# Application Note 385
# DS3131 Step by Step Configuration—Bridge Mode

www.maxim-ic.com

## OVERVIEW

This application note describes an example of how to configure a single port on the DS3131 operating in bridge mode. Additionally, this example describes how to construct, send, receive, and check a packet in loopback mode on that port. This application note is presented as a coding example for easy adaptation to end-user applications.

The local bus can operate in two modes in DS3131: PCI bridge mode and configuration bus mode.

The PCI bridge mode allows the host on the PCI bus to access the local bus. The PCI bus is used to control and monitor the DS3131 and transfer the packet data. The DS3131 maps data from the PCI bus to the local bus. (Refer to the DS3131 data sheet, Section 10.)

This example has the following configuration:

- Only port 1 and channel 1 of the DS3131 are assigned. All other ports are not used.
- HDLC channel 1 of the DS3131 is assigned four Rx FIFO blocks, four Tx FIFO blocks, an Rx FIFO high watermark of 3, and a Tx low watermark of 1.
- A 16-byte packet is constructed in host memory using one Tx buffer, one Tx descriptor, and one Tx pending queue entry. Since the DS3131 is in loopback mode, when the packet is transmitted it is also be received by the DS3131. The received packet is written to host memory using one Rx buffer, one Rx descriptor, and one Rx done queue entry.
- The host memory is configured as follows:

Receive Side:

| | | |
|---|---|---|
| Rx free queue base address (RFQBA1/0) | = | 0x10000000 |
| Rx done queue base address (RDQBA1/0) | = | 0x10000100 |
| Rx descriptor base address (RDBA1/0) | = | 0x10000200 |
| Rx buffer base address | = | 0x10001000 |

Transmit Side:

| | | |
|---|---|---|
| Tx pending queue base address (TPQBA1/0) | = | 0x10000300 |
| Tx done queue base address (TDQBA1/0) | = | 0x10000400 |
| Tx descriptor base address (TDBA1/0) | = | 0x10000500 |
| Tx buffer base address | = | 0x10002000 |

## Definition of the Coding Example Function Calls

In order to improve readability, the code in this example uses several function calls. The definitions of these functions are as follows:

- **write_reg(address, data)**

Write the specified data to the specified DS3131 register address.

Inputs:
Address = the register address where data is to be written
Data = the data to be written to the specified register
Outputs: None

- **read_reg(address, data)**

Read the contents of the DS3131 register at the specified address.

Inputs:
Address = the register address that is to be read
Outputs:
Data = the value read from the register

- **write_reg_IS(address, data)**

Write the specified data to the specified DS3131 indirect select register and then wait for that register's busy bit to clear before returning.

Inputs:
Address = the indirect select register where data is to be written
Data = the data to be written to the specified register
Outputs: None

Function Code:
```
write_reg(address, data);
bit_check = 0x8000;
while (bit_check & 0x8000)
        read_reg(address, bit_check);
```

- **wr_dword(address, data)**

Write the specified 32-bit data value to the specified 32-bit host memory address.

Inputs:
Address = the host memory address where data is to be written
Data = the data to be written to the specified memory address
Outputs: None

- **rd_dword(address, data)**

Read a 32-bit data value from the specified 32-bit host memory address.

Inputs:
Address = the host memory address which is to be read
Outputs:
Data = the 32-bit data value read from host memory

- **frame_wait(count)**

Provides a delay equal to count number of frame periods where a frame period is 125µs.

Inputs:
Count = number of frame periods to wait
Outputs: None

## Configuration Mode Coding Example

The code in this application note consists of the following steps:

1) Reset the DS3131
2) Configure the DS3131
3) Enable the HDLC channel
4) Put the HDLC channel in loopback mode
5) Queue, send, receive, and check a data packet

Each of these steps is detailed in the following sections by a brief description and coding example. Register names are used instead of addresses to improve readability. The corresponding address/offset of the DS3131 internal device configuration registers are listed in accompanying tables. Additionally, the abbreviations Tx and Rx are used to represent transmit side and receive side, respectively. Refer to the DS3131 data sheet for more detailed information.

## 1. Reset the DS3131

A software reset can be performed on all registers in the DS3131 using the master reset register (MRID). All internal registers are set to a default value of 0 when bit 0 of the MRID register is set to 1.The host must set this bit back to 0 before the device can be programmed for normal operation. Make sure the DMA is off on transmit and receive side through the channel enable bit in transmit and receive RAM.

| OFFSET/ADDRESS | ACRONYM | REGISTER NAME | DATA SHEET SECTION |
|---|---|---|---|
| 0000 | MRID | Master Reset and ID Register | 4.1 |

```
// Reset the DS3131 using MRID registers master reset bit.
write_reg(MRID, 0x0001);
write_reg(MRID, 0x0000);
```

| OFFSET/ADDRESS | ACRONYM | REGISTER NAME | DATA S HEET SECTION |
|---|---|---|---|
| 0770 | RDMACIS | Receive DMA Configuration Indirect Select | 8.1.5 |
| 0774 | RDMAC | Receive DMA Configuration | 8.1.5 |

```
// Disable the RX DMA configuration RAM
write_reg(RDMAC, 0x0000);
for(channel=0; channel<40; channel=channel+1)
  write_reg_IS(RDMACIS, 0x0400 + channel);
```

| OFFSET/ADDRESS | ACRONYM | REGISTER NAME | DATA SHEET SECTION |
|---|---|---|---|
| 0870 | TDMACIS | Transmit DMA Configuration Indirect Select | 8.2.5 |
| 0874 | TDMAC | Transmit DMA Configuration | 8.2.5 |

```
// Disable the TX DMA configuration RAM
write_reg(TDMAC, 0x0000);
for(channel=0; channel<40; channel=channel+1)
  write_reg_IS(TDMACIS, 0x0400 + channel);
```

## 2. Configure the DS3131

Configuration of the DS3131 consists of the following steps:

1) Configure PCI registers
2) Configure the DMA registers
3) Configure the FIFO registers
4) Configure the Port registers (Layer 1)
5) Configure the HDLC registers (Layer 2)

Configuration of each of these register sets is detailed in the following sections. Several variables are used to improve readability and provide a more algorithmic code structure. The following code provides the initialization of these variables:

```
// This example uses port 1 channel 1
port = 1;
channel = 1;

// RX free queue base address
rfq_base_addr = 0x10000000;

// RX free queue end address
// RX free queue size = 16
rfq_end_idx = 0x0010;

// RX done queue base address
rdq_base_addr = 0x10000100;

// RX done queue end address
// RX done queue size = 16
rdq_end_idx = 0x0010;

// RX descriptor base address
// RX descriptor table size = 256
rdscr_base_addr = 0x10000200;

// RX data buffer base address
rx_buf_base_addr = 0x10001000;

// TX pending queue base address
tpq_base_addr = 0x10000300;

// TX pending queue end address
// TX pending queue size = 16
tpq_end_idx = 0x0010;

// TX done queue base address
tdq_base_addr = 0x10000400;

// TX done queue end address
// TX done queue size = 16
tdq_end_idx = 0x0010;

// TX descriptor base address
// TX descriptor table size = 256
tdscr_base_addr = 0x10000500;

// TX data buffer base address
tx_buf_base_addr = 0x10002000;
```

## Configure the PCI Registers

In bridge mode, the PCI registers control how the DS3131 interfaces to the PCI bus when performing DMA operations. PCI register configuration is system dependent and therefore the coding example below may need to be modified to support a particular user application.

| OFFSET/ADDRESS | ACRONYM | REGISTER NAME | DATA SHEET SECTION |
|---|---|---|---|
| 0x004/0A04 | PCMD0 | PCI Command Status 0 | 9.2 |
| 0x010/0A10 | PDCM | PCI Device Configuration Memory Base Address | 9.2 |

```
// Map DS3131 Configuration Registers to a PCI bus base address
write_reg(PDCM, 32'h80000000);

// PCI command/status register 0 - controls DS3131 DMA functionality
// Set Bit 1 = 1 to enable accesses to internal device configuration registers
through PCI
// bus (required for Bridge mode)
// Set Bit 2 = 1 to allow the device operation as bus master on PCI bus
// (required for DMA)
// Set Bit 6 = 1 to act on parity errors
// Set Bit 8 = 1 to enable the PSERR pin
write_reg(PCMD0, 32'h00000146);
```

## Configure the DMA Registers

The DMA block handles the transfer of packet data from the FIFO block to the PCI block and vice versa. The PCI block controls data transfers between the DS3131 and the external PCI bus. The host, defined as the CPU or intelligent controller that sits on the PCI bus, instructs the DS3131 about how to handle the incoming and outgoing data. This is accomplished using descriptors that are defined as preformatted messages passed from the host to the DMA block or vice versa. Using these descriptors, the host informs the DMA about the location and status of packet data to be transmitted, and where to place packet data that is received. The DMA uses these descriptors to tell the host the status of packet data that has been transmitted, and the status and location of packet data that has been received.

On the receive side the host writes to the free queue descriptors informing the DMA where it can place incoming packet data. Associated with each free queue entry is a receive data buffer location and packet descriptor. As the DS3131 uses receive free queue entries to write received packet data to host memory, it creates entries in the Rx done queue. These Rx done queue entries inform the host about the location and status of received data. Refer to the DS3131 data sheet for more detailed information. The host must configure the Rx DMA by writing to all the of the registers at the following table:

| OFFSET/ADDRESS | ACRONYM | REGISTER NAME | DATA SHEET SECTION |
|---|---|---|---|
| 0700 | RFQBA0 | Receive Free Queue Base Address 0 (lower word) | 8.1.3 |
| 0704 | RFQBA1 | Receive Free Queue Base Address 1 (upper word) | 8.1.3 |
| 0708 | RFQEA | Receive Free Queue end Address | 8.1.3 |
| 070C | RFQSBSA | Receive Free Small Buffer Start Address | 8.1.3 |
| 0710 | RFQLBWP | Receive Free Queue Large Buffer Host Write Pointer | 8.1.3 |
| 0714 | RFQSBWP | Receive Free Queue Small Buffer Host Write Pointer | 8.1.3 |
| 0718 | RFQLBRP | Receive Free Queue Large Buffer DMA Read Pointer | 8.1.3 |
| 071C | RFQSBRP | Receive Free Queue Small Buffer DMA Read Pointer | 8.1.3 |
| 0730 | RDQBA0 | Receive Done Queue Base Address 0 (lower word) | 8.1.4 |
| 0734 | RDQBA1 | Receive Done Queue Base Address 1 (upper word) | 8.1.4 |
| 0738 | RDQEA | Receive Done Queue end Address | 8.1.4 |
| 073C | RDQRP | Receive Done Queue Host Read Pointer | 8.1.4 |
| 0740 | RDQWP | Receive Done Queue DMA Write Pointer | 8.1.4 |
| 0750 | RDBA0 | Receive Descriptor Base Address 0 (lower word) | 8.1.2 |
| 0754 | RDBA1 | Receive Descriptor Base Address 1 (upper word) | 8.1.2 |
| 0770 | RDMACIS | Receive DMA Configuration Indirect Select | 8.1.5 |
| 0774 | RDMAC | Receive DMA Configuration | 8.1.5 |
| 0790 | RLBS | Receive Large Buffer Size | 8.1.1 |

```
// RX large buffer size = 256 bytes
write_reg(RLBS, 0x0100);
// RX free queue base address
write_reg(RFQBA0, rfq_base_addr & 0x0000FFFF);
write_reg(RFQBA1, (rfq_base_addr >> 16) & 0x0000FFFF);

// RX free queue large buffer read and write pointers = 0
write_reg(RFQLBRP, 0x0000);
write_reg(RFQLBWP, 0x0000);

// RX free queue small buffer start address = 16
write_reg(RFQSBSA, rfq_end_idx);

// RX free queue small buffer read and write pointers = 0
write_reg(RFQSBRP, 0x0000);
write_reg(RFQSBWP, 0x0000);

// RX free queue end address
write_reg(RFQEA, rfq_end_idx);

// RX done queue base address
write_reg(RDQBA0, rdq_base_addr & 0x0000FFFF);
write_reg(RDQBA1, (rdq_base_addr >> 16) & 0x0000FFFF);

// RX done queue read and write pointers = 0
write_reg(RDQRP, 0x0000);
write_reg(RDQWP, 0x0000);

// RX done queue end address
write_reg(RDQEA, rdq_end_idx);

// RX descriptor base address
write_reg(RDBA0, rdscr_base_addr & 0x0000FFFF);
write_reg(RDBA1, (rdscr_base_addr >> 16) & 0x0000FFFF);

// RX DMA Channel Configuration
```

```
// The data in RDMAC register is written to or read from the Receive Configuration
// RAM
// Set bit 0 = 0 to disable the HDLC Channel
// Set bit 2-1 = 00 for large buffers only
// Set bit 6-3 = 0000 for 0 byte offset from the data buffer address of the first
data buffer
// Set Bit 9-7 = 000 for DMA write to the Done Queue only after packet reception is
// complete
// Set the HDLC Channel Number by RDMACIS register
write_reg(RDMAC, 0x0000);
write_reg_IS(RDMACIS, 0x0400 + channel);
```

On the transmit side, the host writes to the pending queue informing the DMA which channels have packet data that is ready to be transmitted. Associated with each pending queue descriptor is a linked list of one or more transmit packet descriptors that describe the packet data. Each of these transmit packet descriptors also has a pointer to a transmit data buffer that contains the actual data payload of the HDLC packet. As the DS3131 processes transmit pending queue descriptor entries, it creates transmit done queue descriptor queue entries. The DMA writes to the done queue when it has completed transmitting either a complete packet or data buffer depending on how the DS3131 is configured. Using these done queue descriptors, the DMA informs the host about the status of the outgoing packet data. Refer to the DS3131 data sheet for more detailed information. The host must configure the Tx DMA by writing to all the of the registers at the following table:

| OFFSET/ADDRESS | ACRONYM | REGISTER NAME | DATA SHEET SECTION |
|---|---|---|---|
| 0800 | TPQBA0 | Transmit Pending Queue Base Address 0 (lower word) | 8.2.3 |
| 0804 | TPQBA1 | Transmit Pending Queue Base Address 1 (upper word) | 8.2.3 |
| 0808 | TPQEA | Transmit Pending Queue end Address | 8.2.3 |
| 080C | TPQWP | Transmit Pending Queue Host Write Pointer | 8.2.3 |
| 0810 | TPQRP | Transmit Pending Queue DMA Read Pointer | 8.2.3 |
| 0830 | TDQBA0 | Transmit Done Queue Base Address 0 (lower word) | 8.2.4 |
| 0834 | TDQBA1 | Transmit Done Queue Base Address 1 (upper word) | 8.2.4 |
| 0838 | TDQEA | Transmit Done Queue end Address | 8.2.4 |
| 083C | TDQRP | Transmit Done Queue Host Read Pointer | 8.2.4 |
| 0840 | TDQWP | Transmit Done Queue DMA Write Pointer | 8.2.4 |
| 0850 | TDBA0 | Transmit Descriptor Base Address 0 (lower word) | 8.2.2 |
| 0854 | TDBA1 | Transmit Descriptor Base Address 1 (upper word) | 8.2.2 |
| 0870 | TDMACIS | Transmit DMA Configuration Indirect Select | 8.2.5 |
| 0874 | TDMAC | Transmit DMA Configuration | 8.2.5 |

```
// TX pending queue base address
write_reg(TPQBA0, tpq_base_addr & 0x0000FFFF);
write_reg(TPQBA1, (tpq_base_addr >> 16) & 0x0000FFFF);

// TX pending queue read and write pointers = 0
write_reg(TPQRP, 0x0000);
write_reg(TPQWP, 0x0000);

// TX pending queue end address
write_reg(TPQEA, tpq_end_idx);

// TX done queue base address
write_reg(TDQBA0, tdq_base_addr & 0x0000FFFF);
write_reg(TDQBA1, (tdq_base_addr >> 16) & 0x0000FFFF);

// TX done queue read and write pointers = 0
write_reg(TDQRP, 0x0000);
write_reg(TDQWP, 0x0000);
// TX done queue end address
write_reg(TDQEA, tdq_end_idx);
// TX descriptor base address
write_reg(TDBA0, tdscr_base_addr & 0x0000FFFF);
write_reg(TDBA1, (tdscr_base_addr >> 16) & 0x0000FFFF);

// TX DMA Channel Configuration
// The data in TDMAC register is written to or read from the Receive Configuration
RAM
// Set bit 0 = 0 to disable HDLC Channel
// Set bit 1 = 0 for write done queue after packet transmitted
// Set the HDLC Channel Number by TDMACIS register
write_reg(TDMAC, 0x0000);
write_reg_IS(TDMACIS, 0x0200 + channel);
```

### Configure the FIFO Registers

The DS3131 contains an 8kB transmit FIFO and an 8kB receive FIFO. Each FIFO is divided into 512 blocks of 4 dwords, or 16 bytes. FIFO memory is allocated on an HDLC channel basis. The amount of FIFO memory allocated to each HDLC channel is programmable and can be a minimum of 4 blocks and a maximum of 512 blocks. FIFO memory is allocated to HDLC channels by creating a circular linked list out of a group of blocks where each block points to the next block in the chain and the last block points to the first. The FIFO block linked list is assigned to a specific HDLC channel by assigning one block in the linked list to be that channel's FIFO starting block pointer.

The calculation of number of block, high watermark, and low watermark:

Blocks = 512 / number of port uses
High Watermark = Block / 2
Low Watermark = Block / 2

In this example, four Tx FIFO blocks and four Rx FIFO blocks are assigned to the HDLC channel. This example also uses an Rx FIFO high watermark of 3 and Tx FIFO low watermark of 1. The Rx FIFO high watermark indicates how many blocks should be written into Rx FIFO by the HDLC engines before the DMA begins sending the data to the PCI bus. The high watermark setting must be between one block and one less than the number of blocks in the link-list chain for the particular channel involved. The Tx FIFO low watermark indicates how many blocks should be left in the Tx FIFO before the DMA should begin getting more data from the PCI bus. The amount of FIFO memory, Rx FIFO high watermark, and Tx

FIFO low watermark required by an HDLC channel to prevent transmit underflows and receive overflows from occurring is application dependent. The Tx FIFO and Rx FIFO of the DS3131 are configured independently on an HDLC channel basis through the registers listed in the following tables.

| OFFSET/ADDRESS | ACRONYM | REGISTER NAME | DATA SHEET SECTION |
|---|---|---|---|
| 0910 | RFBPIS | Receive FIFO Block Pointer Indirect Select | 7.2 |
| 0914 | RFBP | Receive FIFO Block Pointer | 7.2 |

```
// Build the RX FIFO block linked list
//   0->1->2->3->0
for (block=0; block<4; block=block+1)
{
// Bits 9-0 in RFBP register indicate which block is next in the
// linked list
write_reg(RFBP, block+1);
write_reg_IS(RFBPIS, block);
}

// The last block points to the first block to create a circular linked list
write_reg(RFBP, 0x0000);
write_reg_IS(RFBPIS, 0x0003);

// Assign the circular linked list to a specific channel
write_reg(RFSBP, 0x0000);
write_reg_IS(RFSBPIS, channel);
```

| OFFSET/ADDRESS | ACRONYM | REGISTER NAME | DATA SHEET SECTION |
|---|---|---|---|
| 0920 | RFHWMIS | Receive FIFO High Watermark Indirect Select | 7.2 |
| 0924 | RFHWM | Receive FIFO High Watermark | 7.2 |

```
// Set RX FIFO high watermark for channel to 3
write_reg(RFHWM, 0x0003);
write_reg_IS(RFHWMIS, channel);
```

| OFFSET/ADDRESS | ACRONYM | REGISTER NAME | DATA SHEET SECTION |
|---|---|---|---|
| 0990 | TFBPIS | Transmit FIFO Block Pointer Indirect Select | 7.2 |
| 0994 | TFBP | Transmit FIFO Block Pointer | 7.2 |

```
// TX FIFO block linked list
//   0->1->2->3->0
for (block=0; block<3; block=block+1)
{
// Bits 9-0 in RFBP register indicate which block is next in the linked list
write_reg(TFBP, block+1);
write_reg_IS(TFBPIS, block);
}

// The last block points to the first block to create a circular linked list
write_reg(TFBP, 0x0000);
write_reg_IS(TFBPIS, 0x0003);
```

| OFFSET/ADDRESS | ACRONYM | REGISTER NAME | DATA SHEET SECTION |
|---|---|---|---|
| 0980 | TFSBPIS | Transmit FIFO Starting Block Pointer Indirect Select | 7.2 |
| 0984 | TFSBP | Transmit FIFO Starting Block Pointer | 7.2 |

```
// Assign the circular linked list to a specific channel
write_reg(TFSBP, 0x0000);
write_reg_IS(TFSBPIS, channel);

// Set TX FIFO low watermark for channel to 1
write_reg(TFLWM, 0x0001);
write_reg_IS(TFLWMIS, channel);
```

## Configure Port Registers (Layer 1)

Each port of the DS3131 contains a Layer 1 controller that performs several functions including:

- Assigning the HDLC channel number to the incoming and outgoing data
- Routing data to and from the BERT function

Layer 1 configuration is performed on a port basis by the RP[n]CR, TP[n]CR, registers where n is the port to be configured.

| OFFSET/ADDRESS | ACRONYM | REGISTER NAME | DATA SHEET SECTION |
|---|---|---|---|
| 01xx | RP[n]CR | Receive Port n Control Register | 5.2 |
| 02xx | TP[n]CR | Transmit Port n Control Register | 5.2 |

```
// Set RX Port Control Register
// Set bits 1-0 = 00 for clock and data are not inverted
// Set bit 10 = 0 to disable local loopback A
write_reg(RP0CR + 4*port, 0x0000);

// Set TX Port Control Register
// Set bit 1-0 = 00 for clock and data are not inverted
// Set bit 3 = 0 to force all data at TD to be 1 (TFDA1)
write_reg(TP0CR + 4*port, 0x0000);
```

## Configure HDLC Registers (Layer 2)

The DS3131 contains a 40 HDLC controller, which performs the Layer 2 functions. Functions performed by this controller include:

- Zero stuffing and de-stuffing
- Flag detection and byte alignment
- CRC generation and checking
- Data inversion and bit flipping

The HDLC controller is configured on a channel basis by the RH[n]CD and TH[n]CD registers.

| OFFSET/ADDRESS | ACRONYM | REGISTER NAME | DATA SHEET SECTION |
|---|---|---|---|
| 03xx | RH[n]CR | Receive HDLC n Control Register | 6.2 |
| 04xx | TH[n]CR | Transmit HDLC n Control Register | 6.2 |

```
// RX HDLC configuration
// Set bits 3-2 = 10 for 32-bit CRC
write_reg(RH[n]CR, 0x0008);

// TX HDLC configuration
// Set bit 1= 0 to select an interfill byte of 7E
// Set bits 3-2 = 10 for 32-bit CRC
// Set bits 11-8 = 1000 for closing flag/no interfill bytes/opening flag
write_reg(TH[n]CR, 0x0108);
```

## 3. Enable the HDLC Channel

After the DS3131 has been initialized the next step is to enable the HDLC channel. In addition to the configuration steps already described, the following steps must be performed to enable packet transmission and reception in the DS3131:

1) Enable the channel in the port Tx and Rx configuration RAMs
2) Enable port data transmission in Layer 1
3) Enable Tx DMA and Rx DMA for the DS3131
4) Enable HDLC channel in Tx DMA and Rx DMA

| OFFSET/ADDRESS | ACRONYM | REGISTER NAME | DATA SHEET SECTION |
|---|---|---|---|
| 0010 | MC | Master Configuration Register | 4.2 |
| 02xx | TP[n]CR | Transmit Port n Control Register | 5.2 |
| 03xx | RH[n]CR | Receive HDLC n Control Register | 6.2 |
| 0770 | RDMACIS | Receive DMA Configuration Indirect Select Register | 8.1.5 |
| 0774 | RDMAC | Receive DMA Configuration Register | 8.1.5 |
| 0870 | TDMACIS | Transmit DMA Configuration Indirect Select Register | 8.2.5 |
| 0874 | TDMAC | Transmit DMA Configuration Indirect Select Register | 8.2.5 |

```
// TX port control register
// Set Bit 3 = 1 to allow data to be transmitted normally
read_reg(TP0CR + 4*port, data);
write_reg(TP0CR + 4*port, data | 0x0008);

// RX HDLC configuration
// Set bits 9 = 1 to enable the port (RPEN)
write_reg(RH[n]CR, 0x0200);

// Read the current channel value from the RX DMA Configuration RAM
// Set RDMACIS bits 5-0 = channel
// Set RDMACIS bits 10-8 = 100 to read lower word of dword 2
// Set RDMACIS bit 14 = 1 to read from RAM
write_reg_IS(RDMACIS, 0x4400 + channel);
read_reg(RDMAC, data);

// Enable channel in RX DMA
// Update RAM with new value
// Set RDMAC bit 0 = 1 to enable the HDLC channel
// Set RDMACIS bits 5-0 = channel
// Set RDMACIS bits 10-8 = 100 to read lower word of dword 2
// Set RDMACIS bit 14 = 1 to read from RAM
write_reg(RDMAC, data | 0x0001);
write_reg_IS(RDMACIS, 0x0400 + channel);

// Read the current channel value from the TX DMA Configuration RAM
```

```
// Set TDMACIS bits 5-0 = channel
// Set TDMACIS bits 11-8 = 0010 to read lower word of dword 1
// Set TDMACIS bit 14 = 1 to read from RAM
write_reg_IS(TDMACIS, 0x4200 + channel);
   read_reg(TDMAC, data);

// Enable channel TX DMA
// Update RAM with new value
// Set TDMAC bit 0 = 1 to enable the HDLC channel
// Set TDMACIS bits 5-0 = channel
// Set TDMACIS bits 11-8 = 0010 to read lower word of dword 1
// Set TDMACIS bit 14 = 0 to write to RAM
write_reg((TDMAC, data | 0x0001);
write_reg_IS(TDMACIS, 0x0200, + channel);

// Enable TX and RX DMA in the DS3131 master configuration register
// Set bit 0 = 1 to enable Receive DMA
// Set bits 2-1 = 00 to burst length maximum is 32 dwords
// Set bit 3 = 1 to enable Transmit DMA
// Set bits 6 = 1 for HDLC packet data on PCI bus is big endian
// Set bits 12-7 = 000000 to select Port 0 has the dedicated resources of the BERT
write_reg(MC, 0x0049);
```

## 4. Place the HDLC Channel in Loopback Mode

After the channel has been configured and enabled it takes approximately 5 frame periods, or 625µs, for the internal logic of the DS3131 to complete the transition to the new configuration. Once this transition has completed, the HDLC channel can then be placed in loopback mode so that all data transmitted on the channel is also received on that channel. Placing the HDLC channel in loopback mode prior to the 5-frame wait period can result in garbage data being written into the channel's Rx FIFO.

| OFFSET/ADDRESS | ACRONYM | REGISTER NAME | DATA SHEET SECTION |
|---|---|---|---|
| 01xx | RP[n]CR | Receive Port n Control Register | 5.2 |

```
// Wait for at least 5 frame periods for the internal DS3131 initialization to
complete
frame_wait(5);

// Set Bit 10 = 1 to enable loopback - routes transmit data back to the receive port
read_reg(RP0CR + 4*port, data);
write_reg(RP0CR + 4*port, data | 0x0400);
```

## 5. Queue, Send, Receive, and Check a Data Packet

Once the DS3131 initialization has been completed, data can be transmitted and received. Since the DS3131 is in loopback mode all data transmitted on the HDLC channel is also received on that channel. This section describes the process of how to build a data packet in host memory, transmit and receive the packet, and check the results. The following sections describe this process in detail.

### Initialize the Rx Free Queue

Before the DS3131 can transfer DMA-received packets from its internal FIFO to host memory, the host must instruct the DS3131 where to put the data. This is done through the Rx free queue. Each entry in the Rx free queue contains a pointer to an Rx data buffer and an Rx packet descriptor index. This example uses one Rx free queue entry. This entry contains one Rx free queue large buffer and one Rx packet descriptor. The DS3131 Rx large data buffer size has been set to 256 bytes (RLBS = 256). Additionally,

the DS3131 has been configured to use a 4-byte CRC and to write the Rx CRC into the Rx data buffer. Therefore, one Rx large data buffer is capable of holding up to 252 bytes of packet data.

| OFFSET/ADDRESS | ACRONYM | REGISTER NAME | DATA SHEET SECTION |
|---|---|---|---|
| 0710 | RFQLBWP | Receive Free Queue Large Buffer Host Write Pointer | 8.1.3 |
| 0718 | RFQLBRP | Receive Free Queue Large Buffer DMA Read Pointer | 8.1.3 |

```
// check for space in RX large free queue
read_reg(RFQLBWP, wr_ptr);
read_reg(RFQLBRP, rd_ptr);
if (rd_ptr > wr_ptr)
  cnt = rd_ptr – wr_ptr - 1;
else
  cnt = rfq_end_idx – wr_ptr + rd_ptr;

// If room in RX free queue then put 1 entry in the queue
// dword 0 = RX data buffer address
// (use RX data buffer starting at RX buffer area base address)
// dword 1 = corresponding RX descriptor index (use RX descriptor table index 0)
if (cnt > 0)
{
rx_dscr_idx = 0;
wr_dword(rfq_base_addr + wr_ptr*8, rx_buf_base_addr);
wr_dword(rfq_base_addr + wr_ptr*8 + 4, rx_dscr_idx);

// Advance the RX free queue large buffer write pointer by 1
write_reg(RFQLBWP, (wr_ptr + 1) % (rfq_end_idx + 1));
}
```

## Build the Packet in Host Memory

This example sends a 16-byte data packet. Before a packet can be sent it must be constructed in the host memory. Additionally, a corresponding Tx packet descriptor must also be constructed in host memory. Each of these tasks is detailed in the following code.

```
// Create a 16-byte data packet in memory in a TX buffer whose start address is the
TX
// buffer area base address
wr_dword(tx_buf_base_addr, 0x01234567);
wr_dword(tx_buf_base_addr + 4, 0x89ABCDEF);
wr_dword(tx_buf_base_addr + 8, 0x02468ACE);
wr_dword(tx_buf_base_addr + 12, 0x13579BDF);

// Create a TX descriptor (4 dwords) for the packet at TX descriptor
// TX descriptor table index 0
// dword0 = TX buffer address
// dword1 = EOF, CV, byte count, next descriptor pointer
// dowrd2 = HDLC channel
// dword3 = PV, next pending descriptor pointer (set to 0)
tx_dscr_idx = 0;
wr_dword(tdscr_base_addr + tx_dscr_idx*16, tx_buf_base_addr);
wr_dword(tdscr_base_addr + tx_dscr_idx*16 + 4, 0x80100000);


wr_dword(tdscr_base_addr + tx_dscr_idx*16 + 8, 0x00000000 + channel);
wr_dword(tdscr_base_addr + tx_dscr_idx*16 + 12, 0x00000000);
```

## Transmit and Receive the Packet

In order to transmit the packet, the Tx descriptor must be placed in the transmit pending queue and then the transmit pending-queue write pointer (TPQWP) must be incremented. When the DS3131 detects that pending queue is not empty (TPQWP not equal to TPQRP) it begins processing queue entries and the packet is transmitted.

| OFFSET/ADDRESS | ACRONYM | REGISTER NAME | DATA SHEET SECTION |
|---|---|---|---|
| 0028 | SDMA | Status Register for DMA | 4.3.2 |
| 080C | TPQWP | Transmit Pending Queue Host Write Pointer | 8.2.3 |
| 0810 | TPQRP | Transmit Pending Queue DMA Read Pointer | 8.2.3 |

```
// Read SDMA register to clear any previously set status bits
read_reg(SDMA, data);

// check free space in TX pending queue
read_reg(TPQWP, wr_ptr);
read_reg(TPQRP, rd_ptr)
if (rd_ptr > wr_ptr)
  cnt = rd_ptr – wr_ptr - 1;
else
 cnt = rfq_end_idx – wr_ptr + rd_ptr;

// If room in the TX pending queue create an entry for the packet
if (cnt > 0)
{
wr_dword(tpq_base_addr + wr_ptr*4, 0x0000000 + (channel << 16));

// Advance the TX pending queue write pointer
write_reg(TPQWP, (wr_ptr + 1) % (tpq_end_idx + 1));
}
```

## Check the Results

After waiting a sufficient period of time for the packet to be transmitted and received, several checks can be performed to determine if packet transmission and reception was successful. The following code details these checks.

| OFFSET/ADDRESS | ACRONYM | REGISTER NAME | DATA SHEET SECTION |
|---|---|---|---|
| 0028 | SDMA | Status Register for DMA | 4.3.2 |
| 0710 | RFQLBWP | Receive Free Queue Large Buffer Host Write Pointer | 8.1.3 |
| 0718 | RFQLBRP | Receive Free Queue Large Buffer DMA Read Pointer | 8.1.3 |
| 073C | RDQRP | Receive Done Queue Host Read Pointer | 8.1.4 |
| 0740 | RDQWP | Receive Done Queue DMA Write Pointer | 8.1.4 |
| 083C | TDQRP | Transmit Done Queue Host Read Pointer | 8.2.4 |
| 0840 | TDQWP | Transmit Done Queue DMA Write Pointer | 8.2.4 |

```
// wait 2 frame periods for packet to be transmitted/received
frame_wait(2);

// Check SDMA register
// Expected value = 0x6440, if not, it means there was error
read_reg(SDMA, data);

// Check to see how many entries are in the TX done queue (distance from TDQRP to
// TDQWP)
// Expected value is 1 - one entry in the TX done queue corresponding to the packet
that
// was sent
read_reg(TDQRP, rd_ptr);
read_reg(TDQWP, wr_ptr);
if (wr_ptr >= rd_ptr)
  cnt = wr_ptr - rd_ptr;
else
  cnt = tdq_end_idx + 1 - rd_ptr + wr_ptr;

// Check TX done queue descriptor
// Expected value = 0x0001000
// Bits 15-0 indicates the descriptor pointer
// Bits 21-16 indicate the channel number, it should be 1 in this example
// Bits 28-26 indicate the packet status, all 0 means the packet transmission is
complete
// and the descriptor pointer field corresponds to the first descriptor in the HDLC
packet
// that has been transmitted
rd_dword(tdq_base_addr + rd_ptr*4, tdq_entry);

// Advance the TX done queue read pointer
write_reg(TDQRP, (rd_ptr + 1) % (tdq_end_idx + 1));

// Check the RX large free queue to see how many RX buffers are in the queue
(distance // from RFQLBRP to RFQLBWP)
// Expected number is 0 since the queue had 1 buffer before the packet was received
and // packet reception required 1 buffer
read_reg(RFQLBRP, rd_ptr);
read_reg(RFQLBWP, wr_ptr);
if (wr_ptr >= rd_ptr)
  cnt = wr_ptr - rd_ptr;
else
  cnt = rfq_end_idx + 1 – rd_ptr + wr_ptr;

// Check RX done queue to see if any packets were received (distance from RDQRP to
// RDQWP)
// Expected value is 1 - one entry in the RX done queue entry corresponding to the
one
// packet that should have been received
read_reg(RDQRP, rd_ptr);
read_reg(RDQWP, wr_ptr);
if (wr_ptr >= rd_ptr)
  cnt = wr_ptr - rd_ptr;
else
  cnt = rdq_end_idx + 1 – rd_ptr + wr_ptr;
```

```
// Check the RX done queue descriptor
// Expected value = 0x40010000,
// Bits 15-0 indicates the descriptor pointer
// Bits 21-16 indicate the channel number, it should be 1 in this example
// Bits 26-24 indicate the buffer count, all 0 means that a complete packet has been
// received
rd_dword(rdq_base_addr + 8*rd_ptr, rdq_entry);

// Check the corresponding RX descriptor (4 dwords)
// dword 0 expected value = 0x10001000 the RX buffer address
// dword 1 expected value = 0x80140000
// Bits 15-0 is the next descriptor pointer
// Bits 28-16 is the number of bytes stored in the data buffer
// Bits 31-29 indicates buffer status
// dword 2 excepted value = 0x000B7503
// Bits 5-0 indicates HDLC channel number (should match TDQ entry channel)
// Bits 31-8 indicates the timestamp (can vary)
rdscr_idx = rdq_entry & 0x0000FFFF;
rd_dword(rdscr_base_addr + 16*rdscr_idx, rdscr_dword0);
rd_dword(rdscr_base_addr + 16*rdscr_idx + 4, rdscr_dword1);
rd_dword(rdscr_base_addr + 16*rdscr_idx + 8, rdscr_dword2);

// Check the data in the RX buffer
// 16 bytes of data + 4-byte CRC
// Expected values =    0x01234567
//                           0x89ABCDEF
//                           0x02468ACE
//                           0x13579BDF
//                           0x05127B09  (4-byte CRC)
byte_count = (rdscr_dword1 >> 16) & 0x00001FFF;
for (addr=rdscr_dword0, addr<rdscr_dword0+byte_count; addr=addr+4)
  rd_dword(addr, data);

// Advance the RX done queue read pointer
write_reg(RDQRP, (rd_ptr + 1) % (rdq_end_idx + 1));
```