

# Application Note 2872

## Step-by-Step Configuration for Bridge Mode (DS31256)

[www.maxim-ic.com](http://www.maxim-ic.com)

### INTRODUCTION

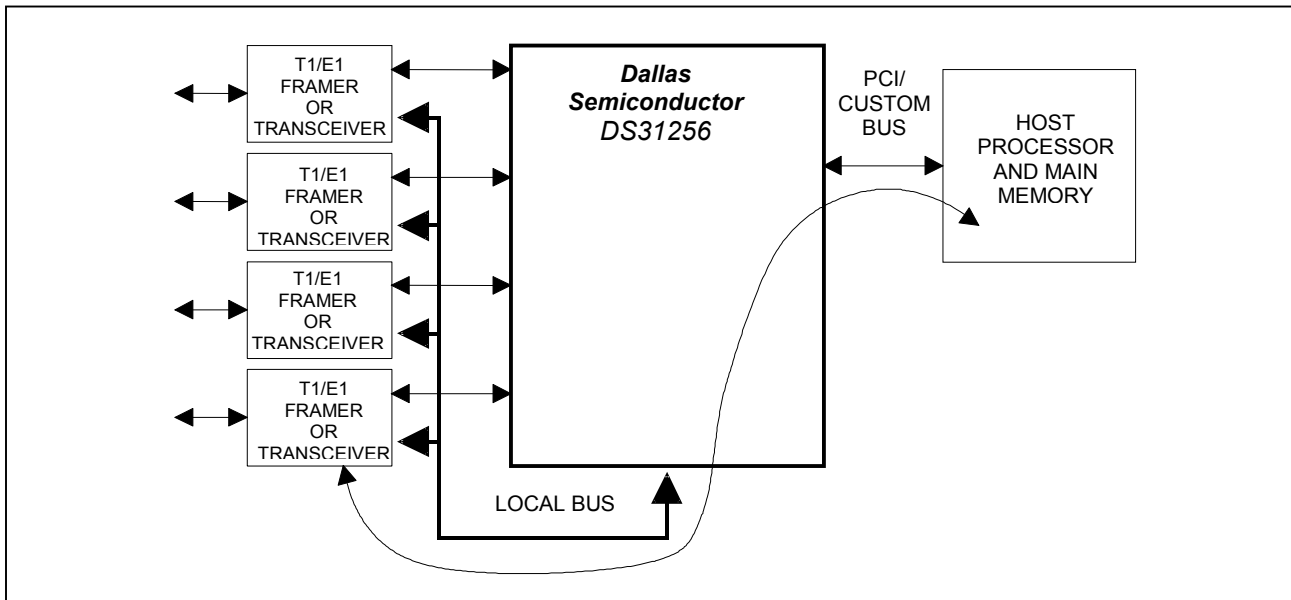
This application note describes an example of how to configure a single T1 port on the DS31256 while operating in bridge mode. Additionally, this example describes how to construct, send, receive, and check a packet in loopback mode on that port. This application note is presented as a coding example for easy adaptation to end user applications.

The local bus can operate in two modes in DS31256:

- PCI Bridge Mode
- Configuration 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 DS31256 and transfer the packet data. The DS31256 maps data from the PCI bus to the local bus. (Refer to the DS31256 data sheet, Section 11.1.1 for details.)

For details about using the configuration mode, please refer to *Application Note 2871: DS31256 Step-by-Step for Configuration Mode* at [www.maxim-ic.com/an2871](http://www.maxim-ic.com/an2871).



**This example has the following configuration:**

- Port 1 of the DS31256 is a channelized T1 port. All other ports are not used.
- Port 1 DS0's 0-3 of the DS31256 is assigned to HDLC channel 3. All other port 1 DS0's are unsigned.
- HDLC channel 3 of the DS31256 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 DS31256 is in loopback mode, when the packet is transmitted it will also be received by the DS31256. 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**

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 DS31256 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 DS31256 register at the specified address

Inputs:

address	=	the register address which is to be read
---------	---	--

Outputs:

data	=	the value read from the register
------	---	----------------------------------

- **write\_reg\_IS(address, data)**

Write the specified data to the specified DS31256 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 125 microseconds

Inputs:

- count = number of frame periods to wait

Outputs: none

**T1 CONFIGURATION MODE CODING EXAMPLE**

This coding example consists of the following steps:

1. Reset the DS31256.
2. Configure the DS31256.
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 DS31256 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 DS31256 data sheet for more detailed information.

**Reset the DS31256**

Resetting the DS31256 consists of three steps. First the DS31256 internal registers must be reset, second the internal RAM's of the DS31256 must be zeroed and then reset the DS31256 internal registers again.

**Reset the DS31256 Internal Registers**

A software reset can be performed on all registers in the DS31256 using the Master Reset Register (MRID). All internal registers will be 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.

OFFSET/ADDRESS	ACRONYM	REGISTER NAME	DATA SHEET SECTION
0000	MRID	Master Reset & ID Register	5.1

// Reset DS31256 using MRID registers master reset bit.

```
write_reg(MRID, 0x0001);
write_reg(MRID, 0x0000);
```

**Zero the DS31256 Internal RAMs**

The DS31256 internal configuration RAMs are not cleared by resetting the chip and therefore must be manually zeroed. This task is accomplished performing a series of writes to each internal ram in the DS31256 using the appropriate data and indirect select registers of the DS31256. This section details the procedure for accomplishing this task.

OFFSET/ADDRESS	ACRONYM	REGISTER NAME	DATA SHEET SECTION
03xx	CP[n]RDIS	Channelized Port n Register Data Indirect Select	6.3
03xx	CP[n]RD	Channelized Port n Register Data	6.3

```
// Zero Rx configuration and Tx configuration RAM's for all ports
```

```
for(port = 0; port < 16; port = port + 1)
```

```
{
  write_reg(CP[0]RD + 8 * port, 0x0000);
  for(ds0 = 0; ds0 < 128; ds0 = ds0 + 1)
  {
    // Set bits 9-8 = 01 to select Rx Configuration RAM
    // Set bits 9-8 = 10 to select Tx Configuration RAM
    write_reg_IS(CP[0]RDIS + 8 * port, (0x0100 + ds0));
    write_reg_IS(CP[0]RDIS + 8 * port, (0x0200 + ds0));
  }
}
```

OFFSET/ADDRESS	ACRONYM	REGISTER NAME	DATA SHEET SECTION
0400	RHCDIS	Receive HDLC Channel Definition Indirect Select	7.2
0404	RHCD	Receive HDLC Channel Definition	7.2

```
// Zero the Rx HDLC channel definition RAM
```

```
write_reg(RHCD, 0x0000);
```

```
for(channel = 0; channel < 256; channel = channel + 1)
```

```
  write_reg_IS(RHCDIS, channel);
```

OFFSET/ ADDRESS	ACRONYM	REGISTER NAME	DATA SHEET SECTION
0480	THCDIS	Transmit HDLC Channel Definition Indirect Select	7.2
0484	THCD	Transmit HDLC Channel Definition	7.2

```
// Zero the Tx HDLC channel definition RAM
```

```
write_reg(THCD, 0x0000);
```

```
for(channel = 0; channel < 256; channel = channel + 1)
```

```
  write_reg_IS(THCDIS, channel);
```

OFFSET/ADDRESS	ACRONYM	REGISTER NAME	DATA SHEET SECTION
0900	RFSBPIS	Receive FIFO Starting Block Pointer Indirect Select	8.2
0904	RFSBP	Receive FIFO Starting Block Pointer	8.2

```
// Zero the Rx FIFO Starting Block Pointer RAM
```

```
write_reg(RFSBP, 0x0000);
```

```
for(channel = 0; channel < 256; channel = channel + 1)
```

```
  write_reg_IS(RFSBPIS, channel);
```

OFFSET/ADDRESS	ACRONYM	REGISTER NAME	DATA SHEET SECTION
0910	RFBPIS	Receive FIFO Block Pointer Indirect Select	8.2
0914	RFBP	Receive FIFO Block Pointer	8.2

```
// Zero the Rx FIFO Block Pointer RAM
```

```
write_reg(RFBP, 0x0000);
```

```
for(channel = 0; channel < 256; channel = channel + 1)
```

```
  write_reg_IS(RFBPIS, channel);
```

OFFSET/ADDRESS	ACRONYM	REGISTER NAME	DATA SHEET SECTION
0920	RFHWMIS	Receive FIFO High Watermark Indirect Select	8.2
0924	RFHWM	Receive FIFO High Watermark	8.2

```
// Zero the Rx FIFO High Watermark RAM
write_reg(RFHWM, 0x0000);
for(channel = 0; channel < 256; channel = channel + 1)
    write_reg_IS(RFHWMIS, channel);
```

OFFSET/ADDRESS	ACRONYM	REGISTER NAME	DATA SHEET SECTION
0980	TFSBPIS	Transmit FIFO Starting Block Pointer Indirect Select	8.2
0984	TFSBP	Transmit FIFO Starting Block Pointer	8.2

```
// Zero the Tx FIFO Starting Block Pointer registers
write_reg(TFSBP, 0x0000);
for(channel = 0; channel < 256; channel = channel + 1)
    write_reg_IS(TFSBPIS, channel);
```

OFFSET/ADDRESS	ACRONYM	REGISTER NAME	DATA SHEET SECTION
0990	TFBPIS	Transmit FIFO Block Pointer Indirect Select	8.2
0994	TFBP	Transmit FIFO Block Pointer	8.2

```
// Zero the Tx FIFO Block Pointer RAM
write_reg(TFBP, 0x0000);
for(channel = 0; channel < 256; channel = channel + 1)
    write_reg_IS(TFBPIS, channel);
```

OFFSET/ADDRESS	ACRONYM	REGISTER NAME	DATA SHEET SECTION
09A0	TFLWMIS	Transmit FIFO Low Watermark Indirect Select	8.2
09A4	TFLWM	Transmit FIFO Low Watermark	8.2

```
// Zero the Tx FIFO Low Watermark RAM
write_reg(TFLWM, 0x0000);
for(channel = 0; channel < 256; channel = channel + 1)
    write_reg_IS(TFLWMIS, channel);
```

OFFSET/ADDRESS	ACRONYM	REGISTER NAME	DATA SHEET SECTION
0770	RDMACIS	Receive DMA Configuration Indirect Select	9.3.5
0774	RDMAC	Receive DMA Configuration	9.3.5

```
// Zero the Rx DMA configuration RAM
write_reg(RDMAC, 0x0000);
for(channel = 0; channel < 256; channel = channel + 1)
    write_reg_IS(RDMACIS, 0x0400 + channel);
```

OFFSET/ADDRESS	ACRONYM	REGISTER NAME	DATA SHEET SECTION
0870	TDMACIS	Transmit DMA Configuration Indirect Select	9.3.5
0874	TDMAC	Transmit DMA Configuration	9.3.5

```
// Zero the Tx DMA configuration RAM
write_reg(TDMAC, 0x0000);
for(channel = 0; channel < 256; channel = channel + 1)
    write_reg_IS(TDMACIS, 0x0200 + channel);
```

**Reset the DS31256 Internal Registers**

A software reset can be performed on all registers in the DS31256 using the Master Reset Register (MRID). All internal registers will be 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.

OFFSET/ADDRESS	ACRONYM	REGISTER NAME	DATA SHEET SECTION
0000	MRID	Master Reset & ID Register	5.1

```
// Reset DS31256 using MRID registers master reset bit.
```

```
write_reg(MRID, 0x0001);
```

```
write_reg(MRID, 0x0000);
```

**Configure the DS31256**

Configuration of the DS31256 consists of the following steps:

1. Configure the PCI registers
2. Configure the Layer 1 registers
3. Configure the HDLC registers
4. Configure the FIFO registers
5. Configure the DMA registers

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 3
```

```
port = 1;
```

```
channel = 3;
```

```
// Rx free queue base address
```

```
rfq_base_addr = 0x10000000;
```

```
// Rx free queue end address
```

```
// Rx free queue size = 16
```

```
rfq_end_idx = 0x000F;
```

```
// Rx done queue base address
```

```
rdq_base_addr = 0x10000100;
```

```
// Rx done queue end address
```

```
// Rx done queue size = 16
```

```
rdq_end_idx = 0x000F;
```

```
// 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 = 0x000F;
```

```
// Tx done queue base address
```

```
tdq_base_addr = 0x10000400;

// Tx done queue end address
// Tx done queue size = 16
tdq_end_idx = 0x000F;
// 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**

The PCI Bridge Mode allows the Host on the PCI Bus to access the Local Bus. The PCI Bus is used to control, monitor the DS31256 and transferred the packet data. The DS31256 will map data from the PCI Bus to the Local Bus. (Refer to the DS31256 data sheet section 10).

OFFSET/ ADDRESS	ACRONYM	REGISTER NAME	DATA SHEET SECTION
0x004/0A04	PCMD0	PCI Command Status 0	10.2

// Map DS31256 Configuration Registers to a PCI bus base address

```
write_reg(PCMD, 32'h80000000);

// PCI command/status register 0 - controls DS31256 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 Layer 1 Registers**

Each port of the DS31256 contains a layer 1 controller that performs several functions including:

- Assigning the HDLC channel number to the incoming and outgoing data
- Channelized local and network loopbacks
- Channelized selection of 64kps, 56kps or no data
- Channelized transmit DS0 channel fill of all ones
- Routing data to and from the BERT function
- Routing data to the V.54 loop pattern detector

Layer 1 configuration is performed on a port basis via the RP[n]CR, TP[n]CR, CP[n]RD, and CP[n]RDIS 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	6.2
02xx	TP[n]CR	Transmit Port n Control Register	6.2
03xx	CP[n]RDIS	Channelized Port n Register Data Indirect Select	6.3
03xx	CP[n]RD	Channelized Port n Register Data	6.3

```
// Set RX Port Control Register
// Set bits 2-0 = 000 for clock, data and sync are not inverted
// Set bits 5-4 = 00 for sync pulse 0 clocks early
// Set bits 7-6 = 00 for T1 mode
// Set bit 10 = 0 to disable local loopback
write_reg(RP[0]CR + 4 * port, 0x0000);

// Set Tx Port Control Register
// Set bit 2-0 = 000 for clock, data and sync are not inverted
```

```
// Set bit 3 = 0 to force all data at TD to be 1
// Set bits 5-4 = 00 for sync pulse 0 clocks early
// Set bits 7-6 = 0 for T1 mode
write_reg(TP[0]CR + 4 * port, 0x0000);

// RX Port Configuration Registers
// DS0's 0-3 disabled, assigned to HDLC channel
// CP[n]RDIS bits 9-8 = 01 for Receive Configuration
write_reg(CP[0]RD + 8 * port, 0x0000 + channel);
for(ds0 = 0; ds0 < 4; ds0 = ds0 + 1)
    write_reg_IS(CP[0]RDIS + 8 * port, 0x0100 + ds0);

// Tx Port Configuration Registers
// DS0's 0-3 disabled, assigned to HDLC channel
// CP[n]RDIS bits 9-8 = 10 for Transmit Configuration
write_reg(CP[0]RD + 8 * port, 0x0000 + channel);
for(ds0 = 0; ds0 < 4; ds0 = ds0 + 1)
    write_reg_IS(CP[0]RDIS + 8 * port, 0x0200 + ds0);
```

**Configure the HDLC Registers**

The DS31256 contains a 256-channel 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 via the RHCD, RHCDIS, THCD, and THCDIS registers.

OFFSET/ ADDRESS	ACRONYM	REGISTER NAME	DATA SHEET SECTION
0400	RHCDIS	Receive HDLC Channel Definition Indirect Select	7.2
0404	RHCD	Receive HDLC Channel Definition	7.2
0480	THCDIS	Transmit HDLC Channel Definition Indirect Select	7.2
0484	THCD	Transmit HDLC Channel Definition	7.2

```
// RX HDLC configuration
// Set bits 3-2 = 10 for 32-bit CRC
write_reg(RHCD, 0x0008);
write_reg_IS(RHCDIS, channel);

// 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(THCD, 0x0108);
write_reg_IS(THCDIS, channel);
```

**Configure the FIFO Registers**

The DS31256 contains a 16kB transmit FIFO and a 16kB receive FIFO. Each FIFO is divided into 1024 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 1024 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.

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 FHFO high watermark indicates how many blocks should be written into Rx FIFO by the HDLC engines before the DMA will begin 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 DS31256 are configured independently on an HDLC channel basis via the registers listed in the following tables.

OFFSET/ ADDRESS	ACRONYM	REGISTER NAME	DATA SHEET SECTION
0910	RFBPIS	Receive FIFO Block Pointer Indirect Select	8.2
0914	RFBP	Receive FIFO Block Pointer	8.2

```
// 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	8.2
0924	RFHWM	Receive FIFO High Watermark	8.2

```
// 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	8.2
0994	TFBP	Transmit FIFO Block Pointer	8.2

```
// Tx FIFO block linked list
// 0->1->2->3->0
for (block = 0; block < 4; block = block + 1)
{
// Bits 9-0 in TFBP 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	8.2
0984	TFSBP	Transmit FIFO Starting Block Pointer	8.2

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

OFFSET/ ADDRESS	ACRONYM	REGISTER NAME	DATA SHEET SECTION
09A0	TFLWMIS	Transmit FIFO Low Watermark Indirect Select	8.2
09A4	TFLWM	Transmit FIFO Low Watermark	8.2

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

### 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 DS31256 and the external PCI bus. The host, defined as the CPU or intelligent controller that sits on the PCI bus, instructs the DS31256 on how to handle the incoming and outgoing data.

This is accomplished using descriptors which are defined as pre-formatted messages passed from the host to the DMA block or vice versa. Via 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 will write 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 DS31256 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 DS31256 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)	9.2.3
0704	RFQBA1	Receive Free Queue Base Address 1 (upper word)	9.2.3
0708	RFQEA	Receive Free Queue end Address	9.2.3
070C	RFQSBSA	Receive Free Small Buffer Start Address	9.2.3
0710	RFQLBWP	Receive Free Queue Large Buffer Host Write Pointer	9.2.3
0714	RFQSBWP	Receive Free Queue Small Buffer Host Write Pointer	9.2.3
0718	RFQLBRP	Receive Free Queue Large Buffer DMA Read Pointer	9.2.3
071C	RFQSBRP	Receive Free Queue Small Buffer DMA Read Pointer	9.2.3
0730	RDQBA0	Receive Done Queue Base Address 0 (lower word)	9.2.4
0734	RDQBA1	Receive Done Queue Base Address 1 (upper word)	9.2.4
0738	RDQEA	Receive Done Queue end Address	9.2.4
073C	RDQRP	Receive Done Queue Host Read Pointer	9.2.4
0740	RDQWP	Receive Done Queue DMA Write Pointer	9.2.4
0750	RDBA0	Receive Descriptor Base Address 0 (lower word)	9.2.2
0754	RDBA1	Receive Descriptor Base Address 1 (upper word)	9.2.2
0770	RDMACIS	Receive DMA Configuration Indirect Select	9.3.5
0774	RDMAC	Receive DMA Configuration	9.3.5
0790	RLBS	Receive Large Buffer Size	9.2.1

```
// 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 201 = 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 will write 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 DS31256 processes transmit pending queue descriptor entries it creates transmit done queue descriptor queue entries. The DMA will write to the done queue when it has completed transmitting either a complete packet or data buffer depending on how the DS31256 is configured. Via these done queue descriptors, the DMA informs the host about the status of the outgoing packet data. Refer to the DS31256 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)	9.3.3
0804	TPQBA1	Transmit Pending Queue Base Address 1 (upper word)	9.3.3
0808	TPQEA	Transmit Pending Queue end Address	9.3.3
080C	TPQWP	Transmit Pending Queue Host Write Pointer	9.3.3
0810	TPQRP	Transmit Pending Queue DMA Read Pointer	9.3.3
0830	TDQBA0	Transmit Done Queue Base Address 0 (lower word)	9.3.4
0834	TDQBA1	Transmit Done Queue Base Address 1 (upper word)	9.3.4
0838	TDQEA	Transmit Done Queue end Address	9.3.4
083C	TDQRP	Transmit Done Queue Host Read Pointer	9.3.4
0840	TDQWP	Transmit Done Queue DMA Write Pointer	9.3.4
0850	TDBA0	Transmit Descriptor Base Address 0 (lower word)	9.3.2
0854	TDBA1	Transmit Descriptor Base Address 1 (upper word)	9.3.2
0870	TDMACIS	Transmit DMA Configuration Indirect Select	9.3.5
0874	TDMAC	Transmit DMA Configuration	9.3.5

// 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 Transmit 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);
```

## Enable the HDLC Channel

After the DS31256 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 DS31256:

1. Enable the HDLC channel in the port Tx and Rx configuration RAMs (Enable the HDLC channels)
2. Initialize the Rx Free Queue (Load the DMA Descriptors)
3. Enable Tx DMA and Rx DMA for the DS31256 (Enable the DMAs)
4. Enable HDLC channel Tx and Rx DMA
5. Enable port data transmission in Layer 1 (Turns on HDLC channels)

### Enable the HDLC Channel in the Port Tx and Rx Configuration RAMs

OFFSET/ ADDRESS	ACRONYM	REGISTER NAME	DATA SHEET SECTION
03xx	CP[n]RDIS	Channelized Port n Register Data Indirect Select	6.3
03xx	CP[n]RD	Channelized Port n Register Data	6.3

```
// Enable packet reception in port layer 1 Rx configuration RAM
for (ds0 = 0; ds0 < 4; ds0 = ds0 + 1)
```

```
{
// Read the current data value from the Rx Configuration RAM
// Set CP[n]RDIS bits 6-0 = DS0
// Set CP[n]RDIS bit 14 = 1 to read data from the RAM
// Set CP[n]RDIS bits 9-8 = 01 to select Rx configuration RAM
write_reg_IS(CP[0]RDIS + 8 * port, 0x4100 + ds0);
read_reg(CP[0]RD + 8 * port, data);
```

```
// Update memory with new value
// Set CP[n]RDIS bits 6-0 = DS0
// Set CP[n]RDIS bit 14 = 0 to write data to the RAM
// Set CP[n]RDIS bits 9-8 = 01 to select Rx configuration RAM
// Enable DS0 by setting bit 15 = 1 in CP[0]RD register
write_reg(CP[0]RD + 8 * port, data | 0x8000);
write_reg_IS(CP[0]RDIS + 8 * port, 0x0100 + ds0);
}
```

### Initialize the Rx Free Queue

Before the DS31256 can DMA received packets from its internal FIFO to host memory, the host must instruct the DS31256 where to put the data. This is done via 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 DS31256 Rx large data buffer size has been set to 256 bytes (RLBS = 256). Additionally, the DS31256 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	9.2.3
0718	RFQLBRP	Receive Free Queue Large Buffer DMA Read Pointer	9.2.3

```
// 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
if (wr_ptr = rfq_end_idx)
wr_ptr = 0;
else
wr_ptr = wr_ptr + 1;
write_reg(RFQLBWP, wr_ptr);
}
```

**Enable Tx and Rx DMA for the DS31256**

OFFSET/ ADDRESS	ACRONYM	REGISTER NAME	DATA SHEET SECTION
0010	MC	Master Configuration	5.2

```
// Enable Tx and Rx DMA in the DS31256 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 11-7 = 00000 to select Port 0 has the dedicated resources of the BERT
write_reg(MC, 0x0049);
```

**Enable HDLC Channel Tx and Rx DMA**

OFFSET/ ADDRESS	ACRONYM	REGISTER NAME	DATA SHEET SECTION
0770	RDMACIS	Receive DMA Configuration Indirect Select Register	9.3.5
0774	RDMAC	Receive DMA Configuration Register	9.3.5
0870	TDMACIS	Transmit DMA Configuration Indirect Select Register	9.3.5
0874	TDMAC	Transmit DMA Configuration Indirect Select Register	9.3.5

```
// Read the current channel value from the Rx DMA Configuration RAM
// Set RDMACIS bits 7-0 = 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 Rx DMA
// Update RAM with new value
// Set RDMAC bit 0 = 1 to enable the HDLC channel
// Set RDMACIS bits 7-0 = 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 7-0 = 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 7-0 = 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 Port Data Transmission in Layer 1

OFFSET/ ADDRESS	ACRONYM	REGISTER NAME	DATA SHEET SECTION
02xx	TP[n]CR	Transmit Port n Control Register	6.2

```
// Tx port control register
// Set Bit 3 TFDA1 = 1 to allow data to be transmitted normally
read_reg(TP[0]CR + 4 * port, data);
write_reg(TP[0]CR + 4 * port, data | 0x0008);
```

### Put the HDLC Channel in Loopback Mode

After the channel has been configured and enabled it takes approximately 5 frame periods, or 625 $\mu$ s, for the internal logic of the DS31256 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 will also be received on that channel. Placing the HDLC channel in loopback mode prior to the 5-frame wait period may 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	6.2

```
// Wait for at least 5 frame periods for the internal DS31256 initialization to complete
frame_wait(5);
```

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

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

Once the DS31256 initialization has been completed, data can be transmitted and received. Since the DS31256 is in loopback mode all data transmitted on the HDLC channel will also be received on that channel. This section will describe 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.

### BUILD THE PACKET IN HOST MEMORY

This example will send 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
```

```

// dword2 = 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 DS31256 detects that pending queue is not empty (TPQWP not equal to TPQRP) it will begin processing queue entries and the packet will be transmitted.

OFFSET/ ADDRESS	ACRONYM	REGISTER NAME	DATA SHEET SECTION
0028	SDMA	Status Register for DMA	5.3.2
080C	TPQWP	Transmit Pending Queue Host Write Pointer	9.3.3
0810	TPQRP	Transmit Pending Queue DMA Read Pointer	9.3.3

```

// 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, 0x00000000 + (channel << 16));

```

```

// Advance the Tx pending queue write pointer

```

```

if (wr_ptr = tpq_end_idx)

```

```

    wr_ptr = 0;

```

```

else

```

```

    wr_ptr = wr_ptr + 1;

```

```

write_reg(TPQWP, wr_ptr);

```

```

}

```



## 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	5.3.2
0710	RFQLBWP	Receive Free Queue Large Buffer Host Write Pointer	9.2.3
0718	RFQLBRP	Receive Free Queue Large Buffer DMA Read Pointer	9.2.3
073C	RDQRP	Receive Done Queue Host Read Pointer	9.2.4
0740	RDQWP	Receive Done Queue DMA Write Pointer	9.2.4
083C	TDQRP	Transmit Done Queue Host Read Pointer	9.3.4
0840	TDQWP	Transmit Done Queue DMA Write Pointer	9.3.4

```
// 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 = 0x0003000
// Bits 15-0 indicates the descriptor pointer
// Bits 23-16 indicate the channel number, it should be 3 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
if (wr_ptr = tdq_end_idx)
    wr_ptr = 0;
else
    wr_ptr = wr_ptr + 1;
write_reg(TDQRP, wr_ptr);
```

```
// 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 = 0x40030000,
// Bits 15-0 indicates the descriptor pointer
// Bits 23-16 indicate the channel number, it should be 3 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 expected value = 0x000B7503
// Bits 7-0 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
if (rd_ptr = rdq_end_idx)
    rd_ptr = 0;
else
    rd_ptr = rd_ptr + 1;
write_reg(RDQRP, rd_ptr);

```

## CONCLUSION

This application note has shown how to configure a single T1 port on the DS31256 in bridge mode. The coding example describes how to program (configure) the DS31256 and transmit/receive the data packet in detail (step by step).

If you have further questions about our HDLC controller products, please contact the Telecommunication Applications support team via email [telecom.support@dalsemi.com](mailto:telecom.support@dalsemi.com) or call 972-371-6555.