Digi®

Connectware™

NetSilicon

A Digi International Company

NS9775 Sample Driver Configurations

Making DEVICE NETWORKING *easy*™

NS9775 Sample Driver Configurations

Part number/version: 90000575_A
Release date: December 2004
www.netsilicon.com

To be used in conjunction with the NS9775 Hardware Reference, Rev. C (90000524_C)

NetSilicon, Inc. (Corporate Headquarters)
411 Waverley Oaks Road, Suite 304
Waltham, MA 02452 U.S.A.
Toll Free: 800 243-2333
Phone: 781 647-1234
Fax: 781 893-1338
Web: http://www.netsilicon.com/
Email: info@netsilicon.com

Contents

Using This Guide

Review this section for basic information about the guide you are using, as well as general support and contact information.

About this guide

This guide provides information sample driver configurations that you can use to create your own driver configurations.

The NET+ARM family is part of the NET+Works integrated product family, which includes the NET+OS network software suite.

Who should read this guide

This guide is for hardware developers, system software developers, and applications programmers who want to use the NS9775 for development.

To complete the tasks described in this guide, you must:

- Understand the basics of hardware and software design, operating systems, and microprocessor design.
- Understand the NS9775 architecture.

What's in this guide

The *NS9775 Sample Driver Configurations* provides examples of driver configurations for these modules: system control, Ethernet, PCI, memory controller, BBus DMA, serial controller, IEEE 1284, LCD, and USB.

Conventions used in this guide

This table describes the typographic conventions used in this guide:

This convention	Is used for
italic type	Emphasis, new terms, variables, and document titles.
bold, sans serif type	Menu commands, dialog box components, and other items that appear on-screen.
Select **Menu → option**	Menu commands. The first word is the menu name; the words that follow are menu selections.
`monospaced type`	Filenames, pathnames, and code examples.

Related documentation

- For information on the chip you are using, see the appropriate *Hardware Reference*.
- For schematics and BOM, review the documentation CD-ROM that came with your development kit.
- See the NET+OS software documentation for information for the chip you are using.

Customer support

To get help with a question or technical problem with this product, or to make comments and recommendations about our products or documentation, use the contact information listed in this table:

For	Contact information
Technical support	Telephone: 1 800 243-2333/ 1 781 647-1234 Fax: 1 781 893-1388 Email: tech_support@netsilicon.com
Documentation	techpubs@netsilicon.com
Ns9750 Errata	www.netsilicon.com/support/errata.jsp
NetSilicon home page	www.netsilicon.com
Online problem reporting	www.netsilicon.com/problemreporting.jsp

System Control Module Configuration

CHAPTER 1

This chapter provides sample driver configurations for the System Control module. Use these samples as guidelines for developing your own drivers.

Keep in mind that this is only one way to configure the System Control module.

SDRAM address compression

The NS9775 supports up to four SDRAM chip selects, allowing a maximum of four rows of external SDRAM parts. Each of these chip selects can be assigned a unique address space. These are the defaults after reset (see the system address map in the System Control Module chapter in the *NS9775 Hardware Reference)*:

```
0x0000 0000 - 0x0FFF FFFFSystem memory chip select 4 dynamic memory
0x1000 0000 - 0x1FFF FFFFSystem memory chip select 5 dynamic memory
0x2000 0000 - 0x2FFF FFFFSystem memory chip select 6 dynamic memory
0x3000 0000 - 0x3FFF FFFFSystem memory chip select 7 dynamic memory
```

Each of these address spaces is 256 MB. If the parts using chip selects are less than 256 MB, there will be holes in the memory space.

Example: Compressing an SDRAM address

Each chip select has two 256 Mbit x16 parts, resulting in 64 MB for each chip select. Taking the default settings leaves 192 MB holes in the SDRAM address space. Each chip select has a Base Address register and mask, which can be modified to allow you to compress the SDRAM address space and make it contiguous.

To determine the base and mask for each of the four chip selects, use this pseudo-code:

```
for (cs = 4; cs <= 7; cs++) {
          Base = (cs - 4) * SIZE
          Mask = ~(SIZE - 1);
}
```

In this code, `SIZE` = total bytes on each chip select.

Sample values

These are sample values for 64 MB on each chip select (`SIZE = 0x04000000`):

```
System memory chip select 4 dynamic memory base (0xA09001D0)=0x00000000
System memory chip select 5 dynamic memory base (0xA09001D8)=0X04000000
System memory chip select 6 dynamic memory base (0xA09001E0)=0x08000000
System memory chip select 7 dynamic memory base (0xA09001E8)=0x0C000000
```

```
System memory chip select 4 dynamic memory mask (0xA09001D4)=0xFC000000
System memory chip select 5 dynamic memory mask (0xA09001DC)=0xFC000000
System memory chip select 6 dynamic memory mask (0xA09001E4)=0xFC000000
System memory chip select 7 dynamic memory mask (0xA09001EC)=0xFC000000
```

Static memory on chip selects 0 through 3 can also have the address space changed as shown in this example.

Interrupt priorities

The System Control module takes in 32 interrupt lines. Each of these interrupt lines is assigned a unique interrupt ID (see the discussion of interrupt sources in the System Control Module chapter in the *NS9775 Hardware Reference*). The ID is randomly assigned and does not refer to a specific priority level.

Software is responsible for mapping each interrupt ID onto a unique interrupt priority level. For each of the 32 priority levels, there is an 8-bit Int Config register that enables the interrupt, selects IRQ or FIQ, and assigns the ID associated with the level. The Int Config registers are packed in groups of four to make 32-bit registers.

Example: Setting interrupt priorities

Set these interrupts as the four highest priority interrupts (from highest to lowest):

- Timer 1 (ID = 17)
- Ethernet transmit (ID = 5)
- Ethernet receive (ID = 4)
- BBus DMA (ID = 15)

The Int Config registers for the four highest priority level interrupts are grouped in the first set of Int Config registers (Int Config 0/1/2/3, register address A090 0144), as follows:

- Int Config Register 0 is [D31:24]
- Int Config Register 1 is {D23:16]
- Int Config Register 2 is [D15:08]
- Int Config Register 3 is [D07:00]

In this example, assuming none of the interrupts is FIQ, write the following to the Int Config Registers 0-3: 0x91 85 84 8F.

The Interrupt Vector Address registers are based on the level. In this example, the Interrupt Vector Address registers are set as shown:

```
Interrupt Vector Address Register Level 0 (0xA09000C4 = Timer 1 ISR
address

Interrupt Vector Address Register Level 1 (0xA09000C8) = Ethernet
Transmit ISR address

Interrupt Vector Address Register Level 2 (0xA09000CC) = Ethernet
Receive ISR address

Interrupt Vector Address Register Level 3 (0xA09000D0) = BBus DMA
ISR address
```

AHB arbiter configuration

The AHB arbiter has several registers that can be used to adjust system performance. Always write the AHB Arbiter General Configuration register to 0 for the best performance. This allows the CPU the fastest access to memory, and increases overall memory efficiency.

The BRC registers (A090 0004 / A090 0008 / A090 000C / A090 0010) weigh the priority of each master on the AHB bus. The CPU should get every other slot, and the default recommendation is for all other masters to get one slot at 100%.

Example: Programming the BRC

In addition to the CPU, you have five masters on the AHB bus: Eth Rx, Eth Tx, PCI, BBus, and LCD). Program the BRC registers as shown:

```
BRC0 (0xA0900004) = 0x80 81 80 82//CPU, Eth RX, CPU, Eth Tx
BRC1 (0xA0900008) = 0x80 84 80 85//CPU, PCI, CPU, BBus
BRC2 (0xA090000C) = 0x80 86 00 00//CPU, LCD, Unused, Unused
BRC3 (0xA0900010) = 0x00 00 00 00//Unused
```

Ethernet Configuration

CHAPTER 2

This chapter provides sample driver configurations for the Ethernet communications module. Use these samples as guidelines for developing your own drivers.

Keep in mind that this is only one way to configure the Ethernet communications module.

Attributes of sample configuration

Characteristics

- Uses MII PHY
- MAC operates in full duplex mode
- MAC appends CRC to all transmit frames and pads the frames to 64 bytes
- MAC checks length/type field in all TX and RX frames
- Statistics counters do not clear on read
- Station address logic accepts all frames
- Station address is `0x0060_0001_ba88`
- 4 receive rings enabled with frame lengths of 64, 128, 256, and 2K bytes
- Each receive ring consists of 2 buffer descriptors
- Transmit ring consists of 2 buffer descriptors with a complete frame in each
- The 2 transmit frames are 512 bytes and 1K bytes

Receive buffer descriptor layout

RX buffer descriptor	Location
Pool A/0	0x0020_0000
Pool A/1	0x0020_0010
Pool B/0	0x0020_0020
Pool B/1	0x0020_0030
Pool C/0	0x0020_0040
Pool C/1	0x0020_0050
Pool D/0	0x0020_0060
Pool D/1	0x0020_0070

Receive and transmit buffer layout

Buffer	Location
Rx Pool A/0	0x0021_0000
Rx Pool A/1	0x0021_0040
Rx Pool B/0	0x0021_0080
Rx Pool B/1	0x0021_0100
Rx Pool C/0	0x0021_0200
Rx Pool C/1	0x0021_0300
Rx Pool D/0	0x0021_0800
Rx Pool D/1	0x0021_1000
Tx 0	0x0021_2000
Tx 1	0X0021_2400

Note: The MAC, SAL, and RMII modules must be held in reset when any of their configuration bits are changed, because all configuration bits are considered steady state signals and are not synchronized to their respective clock domains. Changing a configuration bit without resetting these modules can cause unexpected results, which could lead to a lock-up condition. The MIIM module, which controls the MII management interface, is not affected because it runs off the same clock as the MAC host interface.

Resets

The SRST (soft reset) field in the MAC Configuration register 1 is a common soft reset to the RX_WR, TX_RD, MAC (except HOST), SAL (except host interface), and RMII modules. When SRST is set to 1, all of these modules are reset.

A less restrictive reset scheme, and one that is necessary when setting up the external PHY using the MII management interface, is to reset only the MCS, TFUN, and RFUN modules in the MAC and the non-host logic in SAL by setting RPEMCSR, RPERFUN, RPEMCST, and RPETFUN in MAC Configuration Register 1. The RMII module is reset by setting RPESMII in the PHY Support register (SUPP).

Ethernet configuration sequence

After reset is negated, use these steps to configure the MAC and Ethernet front-end module.

1. Write `0x8080_0200` to Ethernet General Control Register 1.
   - **a** Remove soft reset from Receive Packet processor by setting ERX.
   - **b** Remove soft reset from Transmit Packet processor by setting ETX.
   - **c** Remove soft reset from MAC, STAT, and SAL host interfaces by clearing MAC_HRST.
2. Write `0x0000_8000` to PHY Support register.
   - **a** Reset RMII interface module by setting RPERMII.

   Write `0x0000_0f00` to MAC Configuration Register 1.
   - **b** Remove common soft reset to RX_WR, TX_RD, MAC, SAL, and RMII modules, except the host interface, by clearing SRST.
   - **c** Reset MCS, TFUN, and RFUN modules in MAC and non-host logic in SAL by setting RPEMCSR, PERFUN, RPEMCST, and RPETFUN. The MIIM module is not reset.
3. Configure the external PHY using the MII management registers in the MAC (MCFG, MCMD, MADR, MWTD, MRDD, and MIND).
4. Write `0x0000_0033` to MAC2.
   - **a** Configure MAC to append CRC and padding by setting PADEN and CRCEN.
   - **b** Configure MAC to check length/type field by setting FLENC.
   - **c** Configure MAC for full-duplex mode by setting FULLD.
5. Write `0x0000_0008` to SAFR.
   - **a** Configure the SAL module to accept all frames by setting PRO.

6 Write 0x0000_88ba to SA1.

Write 0x0000_0100 to SA2.

Write 0x0000_6000 to SA3.

a Configure station address to the unicast address 0x0060_0001_ba88.

7 Write 0x0020_0000 to RXAPTR. This initializes the address of the initial buffer descriptor for the *A* pool of buffers to 0x0020_0000.

8 Write 0x0020_0020 to RXBPTR. This initializes the address of the initial buffer descriptor for the *B* pool of buffers to 0x0020_0020.

9 Write 0x0020_0040 to RXCPTR. This initializes the address of the initial buffer descriptor for the *C* pool of buffers to 0x0020_0040.

10 Write 0x0020_0060 to RXDPTR. This initializes the address of the initial buffer descriptor for the *D* pool of buffers to 0020_0060.

11 Set up first buffer descriptor for the A pool of buffers in system memory, as shown:

a Write 0x0021_0000 to address 0x0020_0000.

b Write 0x0000_0040 to address 0x0020_0004.

c Write 0x0000_0000 to address 0x0020_0008.

d Write 0x2000_0000 to address 0x0020_000C.

This initializes the first buffer descriptor for the A pool of buffers to:

W = 0

I = 0

E = 1

Pointer = 0x0021_0000

Status = 0x0000

F = 0

Length = 0x40

12 Set up the second buffer descriptor for the A pool of buffers in system memory, as shown:

a Write 0x0021_0040 to address 0x00020_0010.

b Write 0x0000_0040 to address 0x0020_0014.

c Write 0x0000_0000 to address 0x0020_0018.

d Write 0xA000_0000 to address 0x0020_001C.

This initializes the second buffer descriptor for the A pool of buffers to:

W = 1

I = 0

E = 1

Pointer = 0x0021_0040

Status = 0x0000

F = 0

Length = 0x40

13 Set up the first buffer descriptor for the B pool of buffers in system memory, as follows:

a Write 0x0021_0080 to address 0x0020_0020.

b Write 0x0000_0080 to address 0x0020_0024.

c Write 0x0000_0000 to address 0x0020_0028.

d Write 0x2000_0000 to address 0x0020_002C.

This initializes the first buffer descriptor for the B pool of buffers to:

W = 0

I = 0

E = 1

Pointer = 0x0021_0080

Status = 0x0000

F = 0

Length = 0x80

14 Set up the second buffer descriptor for the B pool of buffers in system memory, as follows:

a Write `0x0021_0100` to address `0x0020_0030`.

b Write `0x0000_0080` to address `0x0020_0034`.

c Write `0x0000_0000` to address `0x0020_0038`.

d Write `0xA000_0000` to address `0x0020_003C`.

This initializes the second buffer descriptor for the B pool of buffers to:

W = 1

I = 0

E = 1

Pointer = 0x0021_0100

Status = 0x0000

F = 0

Length = 0x80

15 Set up the first buffer descriptor for the C pool of buffers in system memory, as follows:

a Write `0x0021_0200` to address `0x0020_0040`.

b Write `0x0000_0100` to address `0x0020_0044`.

c Write `0x0000_0000` to address `0x0020_0048`.

d Write `0x2000_0000` to address `0x0020_004C`.

This initializes the first buffer descriptor for the C pool of buffers to:

W = 0

I = 0

E = 1

Pointer = 0x0021_0200

Status = 0x0000

F = 0

Length = 0x100

16 Set up the second buffer descriptor for the C pool of buffers in system memory, as follows:

a Write `0x0021_0300` to address `0x0020_0050`.

b Write `0x0000_0100` to address `0x0020_0054`.

c Write `0x0000_0000` to address `0x0020_0058`.

d Write `0xA000_0000` to address `0x0020_005C`.

This initializes the second buffer descriptor for the C pool of buffers to:

W = 1

I = 0

E = 1

Pointer = 0x0021_0300

Status = 0x0000

F = 0

Length = 0x100

17 Set up the first buffer descriptor for the D pool of buffers in system memory, as follows:

a Write `0x0021_0800` to address `0x0020_0060`.

b Write `0x0000_0800` to address `0x0020_0064`.

c Write `0x0000_0000` to address `0x0020_0068`.

d Write `0x2000_0000` to address `0x0020_006C`.

This initializes the first buffer descriptor for the D pool of buffers to:

W = 0

I = 0

E = 1

Pointer = 0x0021_0800

Status = 0x0000

F = 0

Length = 0x800

18 Set up the second buffer descriptor for the D pool of buffers in system memory, as follows:

a Write `0x0021_1000` to address `0x0020_0070`.

b Write `0x0000_0800` to address `0x0020_0074`.

c Write `0x0000_0000` to address `0x0020_0078`.

d Write `0xA000_0000` to address `0x0020_007C`.

This initializes the second buffer descriptor for the D pool of buffers to:

W = 1

I = 0

E = 1

Pointer = 0x0021_1000

Status = 0x0000

F = 0

Length = 0x800

19 Write `0x0000_0000` to TXPTR.

a Set the pointer (that is, the internal RAM address) to the initial transmit buffer descriptor location `0x00` in the TX buffer descriptor RAM.

20 Initialize a ring of 2 transmit buffer descriptors in the TX buffer descriptor RAM, as follows:

a Write `0x0021_2000` to address `0xA060_1000` (RAM addr = 0).

b Write `0x0000_0400` to address `0xA060_1004` (RAM addr = 1).

c Write `0x0000_0000` to address `0xA060_1008` (RAM addr = 2).

d Write `0x1000_0000` to address `0xA060_100C` (RAM addr = 3).

This initializes the first transmit buffer descriptor to:

W = 0

I = 0

E = 1

Pointer = 0x0021_2000

Status = 0x0000

F = 1

Length = 0x400

e Write 0x0021_2400 to address 0xA060_1010 (RAM addr = 4).

f Write 0x0000_0200 to address 0xA060_1014 (RAM addr = 5).

g Write 0x0000_0000 to address 0xA060_1018 (RAM addr = 6).

h Write 0xB000_0000 to address 0xA060_101C (RAM addr = 7).

This initializes the second transmit buffer descriptor to:

W = 1

I = 0

E = 1

Pointer = 0x0021_2400

Status = 0x0000

F = 0

Length = 0x200

21 Fill the system memory with the transmit frame data, as follows:

a Write 1K transmit frame to addresses 0x0021_2000-0x0021_23FC.

b Write 512 byte transmit frame to addresses 0x0021_2400-0x0021_25FF.

22 Write 0x8088_0000 to Ethernet General Control Register 1.

a Start initialization of internal buffer descriptor registers from RXAPTR, RXBPTR, RXCPTR, and RXDPTR by setting ERXINIT.

Wait 5 usec and read the Ethernet General Status register to verify that the RXINIT field is set, indicating that initialization is complete.

Write 0x0010_0000 to the Ethernet General Status register.

b Clear RXINIT.

Write 0x8080_0000 to the Ethernet General Control Register 1.

c Clear ERXINIT.

23 Write 0x02FF_001F to EINTRMSK.

a Enable all interrupts.

24 Write 0x0000_0001 to Ethernet General Control Register 2.

a Release clear of statistics counters by clearing CLRCNT.

b Enable statistics counters by setting STEN.

25 Write 0x0000_0001 to the Mac Configuration Register 1.

a Enable MAC receivers by setting RXEN.

b Take MCS, TFUN, and RFUN modules in MAC and non-host logic in SAL out of reset by clearing RPEMCSR, RPERFUN, RPEMCST, and RPETFUN.

Write 0x0000_0000 to the PHY Support register.

c Take RMII interface module out of reset by clearing RPERMII.

26 Write 0xE0C0_0000 to the Ethernet General Control Register 1.

– Enable RX DMA by setting ERXDMA.

– Enable TX DMA by setting ETXDMA.

Servicing interrupts

This section provides steps for servicing receive and transmit interrupts.

Servicing receive interrupts

After a receive frame has been stored in system memory, the buffer descriptor has been updated, and the next buffer descriptor has been read from memory, a bit is set in the Ethernet Interrupt Status register that indicates in which ring the frame was stored — RXDONEA, RXDONEB, RXDONEC, or RXDONED. If the I bit in the buffer descriptor is set, the RXBUFC field in the Ethernet Interrupt Status register is also set; these set fields cause interrupts to the system if the associated mask bits in the Ethernet Interrupt Enable register are also set.

Sample receive interrupt service routine

1. The software tracks the location of the current buffer descriptor and reads the status, buffer pointer, and buffer length fields from the descriptor. The buffer length field contains the size of the received frame, in bytes.

   The status field has information about the type of frame received (for example, multicast).

2. The buffer pointer and buffer length fields are used to read the complete frame. When the entire frame has been read, the F bit in the buffer descriptor is cleared to allow it to be reused. The buffer length field is updated with the size of the buffer.

3. Write a 1 to clear the RXDONEA/B/C/D and RXBUFC fields in the Ethernet Interrupt Status register.

Servicing transmit interrupts

After a transmit frame has been completely transmitted by the MAC and the buffer descriptor has been updated in the transmit buffer descriptor RAM, TXDONE is set in the Ethernet Interrupt Status register. If the I bit in the buffer descriptor is set, the TXBUFC field in the Ethernet Interrupt Status register is also set; these set fields

cause interrupts to the system if the associated mask bits in the Ethernet Interrupt Enable register are also set.

Sample transmit interrupt service routine

1. Read the pointer to the next buffer descriptor from the Transmit Buffer Descriptor Pointer Offset register and use this to locate the buffer descriptor that was used for the last frame transmitted.

   Read the status, buffer pointer, and buffer length fields from the descriptor. The buffer length field contains the size of the transmitted frame, in bytes.

   The status field has information about the type of frame transmitted (for example, multicast), whether the frame experienced an error, or whether the frame length was aborted.

2. Write a 1 to clear the TXDONE and TXBUFXC fields in the Ethernet Interrupt Status register.

   If the frame experienced an error, or the transmit logic has no more packets to send, the TCLER field in the Ethernet General Control Register 2 must be toggled from low to high to re-enable the transmit process once a new frame is ready to be transmitted.

PCI Bridge Configuration

CHAPTER 3

This chapter provides sample driver configurations for the PCI-to-AHB bridge. Use these samples as guidelines for developing your own drivers.

Keep in mind that this is only one way to configure the PCI-to-AHB bridge.

NS9775 PCI configuration

These are the hardware configuration pins for the PCI-to-AHB bridge:

- PCI_CENTRAL_RSC_N (internal pulldown)
  - 0: NS9750 provides PCI central resource functions
    - RST# driven through NS9750: SERR# input to NS9750
    - AD, C/BE, and PAR driven low when RST# active
  - 1: NS9750 does not provide PCI central functions
    - RST# configured as input; SERR# configured as output
    - AD, CB/E, and PAR tri-stated when RST# active
- RTCK (internal pullup)
  - 0: Disable internal arbiter
  - 1: Enable internal arbiter
- BP_STAT[0](NS9775) (internal pullup)
  - 0: CardBus mode
  - 1: PCI mode

Configuration with NS9775 as PCI Host

Figure 1 shows a sample system consisting of NS9775 and one external PCI device. NS9775 is the PCI host device in this system. Use this diagram as a guide to the configuration sequences discussed in this section.

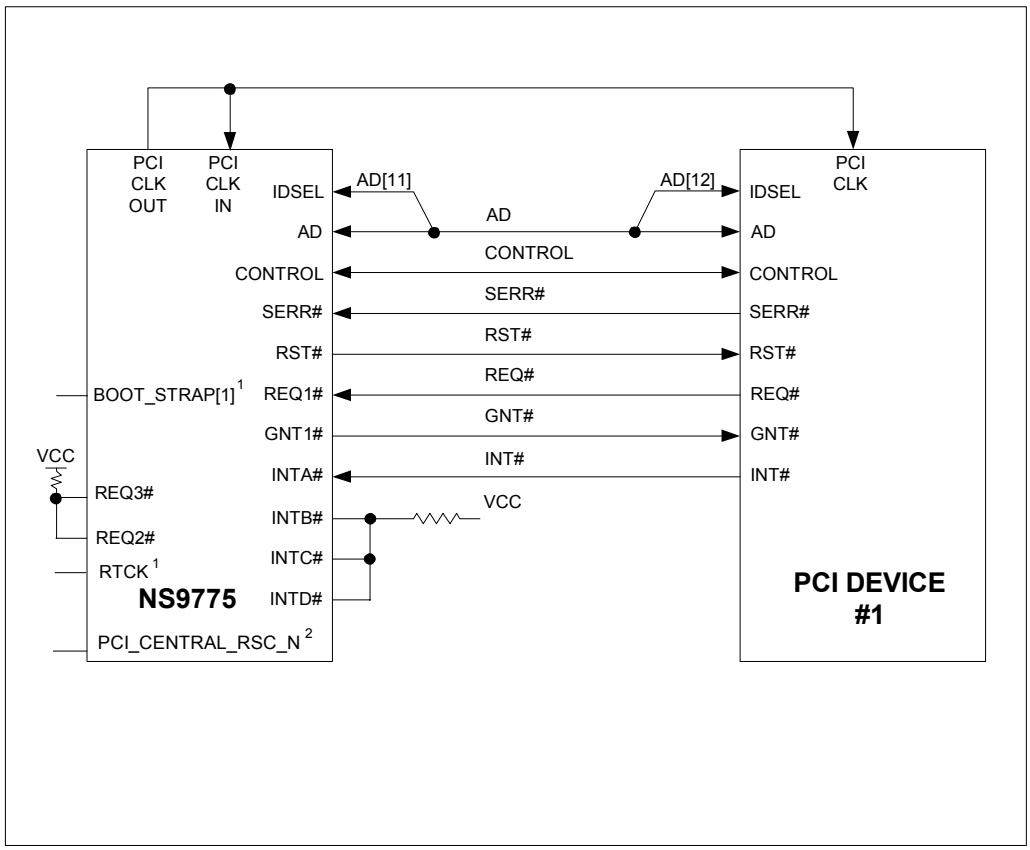

Figure 1: Sample system with NS9775 as PCI host

1 RTCK has an internal pullup, so the pin can float.

2 PCI_CENTRAL_RESOURCE_N has an internal pulldown, so the pin can float.

System characteristics

- NS9775:
  - Mapped to 256 MB window in PCI memory space
  - Provides central resource functions
  - Provides PCI arbiter
  - Provides PCI interrupt controller
  - Device 0 on PCI bus

Important: Note that in cases where NS9775 provides the PCI clock, the PCI clock connection to the NS9775 must still be made external to the NS9775, (that is, connect PCI_CLK_OUT to PCI_CLK_IN). This is done to minimize the clock skew between the NS9775 and external PCI devices.

- External PCI Device #1:
  - Mapped to 128 MB window in PCI memory space, using Base Address Register 0
  - Mapped to 64 KB window in PCI IO space, using Base Address Register 1
  - Single function PCI device
  - PCI master
  - Single interrupt
  - Device 1 on PCI bus
- System:
  - PCI Device #1 memory mapped to 0x0F0000_0000-0xF7FF_FFFF in PCI memory space (128 MB)
  - NS9775 memory mapped to 0x1000_0000-0x1FFF_FFFF in PCI memory space (256 MB)
  - PCI Device #1 IO mapped to 0x2000_0000-0x2000_FFFF in PCI IO space (64 KB)
  - PCI Device #1 interrupt connected to INTA# of NS9775
  - PCI Device #1 accesses to NS9775 mapped to 0x3000_0000-0x3FFF_FFFF in NS9775's memory space

PCI configuration sequence

In this application, RST# is controlled by the PCI bit in the Reset and Sleep Control register in the System Control module. The PCI bit is cleared to 0 to assert RST#. Since PCI defaults to 1, RST# is negated without the processor writing to this bit when NS9775 comes out of reset.

The PCI configuration sequence includes these steps, explained in detail on the following pages:

- Configure the bridge-specific registers
- Configure the Bridge PCI Configuration registers
- Configure PCI Device #1
- Final configuration

Configure the bridge-specific registers

Configuring the bridge-specific registers pertains only to the initialization of those registers that control the operation of the PCI-to-AHB bridge and the PCI arbiter.

Note: Registers that set the read-only values of the PCI Configuration registers in the bridge (for example, the PCI Configuration 0/1/2/3 registers) are not initialized at this time because the NS9775 does not need this information when it is the host.

1. Write 0x0000_0013 to the PCI Arbiter Interrupt Enable register.
   - Enables interrupts from these sources:
     - Bridge broken master (bit 0)
     - PCI Device #1 broken master (bit 1)
     - SERR# asserted by PCI Device #1 (bit 4)
2. Write 0x0000_0010 to the PCI Miscellaneous Support register.
   - Enables Base Address Register 0, which decodes a 256 MB window in PCI memory space by setting EN_BAR0. All other Base Address registers are disabled.

3 Write 0x7878_7878 to the PCI Bridge AHB to PCI Memory Address Translate 0 register.
   – Maps accesses to the lower 128 MB of NS9775's PCI memory window (0x80000_0000-0x87FF_FFFF) to the 128 MB window where PCI Device #1 is located (0xF000_0000-0xF7FF_FFFF) by setting PALT0VAL, PALT1VAL, PALT2VAL, and PALT3VAL to 0x78.

4 Write 0x0000_0200 to the PCI Bridge AHB to PCI IO Address Translate register.
   – Maps accesses to the lower 64 KB of Mercury's PCI IO window (0xA000_0000-0xA000_FFFF) to the 64 KB window where the IO space for PCI Device #1 is located (0x2000_0000-0x2000_FFFF) by setting PALT8VAL to 0x200.

5 Write 0x0000_0003 to the PCI Bridge PCI to AHB Memory Address Translate 0 register.
   – Maps PCI accesses to NS9775 to a 256 MB window in NS9775's memory space located at 0x3000_0000-0x3FFF_FFFF by setting MALT0VAL to 0x3. All other values in this register are not used because only Base Address Register 0 is enabled in the PCI Miscellaneous Support register.

6 Write 0x0000_0003 to the PCI Bridge Address Translation Control register.
   – Enables PCI to AHB address translation by setting MALT_EN.
   – Enables AHB to PCI address translation by setting PALT_EN.

Configure the Bridge PCI Configuration registers

Configuring the Bridge PCI Configuration registers pertains only to the initialization of registers that control the operation of the PCI-to-AHB bridge for this system. **It is important that you follow these steps exactly in the order in which they are shown.**

1 Write 0x8000_0004 to 0xA010_0000 (PCI CONFIG_ADDR space).

   a Set up the Configuration Address Port (CONFIG_ADDR) register in the bridge to access the PCI Status and Command configuration registers in the bridge. Note that the bridge is accessed as DEVICE_NUMBER 0.

   Write 0x0000_0046 to 0xA020_0000 (PCI CONFIG_DATA space).

   b Initialize the PCI Command register as follows:

      i Disable bridge response to PCI IO accesses (bit 0 = 0).

ii Enable bridge response to PCI memory accesses (bit 1 = 1).

iii Enable bridge as PCI master (bit 2 = 1).

iv Disable bridge's ability to generate memory write and invalidate command (bit 4 = 0). This bit has no effect on the operation of the bridge and should always be 0.

v Enable assertion of PERR# by the bridge when it detects a parity error (bit 6 = 1).

vi Disable the bridge driving SERR# (bit 8 = 0) since NS9775 is the host.

2 Write `0x8000_0010` to `0xA010_0000` (PCI CONFIG_ADDR space).

a Set up the Configuration Address Port register in the bridge to access PCI Base Address 0 register in the bridge.

Write `0x1000_0000` to `0xA020_0000` (PCI CONFIG_DATA space).

b Initialize the PCI Base Address 0 register to allow the NS9775 to respond to a 256 MB window in the PCI memory space starting at `0x1000_0000`. All other Base Address registers are disabled.

3 Write `0x8000_000C` to `0xA010_0000` (PCI CONFIG_ADDR space).

a Set up the Configuration Address Port register in the bridge to access the PCI Latency Timer and Cache Size configuration registers in the bridge.

Write `0x0000_FF00` to `0xA020_0000` (PCI CONFIG_DATA space).

b Initialize the PCI Latency Timer register to `0xFF` to allow NS9775 to stay on the bus for up to 255 PCI clocks when it is bursting data on the PCI bus (bits [15:08] = `0xFF`).

c Initialize the PCI Cache Size register to 0 (bits [07:00] = `0x00`). Note that this field has no effect on bridge operation and should always be set to `0x00`.

4 Write `0x0000_F901` to the PCI Bridge Interrupt Enable register.

– Enables all of the interrupts from the bridge that are caused by PCI or AHB bus errors.

Configure PCI Device #1

Configuring PCI Device #1 pertains only to the initialization of registers that control the operation of PCI Device #1. NS9775 must wait at least 2^{25} PCI clocks from RST# negated before initiating any external configuration cycles.

1. Write 0x8000_0810 to 0xA010_0000 (PCI CONFIG_ADDR space).
   - **a** Set up the Configuration Address Port register in the bridge to access the PCI Base Address 0 register in PCI Device #1.

   Write 0xF000_0000 to 0xA020_0000 (PCI CONFIG_DATA space).
   - **b** Map 128 MB memory window supported by PCI Device #1 to 0xF000_0000–0xF7FF_FFFF by setting the base address using bits [31:27] of the PCI Base Address register.
2. Write 0x8000_0814 to 0xA010_0000 (PCI CONFIG_ADDR space).
   - **a** Set up the Configuration Address Port register in the bridge to access the PCI Base Address 1 register in PCI Device #1.

   Write 0x2000_0001 to 0xA020_0000 (PCI CONFIG_DATA space).
   - **b** Map 64 KB I/O window supported by PCI Device #1 to 0x2000_0000–0x2000_FFFF by setting the base address using bits [31:16] of PCI Base Address 1 register.
3. Write 0x8000_080C to 0xA010_0000 (PCI CONFIG_ADDR space).
   - **a** Set up the Configuration Address Port register in the bridge to access PCI Latency Timer and Cache Size configuration registers in PCI Device #1.

   Write 0x0000_FF00 to 0xA020_0000 (PCI CONFIG_DATA space).
   - **b** Initialize the PCI Latency Timer register to 0xFF to allow PCI Device #1 to stay on the bus for up to 255 PCI clocks when it is bursting data on the PCI bus (bits [15:08] = 0xFF). This is necessary because the latency resulting from reads to the bridge can be long, due to the AHB bus arbitration within NS9775.
   - **c** Initialize the PCI Cache Size register to 0 (bits [07:00] = 0x00).

Final configuration

The final configuration is where PCI Device #1 is enabled for PCI bus operations. **It is important that you follow these steps exactly in the order in which they are shown.**

1. Write `0x8000_0804` to `0xA010_0000` (PCI CONFIG_ADDR space).
   a. Set up the Configuration Address Port register in the bridge to access PCI Status and Command Configuration registers in PCI Device #1.

   Write `0x0000_0343` to `0xA020_0000` (PCI CONFIG_DATA space).

   b. Initialize the PCI Command register as follows:
      i. Enable PCI Device #1 to respond to PCI I/O accesses (bit 0 = 1).
      ii. Enable PCI Device #1 to respond to PCI memory accesses (bit 1 = 1).
      iii. Disable PCI Device #1's ability to act as a bus master (bit 2 = 0). Although PCI Device #1 can be a master, it is premature to enable this capability because it has not been initialized internally.
      iv. Disable PCI Device #1's ability to respond to PCI special cycle operations, because the bridge does not generate these operations (bit 3 = 0).
      v. Disable PCI Device #1's ability to generate the memory write and invalidate command since the bridge does not treat this command differently than write commands (bit 4 = 0).
      vi. Disable *VGA palette snooping* by PCI Device #1 because it is not a graphics adapter (bit 5 = 0).
      vii. Enable assertion of PERR# by PCI Device #1 when it detects a parity error (bit 6 = 1).
      viii. Disable PCI Device #1's ability to perform *address stepping* because there is no need for this function in this system (bit 7 = 0).
      ix. Enable PCI Device #1 to drive SERR# (bit 8 = 1).
      x. Enable PCI Device #1 to perform *fast back-to-back* cycles because the bridge supports these cycles as a target and they improve PCI performance (bit 9 = 1).
2. At this point, NS9775 initializes PCI Device #1's internal resources (for example, non-PCI registers, memories, and so on) through the PCI bus.

3 Write 0x0000_0002 to the PCI Arbiter Configuration register.
   – Enables the PCI bus request from PCI Device #1 that is connected to the REQ1# input of the NS9775 by setting PCIEN_M1.

4 Write 0x8000_0804 to 0xA010_0000 (PCI CONFIG_ADDR space).
   a Set up the Configuration Address Port register in PCI Device #1 to access the PCI Status and Command configuration registers in PCI Device #1.

   Write 0x0000_0347 to 0xA020_0000 (PCI CONFIG_DATA space)
   b Initialize the PCI Command register as in Steps 1b(i) through 1b(x), with the following exception:
      - Enable PCI Device #1 as a bus master (bit 2 = 1), as it has now been initialized.

5 Set bit 15 in the System Control Module Interrupt Configuration 10 register (A090 014C) to enable the INTA# interrupt from PCI Device #1.

Configuration with NS9775 as PCI device

Figure 2 shows a sample system consisting of NS9775 and one external PCI device. NS9775 is a device on the PCI bus in this system. Use this diagram as a guide to the configuration sequences discussed in this section.

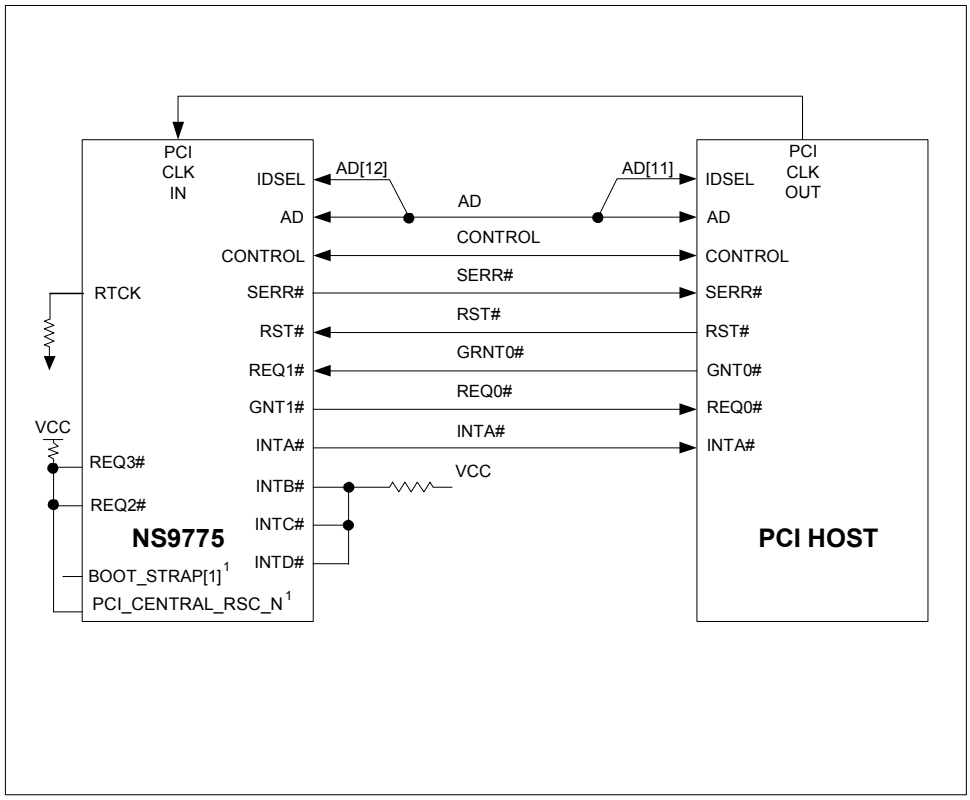

Figure 2: Sample system with NS9775 as PCI device

1 The pin can remain unconnected because the internal pulldown is configured to the correct logic level.

System characteristics

- NS9775
  - Mapped to 256 MB window in PCI memory space, through Base Address Register 0
  - Does not provide PCI central resource functions
  - Does not provide PCI arbiter
  - Does not provide PCI interrupt controller
  - Device 1 on PCI bus
  - Single interrupt
  - PCI master
  - PCI ID information changed for OEM
- External PCI Device #1
  - Mapped to 128 MB window in PCI memory space, using Base Address Register 0
  - Mapped to 64 KB window in PCI IO space, using Base Address Register 1
  - Single function PCI device
  - PCI master
  - Single interrupt
  - Device 0 on PCI bus
  - Provides PCI interrupt controller and arbiter
  - Provides PCI central resource functions, including PCI clock
- System
  - PCI host memory mapped to `0xF000_0000-0xF7FF_FFFF` in PCI memory space (128 MB)
  - NS9775 memory mapped to `0x1000_0000-0x1FFF_FFFF` in PCI memory space (256 MB)
  - PCI host IO mapped to `0x2000_0000-0x2000_FFFF` in PCI IO space (64 KB)
  - NS9775 PCI interrupt connected to `INTA#` of PCI host
  - PCI host accesses to NS9775 mapped to `0x3000_0000-0x3FFF_FFFF` in NS9775's memory space
  - 33 MHz `PCI_CLK`

PCI configuration sequence

In this configuration, RST# acts like a chip for the rest of NS9775. All of the blocks in NS9775 are reset by RST#.

The PCI configuration sequence includes these steps, explained in detail on the following pages:

- Configure the bridge-specific registers
- Configure the PCI host
- Configure the Bridge PCI Configuration registers using the host
- Final configuration
- Post-initialization operation

Configure the bridge-specific registers

Configuring the bridge-specific registers pertains to initialization that must be completed within 2^{25} PCI clocks of the RST# being negated. This is the time allowed from when RST# is negated until the first configuration cycle on the PCI bus. NS9775 initiates this step.

1. Write 0x0000_0010 to the PCI Miscellaneous Support register.
   - Enables the Base Address 0 register, which decodes a 256 MB window in PCI memory space by setting EN_BAR0 (bit 4 = 1). All other Base Address registers are disabled.
2. Write 0x7F3D_271F to the PCI Configuration 0 register.
   - Sets PCI device ID to OEM's code of 0x7F3D.
   - Sets PCI vendor ID to OEM's code of 0x271F.
3. Write 0x0600_0001 to the PCI Configuration 1 register.
   - Sets PCI class code to 0x060_0000 for host/PCI bridge.
   - Sets PCI revision ID to OEM's code of 0x1.
4. Write 0x0D0F_0003 to the PCI Configuration 2 register.
   - Sets PCI subsystem ID to OEM's code of 0x0D0F.
   - Sets PCI sub-vendor ID to OEM's code of 0x0003.

5 Write 0x0014_0101 to the PCI Configuration 3 register.
- Sets PCI Interrupt Pin register to 0x01, which indicates that NS9775 drives its PCI interrupt on INTA#.
- Sets PCI MIN_GNT (minimum grant value), which indicates the amount of time that it takes the NS9775 to burst data on the PCI bus at 33 MHz in increments of 250 ns, to 0x01 (that is, 250 ns). Because the NS9775 is configured to burst eight words at a time, the data portion of a burst takes 240ns at 33 MHz.
- Sets PCI MAX_LAT (maximum latency) field, which indicates how often NS9775 should be serviced in units of 250 ns, to 0x14 (that is, five microseconds).

Note: The value used here is application-specific and will vary according to the rate at which the NS9775 is expected to transfer data to/from the system. The value 0x14 is used here as an example only.

6 Write 0x7878_7878 to the PCI Bridge AHB to PCI Memory Address Translate 0 register.
- Maps the accesses to the lower 128 MB of the NS9775's PCI memory window (0x9000_0000-0x87FF_FFFF) to the 128 MB window where the PCI host is located (0xF000_0000-0xF7FF_FFFF) by setting PALT0VAL, PALT1VAL, PALT2VAL, and PALT3VAL to 0x78.

7 Write 0x0000_0200 to the PCI Bridge AHB to PCI IO Address Translate register.
- Maps the accesses to the lower 64 KB of the NS9775's PCI I/O window (0xA000_0000-0xA000_FFFF) to the 64 KB window where the I/O space for PCI host is located (0x2000_0000-0x2000_FFFF) by setting PALT8VAL to 0x200.

8 Write 0x0000_0003 to the PCI Bridge PCI to AHB Memory Address Translate 0 register.
- Maps PCI accesses to NS9775 to a 256 MB window in NS9775's memory space located at 0x3000_0000-0x3FFF_FFFF by setting MALT0VAL to 0x3. All other values in this register are not used because only Base Address Register 0 is enabled in the PCI Miscellaneous Support register.

9 Write 0x0000_0003 to the PCI Bridge Address Translation Control register.

a Enable PCI to AHB address translation by setting MALT_EN.

b Enable AHB to PCI address translation by setting PALT_EN.

Configure the PCI host

These steps show the initialization sequence that the PCI host must perform on its own PCI Configuration registers to meet the system definition requirements (see "System characteristics," beginning on page 30).

1 Write `0x8000_0010` to PCI CONFIG_ADDR space.

   a Set up the Configuration Address Port register in the host to access the PCI Base Address 0 register in the bridge. Note that the host is accessed as `DEVICE_NUMBER 0`.

   Write `0xF000_0000` PCI CONFIG_DATA space.

   b Map 128 MB memory window supported by PCI host to `0xF000_0000`-`0xF7FF_FFFF` by setting the base address using bits [31:27] of the PCI Base Address 0 register.

2 Write `0x8000_0014` to PCI CONFIG_ADDR space.

   a Set up the Configuration Address Port register in the host to access the PCI Base Address 1 register in the bridge.

   Write `0x2000_0001` to PCI CONFIG_DATA space.

   b Map 64 KB I/O window supported by the PCI host to `0x2000_0000`-`0x2000_FFFF` by setting the base address using bits [31:16] of the PCI Base Address 1 register.

3 Write `0x8000_000C` to PCI CONFIG_ADDR space.

   a Set up the Configuration Address Port register in the host to access the PCI Latency Timer and Cache Size configuration registers in the host.

   Write `0x0000_FF00` to PCI CONFIG_DATA space.

   b Initialize the PCI Latency Timer register to `0xFF` to allow the PCI host to stay on the bus for up to 255 PCI clocks when it is bursting data on the PCI bus (bits[15:08] = `0xFF`). This is necessary because the latency resulting from the reads to the bridge can be long due to the AHB bus arbitration within NS9775.

   c Initialize the PCI Cache Size register to 0 (bits [07:00] = `0x00`).

Configure the Bridge PCI Configuration registers using the host

These steps show the initialization sequence that the PCI host must perform on the NS9775 PCI configuration registers to meet the system requirements (see "System characteristics," beginning on page 30).

The PCI host may query the different PCI configuration registers in the bridge to determine its device ID, memory requirements, and the like. These types of accesses are application-specific and do not change the operation of the NS9775. Although the PCI host accesses NS9775 as Device #1 on the PCI bus, any internal accesses of the PCI configuration registers by NS9775 use Device 0 because these accesses do not exit NS9775.

1. Write `0x8000_0810` to PCI CONFIG_ADDR space.
   - **a** Set up the Configuration Address Port register in the host to access the PCI Base Address 0 register in the bridge. Note that the bridge is accessed as `DEVICE_NUMBER 1`.

   Write `0x1000_0000` to PCI CONFIG_DATA space.
   - **b** Initialize the PCI Base Address 0 register to allow NS9775 to respond to a 256 MB window in PCI memory space starting at `0x1000_0000`. All other Base Address registers are disabled.
2. Write `0x8000_080C` to PCI CONFIG_ADDR space.
   - **a** Set up the Configuration Address Port register in the host to access the PCI Latency Timer and Cache Size configuration registers in the bridge.

   Write `0x0000_FF00` to PCI CONFIG_DATA space.
   - **b** Initialize the PCI Latency Timer register to `0xFF` to allow NS9775 to stay on the bus for up to 255 PCI clocks when it is bursting data on the PCI bus (bits[15:08] = `0xFF`).
   - **c** Initialize the PCI Cache Size register to 0 (bits [07:00] = `0x00`). This field has no effect on the operation of the bridge and should always be set to `0x00`.

Final configuration

The final configuration is where the PCI-to-AHB bridge is enabled for PCI bus operations. All steps are initiated by the PCI host. **It is important that you follow these steps exactly in the order in which they are shown.**

1 Write `0x8000_0804` to PCI CONFIG_ADDR space.

a Set up the Configuration Address Port register in the bridge to access the PCI Status and Command configuration registers in the bridge. Note that the bridge is accessed as `DEVICE_NUMBER 1`.

Write `0x0000_0142` to PCI CONFIG_DATA space.

b Initialize the PCI Command register as follows:

i Disable the bridge response to PCI I/O accesses (bit 0 = 0).

ii Enable the bridge response to PCI memory accesses (bit 1 = 1).

iii Disable the bridge's ability to act as a PCI master (bit 2 = 0). Although the bridge can be a master, it is too early to enable this capability because the NS9775 may not be fully initialized at this point.

iv Disable the bridge's ability to generate memory write and invalidate command (bit 4 = 0). This bit has no effect on bridge operation and should always be 0.

v Enable the bridge to assert PERR# when it detects a parity error (bit 6 = 1).

vi Enable the bridge driving SERR# (bit 8 = 1).

vii Be sure bits 3, 5, 7, and 9 are hardwired to 0 in the bridge.

2 At this point, the PCI host waits for NS9775's internal initialization to complete and then performs any additional initialization required from the host.

3 Write `0x8000_0804` to PCI CONFIG_ADDR space.

– Set up the Configuration Address Port register in the host to access the PCI Status and Command configuration registers in the bridge.

Write `0x0000_0347` to PCI CONFIG_DATA space.

Initialize the PCI Command register as in Steps 1b(i) through 1b(vii), with the following exception:

- Enable the bridge as a bus master (bit 2 = 1) now that the bridge has been initialized.

4 The PCI host can now internally enable the PCI interrupt from the NS9775 that drives INTA#.

5 Write 0x0000_F901 to the PCI Bridge Interrupt Enable register.
   – This enables all interrupts from the bridge within NS9775. The number of interrupts enabled can be decreased depending on the application.

Post-initialization operation

The NS9775 can generate a PCI interrupt to the PCI host by setting the INTA2PCI bit in the PCI Miscellaneous Support register.

Configuration with unused NS9775 PCI interface

In applications that do not use the PCI-to-AHB bridge, the system must guarantee that the PCI inputs to the NS9775 do not float. These signals require external pullups:

- FRAME#
- TRDY#
- IRDY#
- DEVSEL#
- STOP#
- SERR#
- PERR#
- LOCK#[1]
- INTA#
- INTB#
- INTC#
- INTD#
- All REQ# inputs to NS9775
- IDSEL

1 The NS9775 does not have a LOCK# pin associated with it. If any PCI device in the system uses the LOCK# signal, the signal must have a pullup resistor.

In addition, the AD[31:00], C/BE[03:00], and PAR signals must be pulled up or down by either external resistors or the NS9775. Figure 3 shows a configuration in which NS9775 can be made to drive these signals.

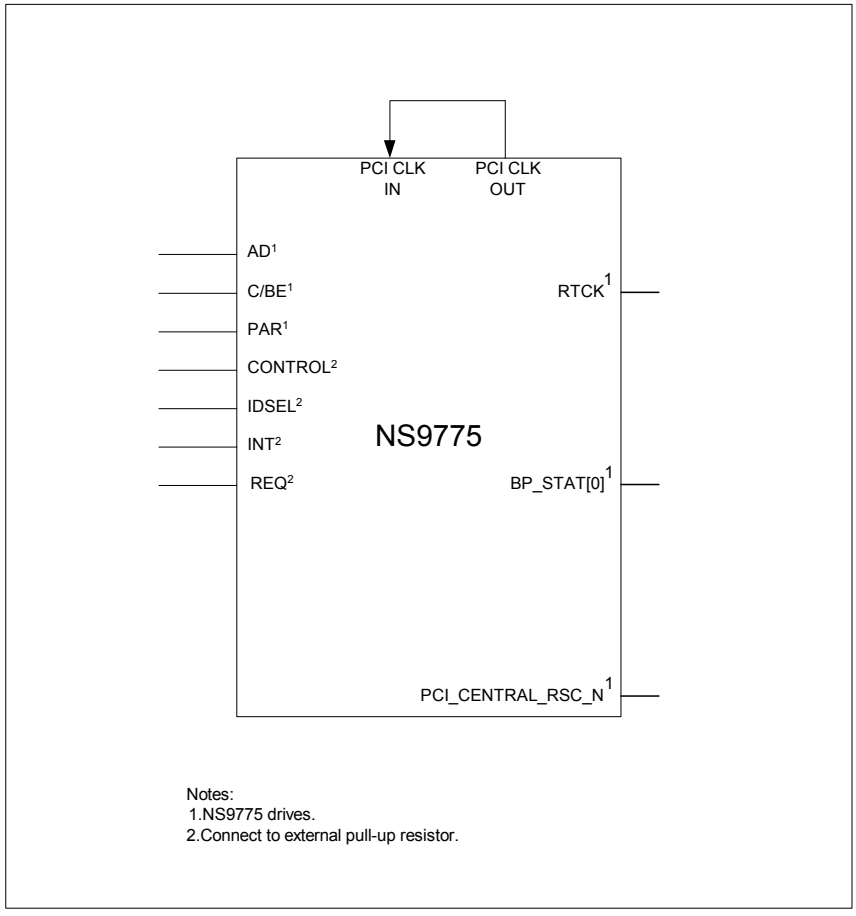

Figure 3: Sample system with NS9775 driving unused PCI interface

1 These pins can remain unconnected because internal resistors configure to correct logic level.

In this example, the NS9775 is configured to provide the PCI central resource functions, since PCI_CENTRAL_RESOURCE_N is tied low by its internal pulldown resistor. This allows the NS9775 to drive AD[31:00], C/BE[03:00], and PAR low during reset. The NS9775 also is configured to use the internal PCI arbiter, since RTCK is tied high by its internal pullup resistor. This allows the NS9775 to drive AD[31:00], C/BE[03:00], and PAR after reset because the internal arbiter parks the PCI bus on the PCI-to-AHB bridge.

PCI configuration

1. Reset the bridge and PCI arbiter after the system reboots by clearing the PCI bit in the Reset and Sleep Control register to a 0 (this register is in the System Control module).This is necessary because the PCI bit defaults to 1. By this point, the internal PCI arbiter has already parked the bus on the bridge, and AD[31:00], C/BE[03:00], and PAR are being driven. The signals will be driven low after the bridge and the PCI arbiter are reset and after the clock is turned off (if the following step is done).

2. Turn off the PCI clock to the bridge and PCI arbiter by clearing the PCI and PCI Arbiter bits in the Clock Configuration register to 0 (these registers are in the System Control module).

   You can eliminate this step, which is done simply to save power.

Configuring NS9775 for CardBus support

Figure 4 shows how the NS9775 is configured for CardBus applications.

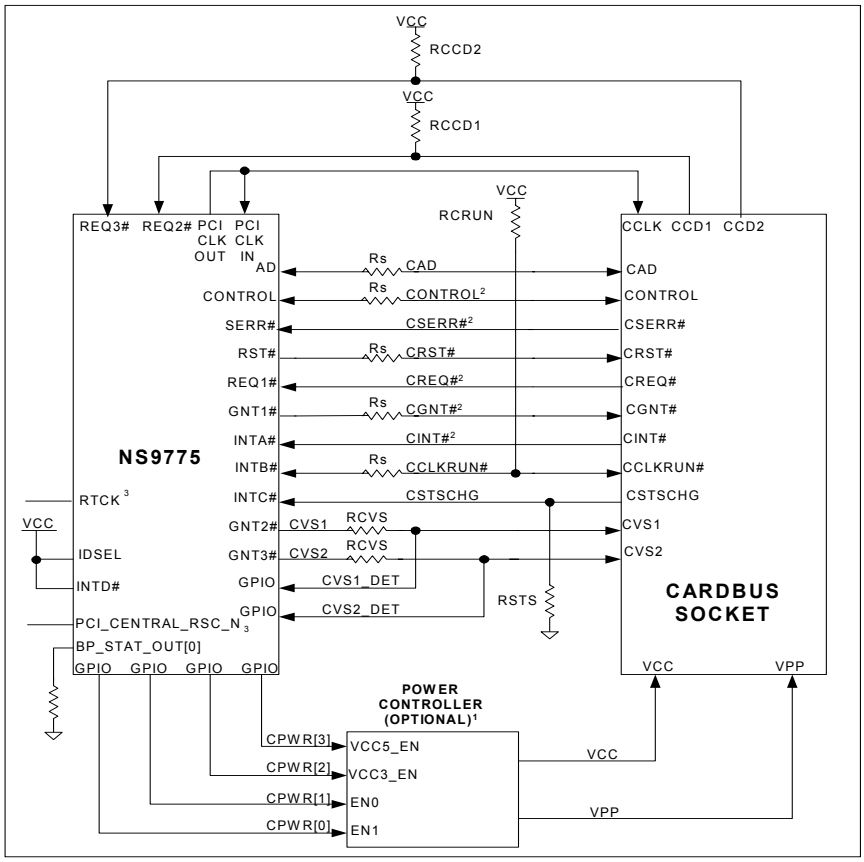

Figure 4: CardBus system connections to NS9775

1 The power controller is required only for applications that support hot-insertion and hot-removal of the CardBus card. This requires additional components to isolate the NS9775 from CardBus.

2 The system must provide external pullups per the standard PCI specification. CAD, C/BE, and PAR do not require pullups.

3 These pins can be left unconnected because internal resistors provide the correct state for this application.

Important: Note that in cases where NS9775 provides the PCI clock, the PCI clock connection to the NS9775 must still be made external to the NS9775; that is, connect PCI_CLK_OUT to PCI_CLK_IN. This is done to minimize the clock skew between the NS9775 and external PCI devices.

Simple configuration for powered socket

This configuration is for a CardBus application where the external CardBus device is already inserted in the socket before applying power to the system in which the NS9775 resides. In addition, power is applied to both the NS9775 and the CardBus socket at the same time, so there is no *hot-insertion*.

1. Determine whether NS9775 is connected to a CardBus or PCI bus.
   a. Read the BP_STAT_OUT[0] field (bootstrap initialization). If the value is 0, NS9775 is connected to an external CardBus.
2. **Optional.** Write 0x2000_0000 to the Cardbus Miscellaneous Support register to set the V3_SKT bit. This results in setting the V3_SKT bit in the CardBus Socket Present State register to indicate that the socket can support 3.3V cards only.
3. Determine whether a card is present in the CardBus socket.
   a. Write 0x0000_0000 (default value) to the CardBus Miscellaneous Support register so both the CVS2 and CVS1 fields are 0. This forces the NS9775 CVS2 and CVS1 pins to 0.
   b. Wait ~1usec to allow the CCD1 and CCD2 pins to be sampled by NS9775 hardware.
   c. Read the CardBus Miscellaneous Support register to determine whether a card is present (that is, both CCD1 and CCD2 are set to 0).
4. If a card is present in the socket, write 0x2108_0040 to the CardBus Miscellaneous Support register to effect the following:
   a. Set the V3_CARD bit to a 1, which sets the V3_CARD bit in the CardBus Socket Present State register to 1 to indicate that the card is a 3.3V card.

      This is an optional step, and does not affect the operation of the external CardBus device.

b Set the CB_CARD bit to a 1, which sets the CB_CARD bit in the CardBus Socket Present State register to 1 to indicate that the card is a CardBus type.

This is an optional step, and does not affect the operation of the external CardBus device.

c Clear the CCD1 and CCD2 bits to a 0 bit, which clears the CCD1 and CCD2 bits in the CardBus Socket Present State register to a 0 to indicate that a card is present in the socket.

This is an optional step, and does not affect the operation of the external CardBus device.

d Set CCLKRUN_EN to a 1 bit, which asserts the `CCLKRUN#` signal on the CardBus.

5 Follow a configuration sequence similar to the sequence described in "Configuration with NS9775 as PCI Host," beginning on page 20. An additional configuration register in the external CardBus device — the CardBus CIS Pointer register — is used as a pointer to the card information structure (CIS) found on all CardBus cards. The CIS provides more information about the card.

6 Enable the `CSTSCHG` interrupt from the card after the external CardBus device card has been initialized.

a Write `0x0000_0001` to the CardBus Socket mask register.

Memory Controller

CHAPTER 4

This chapter provides sample driver configurations for the memory controller. Use these samples as guidelines for developing your own drivers.

Keep in mind that this is only one way to configure the memory controller module.

Generic SDRAM initialization

On power-on-reset, RESET_N, software must initialize the memory controller and each of the dynamic memories connected to the controller. This section provides a sample initialization procedure.

1. Wait 100ms after the power is applied and the clocks have stabilized.
2. Set the SDRAM initialization (I) value to NOP in the Dynamic Control register; this automatically issues a NOP command to the SDRAM memories.
3. Wait 200ms.
4. Set the SDRAM initialization (I) value to PALL in the Dynamic Control register. This automatically issues a *precharge all* instruction (PRE_ALL) to the SDRAM memories. The precharge all instruction precharges all banks and places the device into *all banks idle* status.
5. Perform a number of refresh cycles by writing a 1 into the Dynamic Refresh register. This provides a memory refresh every 16 AHB clock cycles.
6. Wait until eight SDRAM refresh cycles have occurred (128 AHB clock cycles).
7. Program the operational value into the Dynamic Refresh register.
8. Program the operational value into the Dynamic RasCas (latency) register.
9. Program the operational values into the Dynamic Configuration register. The buffers must be disabled during initialization.
10. Set the SDRAM initialization value (I) to MODE in the Dynamic Control register.
11. Program the SDRAM memories mode register. The mode register allows these parameters to be programmed:

Burst length	4 for a 32-bit wide external databus, or 8 for a 16-bit wide external databus
Burst type	Sequential
CAS latency	Depends on operating frequency
Operating mode	Standard operation
Write burst mode	Programmed burst length

- A read transaction from the SDRAM memory programs the mode register.
- The transfer address contains the value to be programmed.
- The mapping from AHB address bus, HADDR, to the SDRAM memories address lines depends on the address mapping value selected in the Dynamic Configuration register.
- The row address bits contain the value to be programmed.
- The bank select signals BA0 and BA1 must both be 0 to program the mode register.

Note that you must use the AHB memory port to perform this transaction. When initializing the memory device, the appropriate chip select must be activated. Depending on the AHB decoder address map, the address programmed might require modification.

12 Set the SDRAM initialization value (I) to NORMAL in the Dynamic Control register.

13 Enable the buffers in the Dynamic Configuration register. The SDRAM is now ready for normal operation.

4 MBx16 SDRAM initialization

Use this procedure to initialize two SDRAM devices — 64 MB and 4 MB x 16, speed grade -8E, configured to provide a 32-bit bus. HCLK and CLK are 100 MHz.

1 Wait 100ms after the power is applied and the clocks have stabilized.

2 Set the SDRAM initialization (I) value to NOP in the Dynamic Control register; this automatically issues a NOP to the SDRAM memories.

3 Set the SDRAM initialization (I) value to PALL in the Dynamic Control register. This automatically issues a *precharge all* instruction (PRE-ALL) to the SDRAM memories. The precharge all instruction precharges all the banks and places the device into the *all banks idle* state.

4 Perform a number of refresh cycles, by writing a 2 in the Dynamic Refresh register. This provides a memory refresh every 32 AHB clock cycles.

5 Wait until two SDRAM refresh cycles have occurred (64 AHB clock cycles).

6 Program the operational value into the Dynamic Refresh register. This device requires a memory refresh every 15.625µs. With a 100 MHz HCLK, then, the Dynamic Refresh register must be programmed with the following value:

```
(15.625µs x 100 MHz)/16 = 97
```

7 Program the operational value into the Dynamic RASCAS (latency) register. The -8E speed grade devices support CAS latency 2 at 100 MHz. Therefore, the value 0x0202 must be programmed into this register.

8 Program the operational values into the Dynamic Configuration register. The buffers must be disabled during initialization. For this memory device, set the fields as shown:

Memory device (MD)	SDRAM (00)
Address mapping (AM)	32-bit bus, 64 MB, 4 MB x 16 devices, RBC mapping (10000101)
Buffer enable (B)	Disabled (0)
Write protect (P)	Writes not protected (0)
Column width (CW)	8 (010)
Number of banks (NB)	Four banks (1)
Row width (RW)	12 (01)

The value is 0x14804280.

Note that you must use the AHB memory port to perform this transaction. When initializing the memory device, the appropriate chip select must be activated. Depending on the AHB decoder address map, the address programmed might require modification.

9 Set the SDRAM initialization (I) value to MODE in the Dynamic Control register.

10 Program the SDRAM memories mode register. The mode register enables these parameters:

Burst length	4 (for 32-bit databus)
Burst type	Sequential
CAS latency	2 (for -8E device @ 100 MHZ)
Operating mode	Standard operation
Write burst mode	Programmed burst length
Reserved (0)	0

– The value required to program the SDRAM mode register is `0x22`.

– The `HADDR` to SDRAM memory address mapping is 32-bit 64M SDRAM (4M x 16, RBC), and is shown in this table:

Output address	Memory device connections	AHB address to row address	AHB address to column address
14	BA1	11	11
13	BA0	1	10
12	---	---	---
11	11	23	---
10	10/AP	22	AP
9	9	21	---
8	8	20	---
7	7	19	9
6	6	18	8
5	5	17	7
4	4	16	6
3	3	15	5
2	2	14	4
1	1	13	3
0	0	12	2

- The SDRAM memory row address bits are mapped to HADDR[23:12].
- The SDRAM memory bank address bits are mapped to HADDR[11:10].
- The address accessed is 0x22000.

Note that you must use the AHB memory port to perform this transaction. When initializing the memory device, the appropriate chip select must be activated. Depending on the AHB decoder address map, the address programmed might require modification.

11 Set the SDRAM initialization (I) value to NORMAL in the Dynamic Control register.

12 Enable the buffers in the Dynamic Configuration register. The SDRAM is now ready for normal operation.

Low-power SDRAM initialization

Use this procedure to initialize 8 MB x 16 devices configured to provide a 16-bit bus. HCLK and CLK are 100 MHz.

1 Wait 100ms after the power is applied and the clocks have stabilized.

2 Set the SDRAM initialization (I) value to PALL in the Dynamic Control register. This automatically issues a *precharge all* instruction (PRE-ALL) to the SDRAM memories. The precharge all instruction precharges all the banks and places the device into the *all banks idle* state.

3 Perform a number of refresh cycles, by writing a 2 into the Dynamic Refresh register. This provides a memory refresh every 32 AHB clock cycles.

4 Wait until eight SDRAM refresh cycles have occurred (256 AHB clock cycles).

5 Program the operational value into the Dynamic Refresh register. This device requires a memory refresh every 16µs. With a 100 MHz HCLK, the Dynamic Refresh register must be programmed with the following value:

```
(16µs x 100 MHz)/16 = 97
```

6 Program the operational value into the Dynamic RasCas (latency) register. The -8 speed grade devices support CAS latency 2 at 100 MHz operation. The value 0x0202 must be programmed into the register.

7 Program the operational values into the Dynamic Configuration register. The buffers must be disabled during initialization. For this memory device, set the fields as shown:

Memory device (MD)	Low-power SDRAM (01)
Address mapping (AM)	16-bit bus, 128 Mb, 8M x 16 devices, BRC mapping (00101001)
Buffer enable (B)	Disabled (0)
Write protect (P)	Writes not protected (0)
Column width (CW)	9 (011)
Number of banks (NB)	Four banks (1)
Row width (12 (01)	

The value is `0x14C01488`.

Note that you must use the AHB memory port to perform this transaction. When initializing the memory device, the appropriate chip select must be activated. Depending on the AHB decoder address map, the address programmed might require modification.

8 Set the SDRAM initialization value (I) to MODE in the Dynamic Control register.

9 Program the SDRAM memories mode register. The mode register enables these parameters:

Burst length	8 (for 32-bit databus)
Burst type	Sequential
CAS latency	2 (for -8E device @ 100 MHz)
Operating mode	Standard operation

- The value `0x23` must be programmed in the low-power SDRAM mode register.
- The `HADDR` to SDRAM memory address mapping is 16-bit 128 Mb SDRAM (8M x 16, BRC), and is shown in this table:

Output address	Memory device connections	AHB address to row address	AHB address to column address
14	BA1	23	23
13	BA0	22	22
12	---	---	---
11	11	21	---
10	10/AP	20	AP
9	9	19	---
8	8	18	9
7	7	17	8
6	6	16	7
5	5	15	6
4	4	14	5
3	3	13	4
2	2	12	3
1	1	11	2
0	0	10	This bit is controlled by the SDRAM controller.

10 Program the low-power SDRAM memories extended mode register. The mode register enables these parameters:

Partial array self-refresh	All banks
Temperature compensated self-refresh	70°C

- The bank select signals `BA1` and `BA0` must be 1, 0 to select the extended mode register.
- A read transaction from the SDRAM memory programs the mode register.
- The transfer address contains the value to be programmed.
- The mapping from the AHB address bus, `HADDR`, to the SDRAM memories address lines depends on the address mapping value selected in the

Dynamic Configuration register (in this case, the value is 16-bit, 128, 8Mx16, BRC).

- The row address bits contain the value to be programmed.
- The value `0x00` is required to program the low-power SDRAM extended mode register.
- The `HADDR` to SDRAM memory address mapping is 16-bit, 128 Mb SDRAM (8Mx16, BRC).
- The SDRAM memory row address bits are mapped to `HADDR[21:10]`. The SDRAM memory bank address bits are mapped to `HADDR{23:22]`. The address to be accessed is `0x800000`. (See the address mapping table in Step 9, on page 49.)

Note that you must use the AHB memory port to perform this transaction. When initializing the memory device, the appropriate chip select must be activated. Depending on the AHB decoder address map, the address programmed might require modification.

11 Set the SDRAM initialization value (I) to NORMAL in the Dynamic Control register.

12 Enable the buffers in the Dynamic Configuration register. The SDRAM is now ready for normal operation.

BBus DMA Configurations

CHAPTER 5

This chapter provides sample driver configurations for the BBus DMA module. Use these samples as guidelines for developing your own drivers.

Keep in mind that this is only one way to configure BBus DMA.

Configuring BBus DMA drivers

Configuration example #1

System characteristics

- DMA channel #1.
- Fly-by write transfer from serial controller B to system memory.
- Buffer descriptor pool contains two entries.

Configuration sequence

1. Configure PORT B Serial Controller module, as described in the Serial Controller chapter in the *NS9775 Hardware Reference*.
2. Set up the first buffer descriptor in memory:
   a. Write `0x0020_0000` to `0x0001_0000`.
   b. Write `0x0000_0400` to `0x0001_0004`.
   c. Write `0x0000_0000` to `0x0001_0008`.
   d. Write `0x0000_0000` to `0x0001_000C`.
      i. Set data buffer address to `0x0020_0000`.
      ii. Set data buffer length to 1K bytes.
      iii. Set W = 0.
      iv. Set I = 0.
      v. Set L = 0.
      vi. Set F = 0.

3. Set up the second buffer descriptor in memory:
   a. Write 0x0020_0400 to 0x0001_0010.
   b. Write 0x0000_0400 to 0x0001_0014.
   c. Write 0x0000_0000 to 0x0001_0018.
   d. Write 0x8000_0000 to 0x0001_001C.
      i. Set data buffer address to 0x0020_0400.
      ii. Set data buffer length to 1K bytes.
      iii. Set W = 1
      iv. Set I = 0.
      v. Set L = 0.
      vi. Set F = 0.
4. Write 0x0001_0000 to DMA Channel 1 buffer descriptor pointer.
   a. Point DMA channel 1 at its first buffer descriptor.
5. Write 0x01C0_0000 to DMA Channel 1 Status/Interrupt Enable register.
   a. Enable NCIP interrupt generation by setting the NCIE bit.
   b. Enable ECIP interrupt generation by setting the ECIE bit.
   c. Enable NRIP interrupt generation by setting the NRIE bit.
6. Write 0x8200_0000 to the DMA Channel 1 Control register.
   a. Enable the DMA channel by setting the CE bit.
   b. Define the burst size by setting the BTE bit.
7. Process buffer close interrupts as data moves through the system.

Configuration example #2

System characteristics

- DMA channel #2.
- Fly-by read transfer from system memory to serial controller B.
- Buffer descriptor pool contains two entries.

Configuration sequence

1. Configure PORT B Serial Controller module, as described in the Serial Controller chapter in the *NS9775 Hardware Reference*.
2. Set up the first buffer descriptor in memory:
   - **a** Write `0x0080_0000` to `0x0004_0000`.
   - **b** Write `0x0000_0400` to `0x0004_0004`.
   - **c** Write `0x0000_0000` to `0x0004_0008`.
   - **d** Write `0x0000_0000` to `0x0004_000C`.
     - **i** Set data buffer address to `0x0080_000`0.
     - **ii** Set data buffer length to 1K bytes.
     - **iii** Set W = 0.
     - **iv** Set I = 0.
     - **v** Set L = 0.
     - **vi** Set F = 0.
3. Set up the second buffer descriptor in memory:
   - **a** Write `0x0080_0400` to `0x0004_0010`.
   - **b** Write `0x0000_0400` to `0x0004_0014`.
   - **c** Write `0x0000_0000` to `0x0004_0018`.

d Write 0x8000_0000 to 0x0004_001C.

i Set data buffer address to 0x0080_0400.

ii Set data buffer length to 1K bytes.

iii Set W = 1.

iv Set I = 0.

v Set L = 0.

vi Set F = 1

4 Write 0x0004_0000 to DMA channel 2 buffer descriptor pointer.

a Point DMA channel 2 at its first buffer descriptor.

5 Write 0x01C0_0000 to DMA Channel 2 Status/Interrupt Enable register.

a Enable NCIP interrupt generation by setting the NCIE bit.

b Enable ECIP interrupt generation by setting the ECIE bit.

c Enable NRIP interrupt generation by setting the NRIE bit.

6 Write 0x8600_0000 to DMA Channel 2 Control register.

a Enable the DMA channel by setting the CE bit.

b Define fly-by read operation by setting the MODE bit.

c Define the burst size by setting the BTE bit.

7 Process buffer close interrupts as data moves through the system.

IEEE 1284

CHAPTER 6

This chapter provides sample driver configurations for the IEEE 1284 module for these modes:

- Direct access
- Compatibility mode, direct access
- Byte/nibble mode, using direct access compatibility
- DMA mode
- Compatibility mode, DMA support
- Byte/nibble mode, using DMA support compatibility

Use these samples as guidelines for developing your own drivers. Keep in mind that this is only one way to configure IEEE 1284.

Direct access

Perform these steps before the steps for compatibility mode or byte/nibble mode:

1. Write to the Master Reset register in the BBus Utility module:
   a. Bit [8]: Clear BBus utility reset.
2. Write to the Interrupt Enable register in the BBus Bridge module:
   a. Bit [31]: Enable BBus bridge interrupt.
   b. Bit [12]: Enable BBus utility interrupt.
   c. Bit [11]: Enable 1284 interrupt.
3. Write to GPIO Configuration Register #7 in the BBus Utility module:
   a. Bits [3:0]: Allocate 1284 control signal.
   b. Bits [7:4]: Set PLH to be an output at this time.
4. Write to the Port Control register:
   a. Bits [7:0]: Drive pins to a 1 during initialization.
5. Write to GPIO Configuration #5 in the BBus Utility module:
   a. Bits [31:0]: Allocate 1284 control signals.
6. Write to GPIO Configuration Register #1 in the BBus Utility module:
   a. Bits [27:12]: Allocate 1284 control signals.
7. Write to GPIO Configuration Register #6 in the BBus Utility module:
   a. Bits [31:16]: Allocate 1284 control signals.
8. Write to the Endian Configuration register in the BBus Utility module:
   a. Bit [6]: Configure AHB to be big endian.
9. Write to the Master Reset register:
   a. Bit [6]: Clear 1284 reset.

Note: Each gpio signal has four corresponding bits in a GPIO configuration register. 1284 functionality is selected by setting these bits to (`0x1`).

Compatibility mode, direct access

The compatibility mode, direct access programming sequence receives data in compatibility mode and allows negotiation into nibble, byte, and ECP modes.

1. Write to the General Configuration register:
   - **a** Bits [3:0]: Direct CPU access, DMA disabled
   - **b** Bits [5:4]: Reverse data FIFO threshold is 25-28 bytes
   - **c** Bits [9:8]: Forward data FIFO threshold is 4 bytes
   - **d** Bits [11:10]: Forward command FIFO threshold id 4 bytes
   - **e** Bit [13]: PLH signal deasserted
   - **f** Bits [14]: All forward data stored in “data” FIFO
2. Write to the InterruptStatusAndControl register:
   - **a** Bit [17]: Enable Vcm1289Interrupt1
   - **b** Bit [19]: Enable FwDatFifoRdyInterrupt
   - **c** Bit [21]: Set the maximum buffer size `(0xFFFF)`. *This field is for DMA only.*
   - **d** Bit [23]: Enable FwDatFifoByteGap
3. Write to FwDatDmaControl register:
   - **a** Bits [15:0]: Set the gap timer to 2048 BBus clock cycles. This generates an interrupt telling the CPU that there is data in the forward data FIFO. *This field is used for DMA and direct access modes.*
   - **b** Bits[31:16]: Set the maximum buffer size `(0xFFFF)`. *This field is for DMA only.*
4. Write `0x0000_0001` to grn.
   - **a** Bits [7:0]: Write a value of 1 to the granularity counter. This is necessary to initialize the 1284 core.
5. Repeat Step 4 — Write a value of 1 to the granularity counter.
6. Write `0x0000_0001` to feb.
   - **a** Bit [0]: Set to a 1.

7. Write to fei:
   a. Bit [1]: Enable interrupt when the host initiates a negotiation phase.
8. Write to ecr:
   a. Bit [6]: Enable reverse request.
   b. Bit [7]: Set to a 1.
9. Write to grn:
   a. Bits [7:0]: Write a value of 25 to the granularity counter. This causes the maximum time between slave cycles to be 25 BBus clock cycles.
10. Repeat Step 9 — Write a value of 25 to the granularity counter.
11. Write to GPIO Configuration Register #7:
   a. Bits [7:4]: Allocate the 1284 control signal.
12. Write to fea:
   a. Bit [0]: enable printer port.
13. Write to fem:
   a. Bit [2]: Enable auto-negotiate mode.
   b. Bit [4]: Enable auto-transfer mode.
   c. Bit [5]: Enable SPP mode.
   d. Bit [6]: Enable ECP mode.
14. Write to the General Configuration register:
   a. Bit [13]: PLH signal asserted; the core is ready for traffic.

The NS9775 is now configured to accept forward traffic in compatibility mode. The NS9775 is also configured to auto-negotiate byte, nibble, and ECP modes.

Steps 15–18 show data being received in compatibility mode.

15. Wait for a 1284 interrupt.
16. Read the InterruptStatusAndControl register to determine whether data is ready.
   – Bit [3]: If set, forward data from the host is ready to be read.

17 Read the FIFO Status register to determine how much data has been received.
   - Bit [3]: FwDatFifoReady, if set, then forward data is ready to be read.
   - Bit [4]: FwDatFifoAlmostEmpty, if set, then only 1–4 bytes are ready; only perform one read.
   - Bit [5]: FwDatFifoEmpty, if cleared, then the forward data FIFO is not empty.
   - Bits [7:6]: FwDatFifoDepthRemain: Determines how many bytes should be read in the next read, if the FIFO is not empty.

18 Read the FwDatFifoReadReg register to read the data bytes from the host.

19 Write to the InterruptStatusAndControl register.
   a Write a 1 to bit[3] to clear the FwDatFifoRdyInterrupt bit.

Byte/Nibble mode, direct

Byte and nibble modes perform *reverse transfers*; that is, they send data to the host. The configuration steps shown in "Compatibility mode, direct access" (on page 61) enable the NS9775 to negotiate to byte/nibble modes.

This programming sequence illustrates a negotiation to byte/nibble mode and a reverse transfer:

1 Enable the NS9775 as described in Steps 1–13 of "Compatibility mode, direct access," beginning on page 61.

2 Wait for a negotiation start interrupt. This is determined by reading the interrupt status registers as described in Steps 3–5.

3 Read the InterruptStatusAndControl register.
   - If bit [1] (peripheral controller interrupt 1) is set, a 1284 peripheral interrupt has occurred.

4 Read the sti register.
   - If bit [1] (negotiation start interrupt detect) is set, the host has started a negotiation phase.

5 Read the exr register to determine which mode the host is requesting. Valid values are:

- 0x00 — Nibble mode
- 0x01 — Byte mode
- 0x04 — Device ID, nibble mode
- 0x05 — Device ID, byte mode
- 0x14 — Device ID, ECP
- 0x15 — Device ID, ECP with RLE
- 0x10 — ECP mode
- 0x30 — ECP mode with RLE

If the value is 0x00–0x05, reverse data can be transferred to the host. The procedure is the same for nibble and byte modes (as far as the CPU is concerned).

Be Advised: There is approximately a 1000ns time delay between when the negotiation start interrupt is generated and when the exr register is updated.

6 Write data to be transmitted to RvDatFifoWriteReg. If the packet being transmitted does not end on a word boundary, it must be written to the Reverse Data FIFO Write Last register. See the description of this register in the *NS9775 Hardware Reference* for instructions on how to do this. In addition, the RvFifoRdy and RvFifoFull interrupts in the FIFO Status register can be used to verify that there is room in the FIFO.

DMA access

Perform these steps before the steps for compatibility mode or byte/nibble mode:

1 Write to the Master Reset register in the BBus Utility module:

a Bit [8]: Clear BBus utility reset.

2 Write to the Interrupt Enable register in the BBus Bridge module:

a Bit [31]: Enable BBus bridge interrupt.

b Bit [12]: Enable BBus utility interrupt.

c. Bit [11]: Enable 1284 interrupt.

3. Write to GPIO Configuration Register #7 in the BBus Utility module:
   a. Bits [3:0]: Allocate 1284 control signal.
   b. Bits [7:4]: Set PLH to be an output at this time.
4. Write to the Port Control register:
   a. Bits [7:0]: Drive pins to a 1 during initialization.
5. Write to GPIO Configuration #5 in the BBus Utility module:
   a. Bits [31:0]: Allocate 1284 control signals.
6. Write to GPIO Configuration Register #1 in the BBus Utility module:
   a. Bits [27:12]: Allocate 1284 control signals.
7. Write to GPIO Configuration Register #6 in the BBus Utility module:
   a. Bits [31:16]: Allocate 1284 control signals.
8. Write to the Endian Configuration register in the BBus Utility module:
   a. Bit [6]: Configure AHB to be big endian.
9. Write to the Master Reset register:
   a. Bit [6]: Clear 1284 reset.
   b. Bit [0]: Clear BBus DMA reset.
10. Write DMA registers to the Interrupt Enable register in the BBus Bridge module:
    a. Bit [0]: Enable BBus DMA interrupt.
11. Write to BBus DMA Channel 11 Buffer Descriptor register and Channel 12 Buffer Descriptor register in the BBus DMA Controller module:
    a. Bits [31:0]: Write the beginning location of the DMA descriptor ring here.
12. Write to the BBus DMA Channel 11 Control register and BBus DMA Channel 12 Control register in the BBus DMA Controller module:
    a. Bits [27:26], *Channel 11 only*: Set for fly-by write.
    b. Bits [25:24], *both Channel 11 and 12*: Set for four operations.
    c. Bits [27:26], *Channel 12 only*: Set for fly-by read.

13 Write to the BBus DMA Channel 11 Status/Interrupt Enable register and BBus DMA Channel 12 Status/Interrupt Enable in the BBus DMA Controller module:

- **a** Bit [24]: Enable normal completion interrupt.
- **b** Bit [23]: Enable error completion interrupt.
- **c** Bit [22]: Disable buffer not ready interrupt.
- **d** Bit [21]: Enable channel abort interrupt.
- **e** Bit [20]: Enable premature completion interrupt.

14 Write to the BBus Utility DMA Interrupt Enable register in the BBus Utility module:

- **a** Bit [12], *Channel 12 only*: Enable BBus channel 12.
- **b** Bit [11], *Channel 11 only*: Enable Bbus channel 11.

15 Write to the BBus DMA Channel 11 Control register in the BBus DMA Controller module:

- **a** Bits [31]: Set channel enable.

Compatibility mode, DMA support

The compatibility mode, DMA support programming sequence receives data in compatibility DMA mode and allows negotiation into nibble, byte, and ECP modes.

1 Write to the General Configuration register:

- **a** Bits [3, 1:0]: DMA mode enabled.
- **b** Bits [5:4]: Reverse data FIFO threshold is 29–32 bytes.
- **c** Bits [9:8] Forward data FIFO threshold is 4 bytes.
- **d** Bits [11:10]: Forward command FIFO threshold is 4 bytes.
- **e** Bit [13] PLH signal deasserted.
- **f** Bit [14] All forward data stored in “data” FIFO

2 Write to Interrupt and Status Control register:
   a Bit [17]: Enable Vcm1289Interrupt1.
   b Bit [19]: Enable FwDatFifoRdyInterrupt.
   c Bit [21]: Enable FwDatFifoMaxBuffer.
   d Bit [23]: Enable FwDatFifoByteGap.

3 Write to FwDatDmaControl register:
   a Bits [15:0]: Set the gap timer to 2048 BBus clocks. This generates an interrupt telling the CPU that there is data in the forward data FIFO. *Note that this field is used for DMA and direct access modes.*
   b Bits [31:16]: Set the maximum buffer size to 1024 bytes. *This field is for DMA mode only.*

4 Write to grn:
   a Bits [7:0]: Write a value of 1 to the granularity counter. This is necessary to initialize the 1284 core.

5 Repeat Step 4 — Write a value of 1 to the granularity counter.

6 Write to the feb register”
   a Bit [0]: Set to a 1.

7 Write to the fei register:
   a Bit [1]: Enable interrupt when the host initiates a negotiation phase.

8 Write `0x0000_00C0` to the ecr register.
   a Bit [6]: Enable reverse request.
   b Bit [7]: Set to a 1.

9 Write to grn:
   a Bits [7:0]: Write a value of 25 to the granularity counter. This causes the maximum time between slave cycles to be 25 BBus cycles.

10 Repeat Step 9 — Write a value of 25 to the granularity counter.

11 Write to GPIO Configuration register #7:
   a Bits [7:4]: Allocate the 1284 control signal.

12 Write to the fea register:

   a Bit [0]: Printer port enabled.

13 Write to the fem register:

   a Bit [2]: Enable auto-negotiate mode.

   b Bit [4]: Enable auto-transfer mode.

   c Bit [5]: Enable SPP mode.

   d Bit [6]: Enable ECP mode.

14 Write to the General Configuration register.;

   a Bit [13]: PLH signal asserted. The core is ready for traffic.

The NS9775 is now configured to accept forward traffic in compatibility mode through BBus DMA. The NS9775 is also configured to auto-negotiate byte, nibble, and ECP modes.

Byte/Nibble mode, DMA support

Byte and nibble modes perform *reverse transfers*; that is, they send data to the host. The configuration steps shown in "Compatibility mode, DMA support" (on page 66) enable the NS9775 to negotiate to byte/nibble modes. This programming sequence illustrates a negotiation to byte/nibble mode and a reverse transfer:

1 Enable the NS9775 as described in Steps 1–14 of "Compatibility mode, DMA support," beginning on page 66.

2 Wait for a negotiation start interrupt. This is determined by reading the interrupt status registers as described next in Steps 3–5.

3 Read the InterruptStatusandControl register.

   – If bit [1] (peripheral controller interrupt 1) is set, a 1284 peripheral interrupt has occurred.

4 Read the sti register.

   – If bit [1](negotiation start interrupt detect) is set, the host has started a negotiation phase.

5 Read the exr register to determine which mode the host is requesting. Valid values are:

- 0x00 — Nibble mode
- 0x01 — Byte mode
- 0x04 — Device ID, nibble mode
- 0x05 — Device ID, byte mode
- 0x14 — Device ID, ECP
- 0x15 — Device ID, ECP with RLE
- 0x10 — ECP mode
- 0x30 — ECP mode with RLE

Data can now be transmitted using BBus DMA.

Serial Controller

CHAPTER 7

This chapter provides sample driver configurations for the serial controller. Use these samples as guidelines for developing your own drivers.

Keep in mind that this is only one way to configure the serial controller.

Configuring the serial controller in UART mode

This section shows two sample configurations for the serial controller in UART mode.

Configuration example #1

System characteristics

- UART operation
- Odd parity
- 1 stop bit
- 8 data bits per word
- Processor-controlled data transfer (non-DMA)
- Character gap timer set to 10 bit periods
- 230,400 baud rate

Configuration sequence

1. Write `0x0B00_0A02` to Serial Channel B/A/C/D Control Register A.
   a. Enable parity generation and checking by setting the PE bit.
   b. Set the word length to 8 bits by setting the WLS bit.
   c. Enable the RRDY interrupt by setting bit 11 in the RIE field.
   d. Enable the RBC interrupt by setting bit 9 in the RIE field.
   e. Enable the TBC interrupt by setting bit 1 in the TIC field.
2. Write `0x0408_0000` to Serial Channel B/A/C/D Control Register B.
   a. Enable the character gap timer by setting the RCGT bit.
   b. Define MSB-first data streams by setting BITORDR.

3. Write `0xC014_0003` to Serial Channel B/A/C/D Bit Rate register.
   a. Enable the bit rate generator by setting EBIT.
   b. Set the TMODE bit (to 1).
   c. Set the transmit divide rate to 16x by setting TDCR.
   d. Set the receive divide rate to 16x by setting RDCR.
   e. Set the divisor value to 3 by setting the N bit.
4. Write `0x8000_009F` to Serial Channel Receive Character Gap Timer register.
   a. Enable the character gap timer by setting TRUN.
   b. Define the character gap timer value by setting CT.
5. Write `0x8B00_0A02` to Serial Channel B/A/C/D Control Register A.
   a. Enable the serial channel by setting the CE bit.
6. See the discussion about FIFO Management in the *NS9775 Hardware Reference* for information about moving data in and out of the serial controller data FIFOs.

Configuration example #2

System characteristics

- UART operation
- Even parity
- 1 stop bit
- 8 data bits per word
- DMA-controlled data transfer
- Character gap timer set to 4 bit periods
- 921,600 baud rate

Configuration sequence

1. Write `0x1B00_0101` to Serial Channel B/A/C/D Control Register A.
   a. Enable odd parity by setting the EPS bit.
   b. Enable parity generation and checking by setting the PE bit.
   c. Set the word length to 8 bits by setting the WLS bit.
   d. Enable receive path DMA by setting ERXDMA.
   e. Enable transmit path DMA by setting ETXDMA.
2. Write `0x0400_0000` to Serial Channel B/A/C/D Control Register B.
   a. Enable the character gap timer by setting RCGT.
3. Write `0xC014_0000` to Serial Channel B/A/C/D Bit Rate register.
   a. Enable the bit rate generator by setting EBIT.
   b. Set the TMODE bit (to 1).
   c. Set the transmit divide rate to 16x by setting TDCR.
   d. Set the receive divide rate to 16x by setting RDCR.
   e. Set the divisor value to 0 by setting the N bit.
4. Write `0x8000_000F` to Serial Channel B/A/C/D Receive Gap Timer register.
   a. Enable the character gap timer by setting TRUN.
   b. Define the character gap timer value by setting CT.
5. See the BBus DMA Configurations chapter for examples for creating DMA buffer descriptors.
6. Write `0x9B00_0101` to Serial Channel B/A/C/D Control Register A.
   a. Enable the serial channel by setting CE.

Configuring the serial controller in SPI master mode

This section shows a sample configuration sequence for the serial controller in SPI master mode.

System characteristics

- SPI master operation
- Processor-controlled data transfer (non-DMA)
- 3.125 Mbps data rate
- Character gap timer set to 10 bit periods

Configuration sequence

1. Write `0x0000_0A03` to Serial Channel B/A/C/D Control Register A.
   a. Enable the RRDY interrupt by setting bit 11 in the RIE field.
   b. Enable the RBC interrupt by setting bit 9 in the RIE field.
   c. Enable the THALF interrupt by setting bit 2 in the TIC field.
   d. Enable the TBC interrupt by setting bit 1 in the TIC field.
2. Write `0x420` to Serial Channel B/A/C/D Control Register B.
   a. Enable the character gap timer by setting RCGT.
   b. Set the operating mode to SPI master.
3. Write `0xC520_0007` to Serial Channel B/A/C/D Bit Rate register.
   a. Enable the bit rate generator by setting EBIT.
   b. Set the TMODE bit (to 1).
   c. Drive the transmit clock off chip by setting TXEXT.
   d. Define the base frequency as BCLK by setting CLKMUX.
   e. Define the divisor as 7 by setting N.

4. Write `0x8000_000B` to Serial Channel B/A/C/D Receive Character Gap Timer register.
   a. Enable the character gap timer by setting TRUN.
   b. Define the character gap timer value by setting CT.
5. Write `0x8000_0A03` to Serial Channel B/A/C/D Control register A.
   a. Enable the serial channel by setting CE.
6. See the discussion about FIFO Management in the *NS9775 Hardware Reference* for information about moving data in and out of the serial controller data FIFOs.

LCD Configuration

CHAPTER 8

This chapter provides four sample driver configurations for the LCD module. Use these samples as guidelines for developing your own drivers.

Keep in mind that each sample reflects only one way to configure the LCD module.

Configuration for 18-bit TFT LCD panel

This configuration sequence illustrates a system with the NS9775 driving an 18-bit TFT LCD panel.

NS9775 LCD controller characteristics

- 640 x 480 display resolution
- 16 bits-per-pixel display memory
- Common intensity bit for R, G, and B (that is, least significant bit of 6-bit color) supports 64K with 18-bit interface
- Dual display buffers created in system memory at base addresses `0x1000_0000` and `0x1010_0000`
- Big endian byte order
- Generates an interrupt when the contents of the LCDUPBASE register can be updated
- Only requests DMA when at least 8 empty locations in the internal DMA FIFOs
- Internal palette RAM bypassed
- 100 MHz AHB clock
- LCD panel clock (`CLCP`) derived from AHB clock

LCD panel characteristics

- 18-bit color TFT
- 640 x 480 resolution
- 60 Hz refresh rate
- 18 bits-per-pixel (6:6:6 RGB)
- 25 MHz pixel clock rate
- 90 panel clock, active low, horizontal sync pulse width
- 20 panel clock horizontal front porch

- 45 panel clock horizontal back porch
- 4 line, active low, vertical sync pulse width
- 7 line vertical front porch
- 34 line vertical back porch
- 640 pixel clock enable signal, active high
- No line end signal required
- Data and control sampled on falling edge of panel clock (CLCP)

Configuration sequence

What to do first

- **Take the LCD controller out of reset.** The LCDC bit in the Reset and Sleep register (in the System Control module) provides a soft reset to the LCD controller. This bit defaults to a 1, which is the non-reset or enabled state, after powerup or chip reset.
- **Select the LCD panel clock.** The source for the LCD panel clock (CLCP) is selected using the LPCS field in the Clock Configuration register (in the System Control module). In this example, the 100 MHz AHB clock is divided by 4 in the LCD controller to yield a 25 MHz CLCP; the LPCS is set to 000.

  The LCC bit in the Clock Configuration register enables the clocks to the LCD controller and must be set to a 1 (which is the default value).

Configure the registers

The configuration sequence shows the value to which each register in the LCD controller must be configured to meet the internal and LCD panel-specific requirements provided in "NS9775 LCD controller characteristics," beginning on page 78, and "LCD panel characteristics" on page 78.

Note: Unless otherwise noted, you can perform these steps in any order.

See the discussion of LCD registers in the LCD chapter in the *NS9775 Hardware Reference*, as necessary. See also the LCD timing parameter table, in the Timing chapter in the *NS9775 Hardware Reference*, for any LCD timing specifications not addressed in this example.

1 Write 0x2C13_599C to the LCD Timing 0 register, to configure these fields:

HBP (horizontal back porch)	= 0x2C (45 = HBP + 1 pixel clocks)
HFP (horizontal front porch)	= 0x13 (20 = HFP + 1 pixel clocks)
HSW (horizontal sync width)	= 0x59 (90 = HSW + 1 pixel clocks)
PPL (pixels per line)	= 0x27 (640 = 16*(PPL + 1) pixels)

2 Write 0x2207_0DDF to the LCD Timing 1 register, to configure these fields:

VBP (vertical back porch)	= 0x22 (lines)
VFP (vertical front porch)	= 0x07 (lines)
VSW (vertical sync width)	= 0x03 (4 = VSW + 1 lines)
LPP (lines per panel)	= 0x1DF (480 = LPP + 1 lines)

3 Write 0x027F_1802 to the LCD Timing 2 register, to configure these fields:

BCD (bypass pixel clock divider)	= 0x0 (do not bypass clock divider)
CPL (clocks per line)	= 0x27F (640 = CPL + 1 clocks)
IOE (invert output/data enable)	= 0x0 (high true)
IPC (invert panel clock)	= 0x0 (drive data on CLCP rising edge because LCD panel samples data on CLCP falling edge)
IHS (invert horizontal sync pulse)	= 0x1 (low true)
IVS (invert vertical sync pulse)	= 0x1 (low true)
ACB (AC bias bin frequency)	= 0x00 (N/A for TFT)
PCD (panel clock divisor)	= 0x2 (CLCP=CLCDCLK/(PCD+2) to derive 25 MHz panel clock from 100 MHz AHB clock

4 Write 0x0000_0000 to the LCD Timing 3 register, as the LCD panel does not use the line end signal (CLLE).

5 Write 0x1000_0000 to the LCDUPBASE register to initialize the DMA base address to the location of the first display buffer in NS9775 system memory.

Note: LCDLPBASE is not written as it is not used for TFT panels.

6 Write 0x0000_0004 to LCDINTRENABLE to enable the LNBUINTRENB interrupt, which occurs when LCDUPBASE can be updated to the other display buffer at 0x1010_0000.

7 Write 0x0001_0228 to the LCD Control register to configure these fields:

WATERMARK	= 0x1 (request DMA when there are at least 8 empty FIFO locations)
LcdVcomp	= 0x0 (vertical interrupt condition select; N/A in this application)
LcdPwr (LCD power enable)	= 0x0 (power off)
BEPO (big endian pixel ordering)	= 0x0 (little endian)
BEBO (big endian byte ordering)	= 0x1 (big endian)
BGR (RGB format)	= 0x0 (RGB)
LcdDual (single/dual panel)	= 0x0 (always 0 for TFT)
LcdMono8 (STN mono 8-bit interface)	= 0x0 (always 0 for TFT)
LcdTFT (TFT select)	= 0x1 (TFT)
LcdBW (STN mono select)	= 0x0 (always 0 for TFT)
LcdBPP (bits-per-pixel)	= 0x100 (16 bits per pixel)
LcdEn (LCD controller enable)	= 0x0 (disabled)

8 Initialize both display buffers in system memory at 0x1000_0000 and 0x1010_0000. The data format is such that each 32-bit word in a display buffer contains two 16-bit pixels. Pixel0 is in bits [31:16] and Pixel1 is in bits [15:0].

9 *This must be the last step in the configuration sequence*. The LCD controller is enabled in this step and the NS9775 begins driving the TFT LCD panel. *It is the system designer's responsibility to ensure that all power sequencing requirements of the specific LCD panel are satisfied*.

a Set the LcdEn and LcdPwr bits in the LCD Control register to a 1, to enable the LCD controller.

Configuration for 8-bit color STN LCD panel

This configuration sequence illustrates a system with the NS9775 driving an 8-bit color STN LCD panel.

NS9775 LCD controller characteristics

- 320 x 240 display resolution
- 8 bits-per-pixel display memory
- Dual display buffers created in system memory at base addresses `0x1000_0000` and `0x1010_0000`
- Little endian byte order
- Generates an interrupt when the contents of the LCDUPBASE register can be updated
- Only requests DMA when at least 8 empty locations in the internal DMA FIFOs
- Internal palette RAM bypassed
- 100 MHz AHB clock
- LCD panel clock (`CLCP`) derived from AHB clock

LCD panel characteristics

- 8-bit color STN
- 320 x 240 resolution
- 76 Hz refresh rate
- 3 bits-per-pixel (1:1:1 RGB)
- $2^{2/3}$ pixels/clock
- 2.5 MHz panel clock rate
- 4 panel clock, active high, horizontal sync pulse width
- 6 panel clock horizontal front porch
- 6 panel clock horizontal back porch
- 1 line, active high, vertical sync pulse width
- 0 line vertical front porch
- 1 line vertical back porch
- No line end signal required
- Active high display enable control signal driven using CLPOWER output
- Data and control sampled on falling edge of panel clock (CLCP)

Configuration sequence

What to do first

- **Take the LCD controller out of reset.** The LCDC bit in the Reset and Sleep register (in the System Control module) provides a soft reset to the LCD controller. This bit defaults to 1, which is the non-reset or enabled state, after powerup or chip reset.
- **Select the LCD panel clock.** The source for the LCD panel clock (CLCP) is selected using the LPCS field in the Clock Configuration register (in the System Control module). In this example, the 100 MHz AHB clock is divided by 40 to yield a 2.5 MHz CLCP. The LPCS is set to 010 to select the AHB clock divided by 4; the LCD controller then divides the value by 10.

  The LCC bit in the Clock Configuration register enables the clocks to the LCD controller and must be set to 1 (which is the default value).

Configure the registers

The configuration sequence shows the value to which each register in the LCD controller must be configured to meet the internal and LCD panel-specific requirements provided in "NS9775 LCD controller characteristics," beginning on page 82, and "LCD panel characteristics" on page 83.

Note: Unless otherwise noted, you can perform these steps in any order.

See the discussion of LCD registers in the LCD chapter in the *NS9775 Hardware Reference*, as necessary. See also the LCD timing parameter table, in the Timing chapter in the *NS9775 Hardware Reference*, for any LCD timing specifications not addressed in this example.

1 Write `0x0505_034C` to the LCD Timing 0 register, to configure these fields:

HBP (horizontal back porch)	= 0x05 (6 = HBP + 1 panel clocks)
HFP (horizontal front porch)	= 0x05 (6 = HFP + 1 panel clocks)
HSW (horizontal sync width)	= 0x03 (4 = HSW + 1 panel clocks)
PPL (pixels per line)	= 0x13 (320 = 16*(PPL + 1) pixels)

2 Write `0x000_00EF` to the LCD Timing 1 register, to configure these fields:

VBP (vertical back porch)	= 0x0 (1 = VBP + 1 line)
VFP (vertical front porch)	= 0x0 (lines)
VSW (vertical sync width)	= 0x0 (always 1 line for STN)
LPP (lines per panel)	= 0xEF (240 = LPP + 1 lines)

3 Write 0x0077_0008 to the LCD Timing 2 register, to configure these fields:

BCD (bypass pixel clock divider)	= 0x0 (do not bypass clock divider)
CPL (clocks per line)	= 0x77 ($320/2^{2/3}$ = CPL + 1 clocks)
IOE (invert output/data enable)	= 0x0 (N/A for STN)
IPC (invert panel clock)	= 0x0 (drive data on CLCP rising edge because LCD panel samples data on CLCP falling edge)
IHS (invert horizontal sync pulse)	= 0x0 (high true)
IVS (invert vertical sync pulse)	= 0x0 (high true)
ACB (AC bias bin frequency)	= 0x00 (N/A for this STN)
PCD (panel clock divisor)	= 0x8 (CLCP=CLCDCLK/(PCD+2) to derive 2.5 MHz panel clock from 100 MHz AHB clock divided by 4)

4 Write 0x0000_0000 to the LCD Timing 3 register, as the LCD panel does not use the line end signal (CLLE).

5 Write 0x1000_0000 to the LCDUPBASE register to initialize the DMA base address to the location of the first display buffer in NS9775 system memory.

Note: LCDLPBASE is not written as it is not used for single panel STN displays.

6 Write 0x0000_0004 to LCDINTRENABLE to enable the LNBUINTRENB interrupt, which occurs when LCDUPBASE can be updated to the other display buffer at 0x1010_0000.

7 Write 0x0001_0006 to the LCD Control register to configure these fields:

WATERMARK	= 0x1 (request DMA when there are at least 8 empty FIFO locations)
LcdVcomp	= 0x0 (vertical interrupt condition select; N/A in this application)
LcdPwr (LCD power enable)	= 0x0 (power off)
BEPO (big endian pixel ordering)	= 0x0 (little endian)
BEBO (big endian byte ordering)	= 0x0 (little endian)
BGR (RGB format)	= 0x0 (RGB)
LcdDual (single/dual panel)	= 0x0 (single panel STN)
LcdMono8 (STN mono 8-bit interface)	= 0x0 (always 0 for color STN)
LcdTFT (TFT select)	= 0x0 (STN)
LcdBW (STN mono select)	= 0x0 (always 0 for color STN)
LcdBPP (bits-per-pixel)	= 0x011 (8 bits per pixel)
LcdEn (LCD controller enable)	= 0x0 (disabled)

8 Initialize the 256-entry palette RAM using the LCD Palette registers. Color STNs use only bits [4:1] of each color.

9 Initialize both display buffers in memory at 0x1000_0000 and 0x1010_0000. The data format is such that each 32-bit word in a display buffer contains four 8-bit pixels. Pixel0 is in bits [7:0], Pixel1 is in bits [15:8]; Pixel2 is in bits {23:16], and Pixel3 is in bits [31:24].

10 *This must be the last step in the configuration sequence.* The LCD controller is enabled in this step and the NS9775 begins driving the STN LCD panel. *It is the system designer's responsibility to ensure that all power sequencing requirements of the specific LCD panel are satisfied.*

- **a** Set the LcdEn bits in the LCD Control register to 1, to enable the CLLP, CLFP, and CLCP signals to the LCD panel.
- **b** If the LCD panel has a requirement to keep the panel disabled through CLPOWER until the contrast voltage is stable, wait the appropriate amount of time now.
- **c** Set the LcdPwr bit in the LCD Control register to 1, to enable the LCD panel by asserting CLPOWER. Bits CLD [7:0] are activated at this time also.

Configuration for 4-bit monochrome STN LCD panel

This configuration sequence illustrates a system with the NS9775 driving a 4-bit monochrome STN LCD panel.

NS9775 LCD controller characteristics

- 320 x 240 display resolution
- 4 bits-per-pixel display memory
- Dual display buffers created in system memory at base addresses 0x1000_0000 and 0x1010_0000
- Little endian byte order
- Generates an interrupt when the contents of the LCDUPBASE register can be updated
- Only requests DMA when at least 8 empty locations in the internal DMA FIFOs
- Internal palette RAM used
- 100 MHz AHB clock
- LCD panel clock (CLCP) derived from AHB clock

LCD panel characteristics

- 4-bit monochrome STN
- 320 x 240 resolution
- 72 Hz refresh rate
- 1 bit-per-pixel
- 4 pixels/panel clock
- 1.67 MHz panel clock rate
- 4 panel clock, active high, horizontal sync pulse width
- 6 panel clock horizontal front porch
- 6 panel clock horizontal back porch
- 1 line, active high, vertical sync pulse width
- 0 line vertical front porch
- 1 line vertical back porch
- No line end signal required
- Active high display enable control signal driven using CLPOWER output
- Data and control sampled on falling edge of panel clock (CLCP)
- Requires AC bias control signal that toggles every 16 lines to prevent DC charge accumulation

Configuration sequence

What to do first

- **Take the LCD controller out of reset.** The LCDC bit in the Reset and Sleep register (in the System Control module) provides a soft reset to the LCD controller. This bit defaults to 1, which is the non-reset or enabled state, after powerup or chip reset.

- **Select the LCD panel clock.** The source for the LCD panel clock (CLCP) is selected using the LPCS field in the Clock Configuration register (in the System Control module). In this example, the 100 MHz AHB clock is divided by 60 to yield a 1.67 MHz CLCP. The LPCS is set to 010 to select the AHB clock divided by 4; the LCD controller then divides the value by 15.

  The LCC bit in the Clock Configuration register enables the clocks to the LCD controller and must be set to 1 (which is the default value).

Configure the registers

The configuration sequence shows the value to which each register in the LCD controller must be configured to meet the internal and LCD panel-specific requirements provided in "NS9775 LCD controller characteristics" on page 87 and "LCD panel characteristics" on page 88.

Note: Unless otherwise noted, you can perform these steps in any order.

See the discussion of LCD registers in the LCD chapter in the *NS9775 Hardware Reference*, as necessary. See also the LCD timing parameter table, in the Timing chapter in the *NS9775 Hardware Reference*, for any LCD timing specifications not addressed in this example.

1 Write 0x0505_034C to the LCD Timing 0 register, to configure these fields:

HBP (horizontal back porch)	= 0x05 (6 = HBP + 1 panel clocks)
HFP (horizontal front porch)	= 0x05 (6 = HFP + 1 panel clocks)
HSW (horizontal sync width)	= 0x03 (4 = HSW + 1 panel clocks)
PPL (pixels per line)	= 0x13 (320 = 16*(PPL + 1) pixels)

2 Write 0x000_00EF to the LCD Timing 1 register, to configure these fields:

VBP (vertical back porch)	= 0x0 (1 = VBP + 1 lines)
VFP (vertical front porch)	= 0x0 (lines)
VSW (vertical sync width)	= 0x0 (always 1 line for STN)
LPP (lines per panel)	= 0xEF (240 = LPP + 1 lines)

3 Write 0x004F_07CD to the LCD Timing 2 register, to configure these fields:

BCD (bypass pixel clock divider)	= 0x0 (do not bypass clock divider)
CPL (clocks per line)	= 0x4F (320/4 = CPL + 1 clocks)
IOE (invert output/data enable)	= 0x0 (N/A for STN)
IPC (invert panel clock)	= 0x0 (drive data on CLCP rising edge because LCD panel samples data on CLCP falling edge)
IHS (invert horizontal sync pulse)	= 0x0 (high true)
IVS (invert vertical sync pulse)	= 0x0 (high true)
ACB (AC bias bin frequency)	= 0x1F (32 = ACB + 1 lines)
PCD (panel clock divisor)	= 0x0D (CLCP=CLCDCLK/(PCD+2) to derive 1.67 MHz panel clock from 100 MHz AHB clock divided by 4)

4 Write 0x0000_0000 to the LCD Timing 3 register, as the LCD panel does not use the line end signal (CLLE).

5 Write 0x1000_0000 to the LCDUPBASE register to initialize the DMA base address to the location of the first display buffer in NS9775 system memory.

Note: LCDLPBASE is not written as it is not used for single panel STN displays.

6 Write 0x0000_0004 to LCDINTRENABLE to enable the LNBUINTRENB interrupt, which occurs when LCDUPBASE can be updated to the other display buffer at 0x1010_0000.

7 Write `0x0001_0014` to the LCD Control register to configure these fields:

WATERMARK	= 0x1 (request DMA when there are at least 8 empty FIFO locations)
LcdVcomp	= 0x0 (vertical interrupt condition select; N/A in this application)
LcdPwr (LCD power enable)	= 0x0 (power off)
BEPO (big endian pixel ordering)	= 0x0 (little endian)
BEBO (big endian byte ordering)	= 0x0 (little endian)
BGR (RGB format)	= 0x0 (RGB)
LcdDual (single/dual panel)	= 0x0 (single panel STN)
LcdMono8 (STN mono 8-bit interface)	= 0x0 (4-bit interface)
LcdTFT (TFT select)	= 0x0 (STN)
LcdBW (STN mono select)	= 0x1 (always 1 for mono STN)
LcdBPP (bits-per-pixel)	= 0x010 (4 bits per pixel)
LcdEn (LCD controller enable)	= 0x0 (disabled)

8 Initialize the 256-entry palette RAM using the LCD Palette registers. Mono STNs use bits [4:1] of the red palette only. (See the discussion of the LCD Palette register in the LCD chapter of the *NS9775 Hardware Reference*.)

9 Initialize both display buffers in memory at 0x1000_0000 and 0x1010_0000. The data format is such that each 32-bit word in a display buffer contains eight 4-bit pixels; the pixels are aligned within the 32-bit word as shown:

Pixel number	Data bits
0	[3:0]
1	[7:4]
2	[11:8]
3	[15:12]
4	[19:16]
5	[23:20]
6	[27:24]
7	[31:28]

10 *This must be the last step in the configuration sequence*. The LCD controller is enabled in this step and the NS9775 begins driving the STN LCD panel. *It is the system designer's responsibility to ensure that all power sequencing requirements of the specific LCD panel are satisfied.*

- Set the LcdEn bits in the LCD Control register to 1, to enable the CLAC, CLLP, CLFP, and CLCP signals to the LCD panel.
- If the LCD panel has a requirement to keep the panel disabled through CLPOWER until the contrast voltage is stable, wait the appropriate amount of time now.
- Set the LcdPwr bit in the LCD Control register to 1, to enable the LCD panel by asserting CLPOWER. Bits CLD[3:0] are activated at this time also.

USB Configuration

CHAPTER 9

This chapter provides sample driver configurations for the USB module. Use these samples as guidelines for developing your own drivers.

Keep in mind that this is only one way to configure the USB module.

Configuration #1

Characteristics

- USB host mode
- Full speed operation

Configuration sequence

1. Write 0x0000_0000 to the Global Control and Status register.
   a. Put the USB module in host mode by clearing HSTDV.
2. Wait for HRST to be cleared in the GLobal Control and Status register.
3. Write 0x8000_0002 to the Global Interrupt Enable register.
   a. Enable USB global interrupts by setting GBL_EN.
   b. Enable USB host interrupts by setting OHCI_IRQ.
4. See the related industry standards to configure the OHCI (open host controller interface).

Configuration #2

Characteristics

- USB device mode
- Full speed operation
- One bulk-in endpoint
- One bulk-out endpoint
- DMA-controlled data transfer
- USB device dynamic programming disabled

Configuration sequence

1. See the BBus DMA Configurations chapter for examples for creating DMA buffer descriptors
2. Write `0x3800_0000` to the Device Control and Status register.
   a. Define the device as *self-powered* by setting SELF_PWR.
   b. Enable set descriptor support by setting SET_DESC.
   c. Enable start of frame support by setting SOF.
3. Write `0x8803_D000` to the Global Interrupt Enable register.
   a. Enable USB global interrupts by setting GBL_EN
   b. Enable USB DMA global interrupts by setting GBL_DMA.
   c. Enable USB DMA channel 4 interrupts by setting DMA4.
   d. Enable USB DMA channel 3 interrupts by setting DMA3.
   e. Enable USB DMA channel 2 interrupts by setting DMA2.
   f. Enable USB DMA channel 1 interrupts by setting DMA1.
   g. Enable USB FIFO interrupts by setting FIFO.
4. Write `0x0000_0000` to the Device IP Programming Control/Status register.
   a. Disable USB device dynamic programming support by clearing CSRPRG to 0.

5 Write 0x0000_0100 to the Device Descriptor/Setup Command register.
   a Define the setup command pointer for legacy reasons.

6 Write 0x0200_0080 to the Physical Endpoint Descriptor #1 register.
   a Define the endpoint as 0x0.
   b Define the endpoint type as *control* (direction is “don’t care”).
   c Define the configuration as 0x1.
   d Define the alternate as 0x0.
   e Define the interface as 0x0.
   f Define the max packet size as 64.

7 Write 0x0200_00C1 to the Physical Endpoint Descriptor #2 register.
   a Define the endpoint number as 0x1.
   b Define the endpoint direction as *out*.
   c Define the endpoint type as *bulk*.
   d Define the configuration as 0x1.
   e Define the alternate as 0x0.
   f Define the interface as 0x0.
   g Define the max packet size as 64 bytes.

8 Write 0x0200_00D2 to the Physical Endpoint Descriptor #3 register.
   a Define the endpoint number as 0x2.
   b Define the endpoint direction as *in*.
   c Define the endpoint type as *bulk*.
   d Define the configuration as 0x1.
   e Define the alternate as 0x0.
   f Define the interface as 0x0.
   g Define the max packet size as 64 bytes.

9 Write 0x0000_6060 to the FIFO Interrupt Enable #0 register.
   a Enable the endpoint #0 (CTRL-In) NACK interrupt by setting NACK2.
   b Enable the endpoint #0 (CTRL-In) ERROR interrupt by setting ERROR2.

c Enable the endpoint #0 (CTRL-Out) NACK interrupt by setting NACK1.

d Enable the endpoint #0 (CTRL-Out) ERROR interrupt by setting ERROR1.

10 Write `0x0000_6060` to the FIFO Interrupt Enable #1 register.

a Enable the endpoint #2 NACK interrupt by setting NACK4.

b Enable the endpoint #2 ERROR interrupt by setting ERROR4.

c Enable the endpoint #1 NACK interrupt by setting NACK3.

d Enable the endpoint #1 ERROR interrupt by setting ERROR3.

11 Write `0x0400_0000` to the FIFO Packet Control #1 register.

a Define the endpoint #0 (CTRL-Out) max packet size as 64 bytes.

12 Write `0x0400_0000` to the FIFO Packet Control #3 register.

a Define the endpoint #1 max packet size as 64 bytes.

13 Write `0x0400_0000` to the FIFO Packet Control #4 register.

a Define the endpoint #2 max packet size as 64 bytes.

14 Write `0x0004_0000` to the FIFO Status and Control #1 register.

a Define the endpoint #0 (CTRL-In) FIFO type as *control*.

b Take the endpoint #0 (CTRL-In) FIFO out of reset by clearing CLR.

c Define the endpoint #0 (CTRL-In) FIFO direction as *in*.

15 Write `0x0000_0000` to the FIFO Status and Control #2 register.

a Define the endpoint #0 (CTRL-Out) FIFO type as *control*.

b Take the endpoint #0 (CTRL-Out) FIFO out of reset by clearing CLR.

c Define the endpoint #0 (CTRL-Out) FIFO direction as *out*.

16 Write `0x0020_0000` to the FIFO Status and Control #3 register.

a Define the endpoint #1 FIFO type as *bulk*.

b Take the endpoint #1 FIFO out of reset by clearing CLR.

c Define the endpoint #1 FIFO direction as *out*.

17 Write `0x0024_0000` to the FIFO Status and Control #4 register.

a Define the endpoint #2 FIFO type as *bulk*.

b Take the endpoint #2 FIFO out of reset by clearing CLR.

c Define the endpoint #2 FIFO direction as *in*.

18 Connect USB device to USB bus using a pullup resistor to D+ provided by the system.

19 Process FIFO endpoint and DMA interrupts as data moves through the system.

Configuration #3

Characteristics

- USB device mode
- Full speed operation
- One bulk-in endpoint
- One bulk-out endpoint
- DMA-controlled data transfer
- USB device dynamic programming enabled

Configuration sequence

1. See the BBus DMA Configurations chapter for examples for creating DMA buffer descriptors
2. Write `0x3800_0000` to the Device Control and Status register.
   a. Define the device as *self-powered* by setting SELF_PWR.
   b. Enable set descriptor support by setting SET_DESC.
   c. Enable start of frame support by setting SOF.
3. Write `0x8803_D180` to the Global Interrupt Enable register.
   a. Enable USB global interrupts by setting GBL_EN
   b. Enable USB DMA global interrupts by setting GBL_DMA.
   c. Enable USB DMA channel 4 interrupts by setting DMA4.
   d. Enable USB DMA channel 3 interrupts by setting DMA3.
   e. Enable USB DMA channel 2 interrupts by setting DMA2.
   f. Enable USB DMA channel 1 interrupts by setting DMA1.
   g. Enable USB FIFO interrupts by setting FIFO.
   h. Enable SET INTERFACE packet interrupts by setting SETINTF.
   i. Enable SET CONFIGURATION packet interrupts by setting SETCFG.

4. Write 0x0000_0001 to the Device IP Programming Control/Status register.
   a. Enable USB device dynamic programming support by setting CSRPRG to 1.
5. Write 0x0000_0100 to the Device Descriptor/Setup Command register.
   a. Define the setup command pointer for legacy reasons.
6. Write 0x0200_0080 to the Physical Endpoint Descriptor #1 register.
   a. Define the endpoint as 0x0.
   b. Define the endpoint type as *control* (direction is "don't care").
   c. Define the configuration as 0x1.
   d. Define the alternate as 0x0.
   e. Define the interface as 0x0.
   f. Define the max packet size as 64.
7. Write 0x0000_6060 to the FIFO Interrupt Enable #0 register.
   a. Enable the endpoint #0 (CTRL-In) NACK interrupt by setting NACK2.
   b. Enable the endpoint #0 (CTRL-In) ERROR interrupt by setting ERROR2.
   c. Enable the endpoint #0 (CTRL-Out) NACK interrupt by setting NACK1.
   d. Enable the endpoint #0 (CTRL-Out) ERROR interrupt by setting ERROR1.
8. Write 0x0400_0000 to the FIFO Packet Control #1 register.
   a. Define the endpoint #0 (CTRL-Out) max packet size as 64 bytes.
9. Write 0x0000_0000 to the FIFO Status and Control #1 register.
   a. Define the endpoint #0 (CTRL-Out) FIFO type as *control*.
   b. Take the endpoint #0 (CTRL-Out) FIFO out of reset by clearing CLR.
   c. Define the endpoint #0 (CTRL-Out) FIFO direction as *out*.
10. Write 0x0004_0000 to the FIFO Status and Control #2 register.
    a. Define the endpoint #0 (CTRL-In) FIFO type as *control*.
    b. Take the endpoint #0 (CTRL-In) FIFO out of reset by clearing CLR.
    c. Define the endpoint #0 (CTRL-In) FIFO direction as *in*.

11 Connect USB device to USB bus using a pullup resistor to D+ provided by the system.

12 Process USB enumeration requests until SET CONFIGURATION or SET INTERFACE interrupt is received to kick off dynamic programming of the USB device.

13 Read CFG, INTF, and ALT values from the Device Control/Status register.

14 Wait for SETCSR to be cleared in the Device IP Programming register.

15 Write `0x0200_00C1` to the Physical Endpoint Descriptor #2 register.

- **a** Define the endpoint number as `0x1`.
- **b** Define the endpoint direction as *out*.
- **c** Define the endpoint type as *bulk*.
- **d** Define the configuration as `0x1`.
- **e** Define the alternate as `0x0`.
- **f** Define the interface as `0x0`.
- **g** Define the max packet size as 64 bytes.

16 Write `0x0200_00D2` to the Physical Endpoint Descriptor #3 register.

- **a** Define the endpoint number as `0x2`.
- **b** Define the endpoint direction as *in*.
- **c** Define the endpoint type as *bulk*.
- **d** Define the configuration as `0x1`.
- **e** Define the alternate as `0x0`.
- **f** Define the interface as `0x0`.
- **g** Define the max packet size as 64 bytes.

17 Write `0x0000_6060` to the FIFO Interrupt Enable #1 register.

- **a** Enable the endpoint #2 NACK interrupt by setting NACK4.
- **b** Enable the endpoint #2 ERROR interrupt by setting ERROR4.
- **c** Enable the endpoint #1 NACK interrupt by setting NACK3.
- **d** Enable the endpoint #1 ERROR interrupt by setting ERROR3.

18 Write `0x0400_0000` to the FIFO Packet Control #3 register.

a Define the endpoint #1 max packet size as 64 bytes.

19 Write `0x0400_0000` to the FIFO Packet Control #4 register.

a Define the endpoint #2 max packet size as 64 bytes.

20 Write `0x0020_0000` to the FIFO Status and Control #3 register.

a Define the endpoint #1 FIFO type as *bulk*.

b Take the endpoint #1 FIFO out of reset by clearing CLR.

c Define the endpoint #1 FIFO direction as *out*.

21 Write `0x0024_0000` to the FIFO Status and Control #4 register.

a Define the endpoint #2 FIFO type as *bulk*.

b Take the endpoint #2 FIFO out of reset by clearing CLR.

c Define the endpoint #2 FIFO direction as *in*.

22 Write `0x0000_0003` to the Device IP Programming Control/Status register.

a Set DONECSR to indicate that the USB device programming is finished.

23 Wait until SETCSR is cleared in the Device IP Programming Control/Status register.

24 Process FIFO endpoint and DMA interrupts as data moves through the system.

PN:(1P) 90000575 A