaRxChanCtrl	Set DMA RAM for the channel as appropriate	Nothing
	DmaTxChanCtrl	Set DMA RAM for the channel as appropriate	Nothing
	DmaEventRLBR	Rx large buffer read event, called by the ISR	Nothing
	DmaEventRLBRE	Rx large buffer read error event, called by the ISR	Nothing
	DmaEventRSBR	Rx small buffer read event, called by the ISR	Nothing
	DmaEventRSBRE	Rx small buffer read error event, called by the ISR	Nothing
	DmaEventRDQW	Rx done-queue write event, called by the ISR	Nothing
	DmaEventRDQWE	Rx done-queue write error event, called by the ISR	Nothing
	DmaEventTPQR	Tx pending-queue read event, called by the ISR	Nothing
	DmaEventTDQW	Tx done-queue write event, called by the ISR	Nothing
	DmaEventTDQWE	Tx done-queue write error event, called by the ISR	Nothing
	dmaTxPkt	Put a single packet into pending queue	TRUE if new pending Q element is reuqired
	DmaGetTxStats	Get a pointer to the Tx DMA stats	txDmaStats *
	DmaGetRxStats	Get a pointer to the Rx DMA stats	RxDmaStats *
	DrvGetdmaTxDev	Get a pointer to the structure of DMA Tx subsystem of the device	dmaTxDev *
	DrvGetdmaRxDev	Get a pointer to the structure of DMA Rx subsystem of the device	dmaRxDev *

services.c **General Services**

FUNCTION	PURPOSE	RETURNS
bitMapRead	Read the value of a specific bit	0 or 1, int32
bitMapWrite	Write the value of a specific bit	Nothing
bitMapLogicalAnd	Test for common set (=1) bits between two parameters	True if commonalities,else or False (int32)
bitMapLogicalEq	Test two parameters for equivalence	True if equal, else False (int32)
bitMapLogicalSubset	Test to see if parameter 2 is a subset of parameter 1	True if subset, else False (int32)
bitMapSetBits	Set all bits in parameter 1 that are set in parameter 2	Nothing
bitMapClearBits	Clear all bits in parameter 1 that are set in parameter 2	Nothing
BitMapIsEmpty	Test to see if any bits are set in parameter 1	True if all bits = 0, else False (int32)
BitMapSetRange	Set a range of bits in parameter 1	Nothing
BitMapClearRange	Clear a range of bits in parameter 1	Nothing

ds3134.c **DS3131 Card Access Functions (Generated Code; see WinDriver Developer's Guide, same as DS3134)**

FUNCTION	PURPOSE	RETURNS
DS3134_CountCards	Scan PCI and count the number of a certain type of card	Card count, DWORD
DS3134_Open	Open a particular card/device on PCI	True is succesful, else False
DS3134_Close	Disable interrupts, unregister a card, and close the driver	Nothing
DS3134_WritePCIReg	Write to a PCI configuration register	Nothing
DS3134_ReadPCIReg	Read from a PCI configuration register	Register value, DWORD
DS3134_DetectCardElements	Check availability of card info: interrupts, I/O, memory	True if all are found, else False
DS3134_IsAddrSpaceActive	Check if specified address space is active	True if active, else False
DS3134_ReadWriteBlock	Perform general block reads and writes	Nothing
DS3134_ReadByte	Reads a byte from an address space on the board	Byte
DS3134_ReadWord	Reads a word from an address space on the board	Word
DS3134_ReadDword	Reads a DWORD from an address space on the board	DWORD
DS3134_WriteByte	Writes a byte to an address space on the board	Nothing
DS3134_WriteWord	Writes a word to an address space on the board	Nothing
DS3134_WriteDword	Writes a DWORD to an address space on the board	Nothing
DS3134_GetRegAddrs	Get register address	Dword of 0 if found, else 1
DS3134_IntIsEnabled	Checks whether interrupts are enabled or not	True if enables, else False
DS3134_IntHandler	Configure interrupt event handling (indirectly called)	Nothing
DS3134_IntEnable	Enable interrupt processing	True is successfully configured, else False
DS3134_IntDisable	Disable interrupt processing	Nothing