



# Intel<sup>®</sup> IXP2800 and IXP2850 Network Processors

Interfacing Devices to the Slowport Application Note

---

*August 2004*



INFORMATION IN THIS DOCUMENT IS PROVIDED IN CONNECTION WITH INTEL® PRODUCTS. EXCEPT AS PROVIDED IN INTEL'S TERMS AND CONDITIONS OF SALE FOR SUCH PRODUCTS, INTEL ASSUMES NO LIABILITY WHATSOEVER, AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY RELATING TO SALE AND/OR USE OF INTEL PRODUCTS, INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT, OR OTHER INTELLECTUAL PROPERTY RIGHT.

Intel Corporation may have patents or pending patent applications, trademarks, copyrights, or other intellectual property rights that relate to the presented subject matter. The furnishing of documents and other materials and information does not provide any license, express or implied, by estoppel or otherwise, to any such patents, trademarks, copyrights, or other intellectual property rights.

Intel products are not intended for use in medical, life saving, life sustaining, critical control or safety systems, or in nuclear facility applications.

Intel may make changes to specifications and product descriptions at any time, without notice.

Designers must not rely on the absence or characteristics of any features or instructions marked "reserved" or "undefined." Intel reserves these for future definition and shall have no responsibility whatsoever for conflicts or incompatibilities arising from future changes to them.

The IXP2800 and IXP2850 Network Processors may contain design defects or errors known as errata which may cause the product to deviate from published specifications. Current characterized errata are available on request.

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an ordering number and are referenced in this document, or other Intel literature may be obtained by calling 1-800-548-4725 or by visiting Intel's website at <http://www.intel.com>.

Intel and XScale are registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

\*Other names and brands may be claimed as the property of others.

Copyright © 2004, Intel Corporation

# Contents

---

<b>1.0 Preface</b> .....	5
<b>2.0 Flash PROM Interface Logic</b> .....	5
2.1 Address Latch Logic .....	5
<b>3.0 Microprocessor Interface Logic</b> .....	8
3.1 Address Latch Logic .....	8
3.2 Data Multiplexing and Demultiplexing.....	9
3.2.1 Write Transactions .....	9
3.2.2 Read Transactions.....	12
<b>4.0 Summary</b> .....	14

## Figures

1 SlowPort Application Topology.....	6
2 Mode 0 Single Write Transfer for a Fixed-Timed Device.....	7
3 Mode 0 Single Write Transfer for a Self-timing Device.....	7
4 An Interface Topology with Intel / AMCC* SONET/SDH Device .....	10
5 Mode 3 32-Bit Write Transfer.....	11
6 Mode 32-Bit Read Transfer .....	12

## Revision History

Date	Revision	Description
December 2002	001	First release
March 2004	002	Changes to Section 3.1 (Address Latch Logic), Section 3.2.1 (Write Transactions), and Section 3.2.2 (Read Transactions)
August 2004	003	Preparation for web posting.

## 1.0 Preface

The IXP2800 and IXP2850 Network Processors implement a general purpose low-pin-count bus referred to as the slowport, which is used to interface to the PROM FLASH memory as well as 8-, 16-, and 32-bit microprocessor devices. The address and data buses are multiplexed to reduce pin count and therefore external logic is required to latch the address and data. This document describes how to implement the external logic required to interface to downstream devices.

## 2.0 Flash PROM Interface Logic

The slowport supports two types of interfaces: the flash memory interface and the microprocessor interface. There are two chip-select signals that differentiate between the two, SP\_CS[1:0]. The total address window for the slowport is 64 Mbytes, which is divided in two 32-Mbyte regions, one for each interface. Accesses to the lower 32-Mbyte region of address space is decoded with SP\_CS[0], while accesses to the upper 32-Mbyte region of microprocessor space is decoded with SP\_CS[1]. Using these two signals, the glue logic can easily distinguish between accesses to each interface.

Refer to the *Intel<sup>®</sup> IXP2800 Network Processor Hardware Reference Manual* (HRM) for details regarding the slowport unit.

### 2.1 Address Latch Logic

The flash memory interface only supports 8-bit devices; therefore, no data packing/unpacking is required. However, because the address bus is only eight bits wide, the 24-bit address must be latched by the external logic. The address is shifted out of by the IXP2800 Network Processor, eight bits at a time in three consecutive clock cycles to form the upper 24 bits of the address, while the lower two bits (A[1:0]) are provided on dedicated pins.

The external logic monitors the SP\_ALE\_L signal and anytime it is asserted, it latches the address presented on the SP\_AD[7:0] bus on the rising edge of SP\_CLK, in three consecutive cycles, with the least significant byte (LSB) of the address being delivered first followed by the most significant byte (MSB) being presented last.

**Note:** Timing diagrams for all supported modes are provided in the section, “SlowPort Unit” in Chapter 3 of the *Intel<sup>®</sup> IXP2800 Network Processor Hardware Reference Manual*. We strongly recommend that you consult the HRM and review *all* of the timing diagrams in that section.

The Verilog\* code in [Example 1](#) depicts an example implementation of the logic:

### Example 1. PROM Address Latch Logic

```
// implementation of address packing logic

always @(posedge sp_clk) begin

    if (~rst_l) begin
        latched_add    <= 24'h000000;
    end

    else begin
        sp_ale_l_d    <= sp_ale_l;
        if (~sp_ale_l) begin
            latched_add[7:0]    <= sp_ad_in;
            latched_add[15:8]  <= latched_add[7:0];
            latched_add[23:16] <= latched_add[15:8];
        end
    end

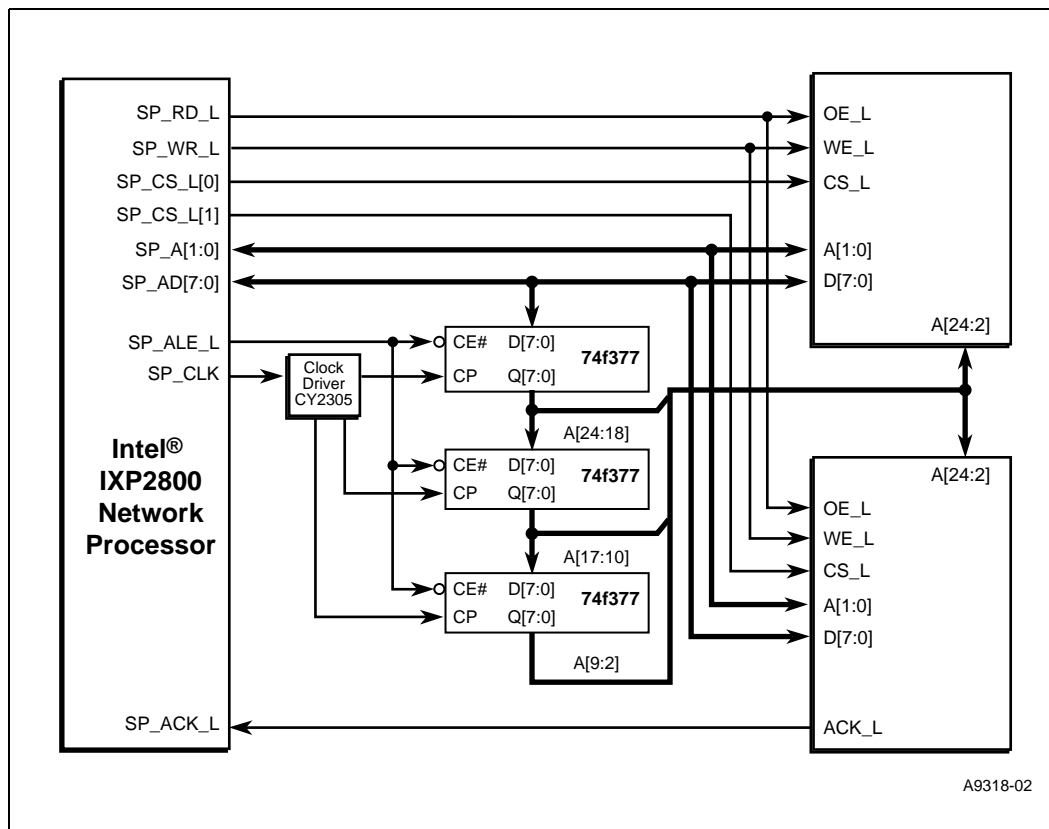
end // else: !if(~rst_l)

end // always @ (posedge sp_clk)
```

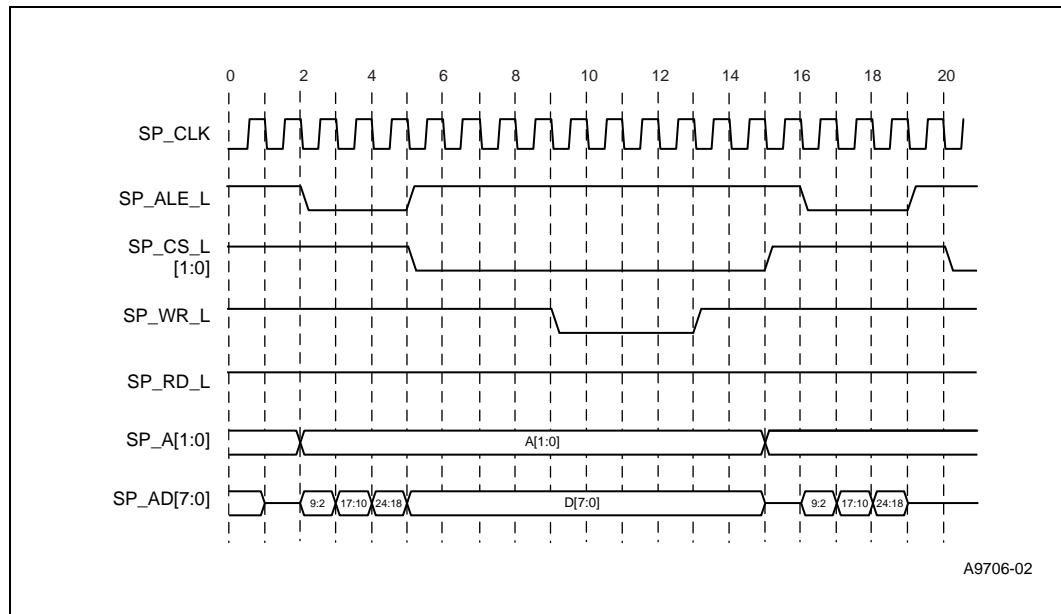
This logic is equivalent to the F377 devices shown in Figure 1.

No additional logic is required to interface to the flash. Figure 2 and Figure 3 depict the timing for a single write and read transaction to the flash memory interface.

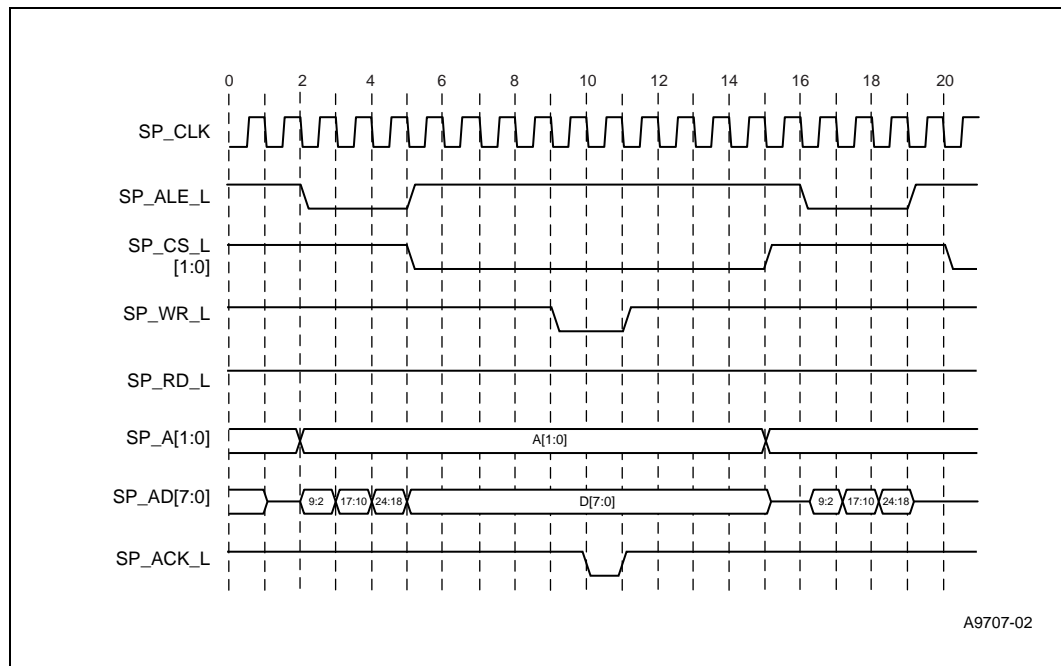
Figure 1. SlowPort Application Topology



**Figure 2. Mode 0 Single Write Transfer for a Fixed-Timed Device**



**Figure 3. Mode 0 Single Write Transfer for a Self-timing Device**



## 3.0 Microprocessor Interface Logic

The slowport also has a microprocessor interface that can be configured to support 8-, 16-, and 32-bit devices. Please refer to the *Intel® IXP2400 and IXP2800 Network Processor Programmer's Reference Manual* for detailed information for each mode.

As with the flash interface, external logic needs to be implemented to latch the address. Additionally, because this interface can also support 8-, 16-, and 32-bit devices, external logic must be implemented to perform the packing and unpacking of the read and write data when configured for 16- and 32-bit mode. To accomplish this, three extra signals, SP\_CP, SP\_OE\_L, and SP\_DIR are provided to control the packing/unpacking of data and to steer the direction of the bus.

### 3.1 Address Latch Logic

The logic required to latch the address is exactly the same as the logic described in [Figure 1](#) for the flash interface. However, there are a few items that vary from the flash interface that need to be understood. First, while the size of the address for the flash interface is fixed, it is programmable in this interface. The slowport address size/data width control register, SP\_ADC, configures the size of the address and data bus. Given that the address can be 8-, 16-, 24-, or 25-bits wide, the number of clock cycles required to latch the address could be as many as four for the 25-bit case, and as few as one, for the 8-bit case. The mechanism for latching the address is the same – on the rising edge of SP\_CLK, if SP\_ALE\_L is asserted. The number of clock cycles in which SP\_ALE\_L is asserted varies, depending on the programmed address size. Given this, the address logic implemented above for the flash interface could be modified as depicted in [Example 2](#).

#### Example 2. Microprocessor Address Latch Logic

```
// implementation of 25-bit address packing control logic

always @(posedge sp_clk) begin
    if (~rst_l) begin
        ale_cnt    <= 2'b00;
    end

    else begin
        if (~sp_ale_l)begin
            ale_cnt    <= ale_cnt + 1;
        end

        else if (sp_ale_l) begin
            ale_cnt    <= 2'b00;
        end

    end // else: !if(~rst_l)
end // always @ (posedge sp_clk)

always @(posedge sp_clk) begin
    if (~rst_l) begin
        latched_add    <= 25'h00000000;
    end

    else begin
        if (~sp_ale_l) begin
            case (ale_cnt) //synopsys full_case parallel_case
                2'b00:    latched_add[7:0]    <= sp_ad_in;
                2'b01:    latched_add[15:8]   <= sp_ad_in;
                2'b10:    latched_add[23:16]  <= sp_ad_in;
                2'b11:    latched_add[24]    <= sp_ad_in[0]; //for 25-bit address space
            endcase
        end
    end
end
```

```

        endcase // case(ale_cnt)
      end

      end // else: !if(~rst_l)

end // always @ (posedge sp_clk)

```

In this example, we use a two-bit count, `ale_cnt`, to steer the data into the appropriate byte-lane. Another method could be to define the `latched_add` register to be 32 bits, dropping any unused upper bits; then `ale_cnt` would not be needed. A coded example of this logic is shown below:

```

// implementation of 32-bit address packing logic

always @(posedge sp_clk) begin
  if (~rst_l) begin
    latched_add    <= 32'h00000000;
  end

  else begin
    if (~sp_ale_l) begin
      latched_add[7:0]      <= sp_ad_in;
      latched_add[15:8]    <= latched_add[7:0];
      latched_add[23:16]   <= latched_add[15:8];
      latched_add[31:24]   <= latched_add[23:16];
    end

    end // else: !if(~rst_l)

end // always @ (posedge sp_clk)

```

## 3.2 Data Multiplexing and Demultiplexing

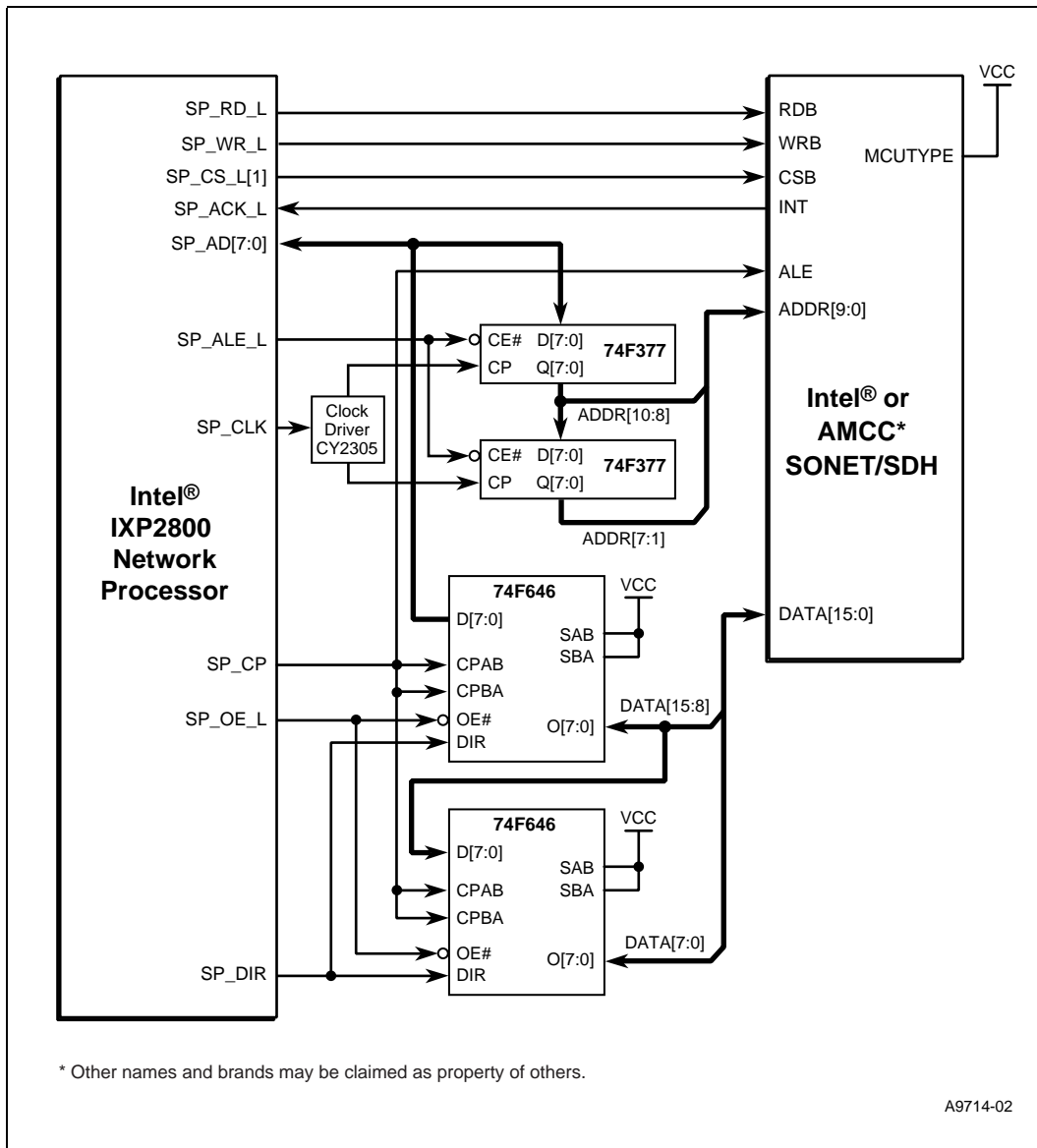
Another difference from the flash interface is that 8-, 16-, and 32-bit devices are supported on the microprocessor interface while only 8-bit devices are supported on the flash interface. This requires that the `SP_AD` data bus be multiplexed/demultiplexed to support read and write transactions. Additionally, there are four different operating modes that are implemented to support a variety of microprocessor interfaces, with different address and data widths. For more information on the operating modes, refer to the *Intel® IXP2800 Network Processor Hardware Reference Manual*.

### 3.2.1 Write Transactions

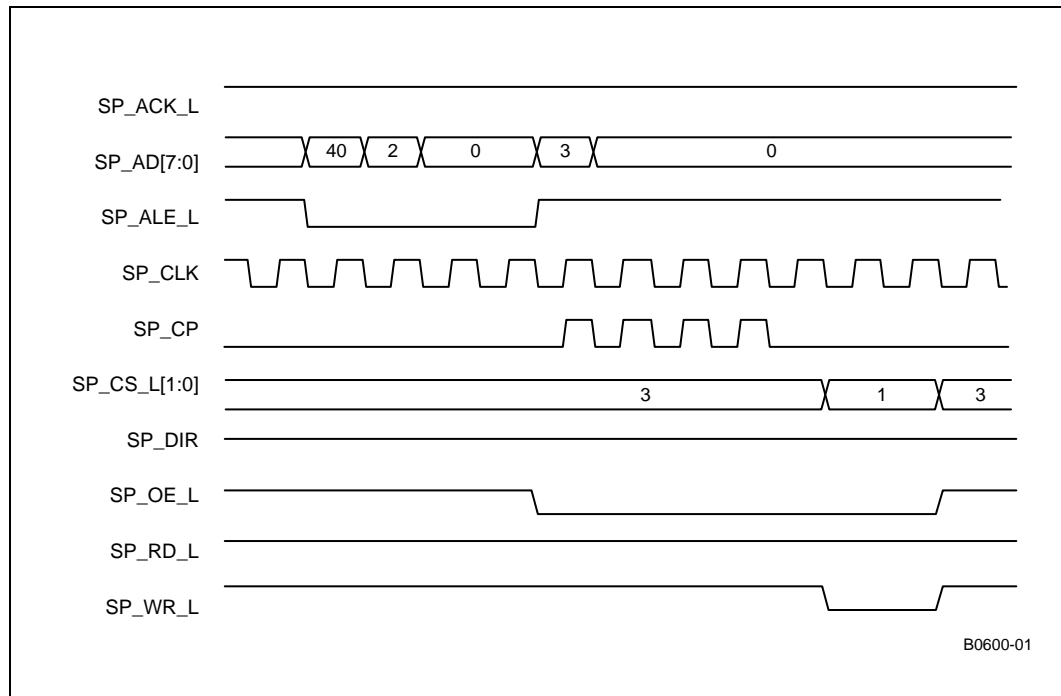
During write transactions to 16- and 32-bit devices, the 8-bit data coming from the IXP2800 `SP_AD` bus must be packed into a 16-bit (WORD) or a 32-bit (DWORD). Extra signals, `SP_CP`, `SP_OE_L`, and `SP_DIR` have been provided to control the pack/unpack operations and to control which device drives the bus during read and write transactions. Note that the actual protocol of these control signals varies slightly, depending on the interface configuration mode (1, 2, 3, or 4). For the remainder of this discussion, we will concentrate on the Mode 3 protocol; this is the mode that would be used to interface to the Intel® IXF1010 10-port 100/1000Mbps Ethernet MAC.

Figure 4 shows an example of discrete components and their associated signal connections that could be used to implement the glue logic. Figure 5 shows the timing of the slowport interface signals presented to the glue logic during a write transaction with the slowport configured for Mode 3, i.e., `SP_PCR = 0x3` and the address and data size set to 32-bit, i.e., `SP_ADC = 0x33`.

Figure 4. An Interface Topology with Intel / AMCC\* SONET/SDH Device



**Figure 5. Mode 3 32-Bit Write Transfer**



To pack the 32-bit write data, the IXP2800 processor drives a byte of write data onto the SP\_AD bus in four cycles that are qualified with the rising edge of the SP\_CP signal, with the SP\_OE\_L && SP\_DIR signal being active, i.e., SP\_OE\_L = 0 and SP\_DIR = 1. On every rising edge of the SP\_CP signal with SP\_OE\_L && SP\_DIR asserted, the data is shifted into a register implemented in the glue logic device – least significant byte (LSB) to most significant byte (MSB).

The Verilog\* code in [Example 3](#) depicts an example implementation of the logic:

**Example 3. 32-bit Write Data Packing Logic Implementation**

```
//implementation of latch control for 32-bit pack/unpacking to/from NP
//to uP port

always @(posedge sp_a[0]) begin
    if (~rst_l) begin
        data[31:0]    <= 32'h0000_0000;
    end

    //data is shifted during four cycles

    if (~sp_oe_l && sp_a[1]) begin
        data[31:24]    <= sp_ad_in;
        data[23:16]    <= data[31:24];
        data[15:8]     <= data[23:16];
        data[7:0]      <= data[15:8];
    end

    //here we can latch all the read data, the controller will
    //pulse SPA[0] multiple times but it's a "don't care"
    //as we capture all 32-bits on each edge of SP_CP
```

```

else begin
  if (~sp_rd_l) begin
    data[31:0]   <= uP_rd_data;
  end

  end // else: !if(~sp_oe_l && sp_rd_l)
end // always @ (posedge spa[0])

```

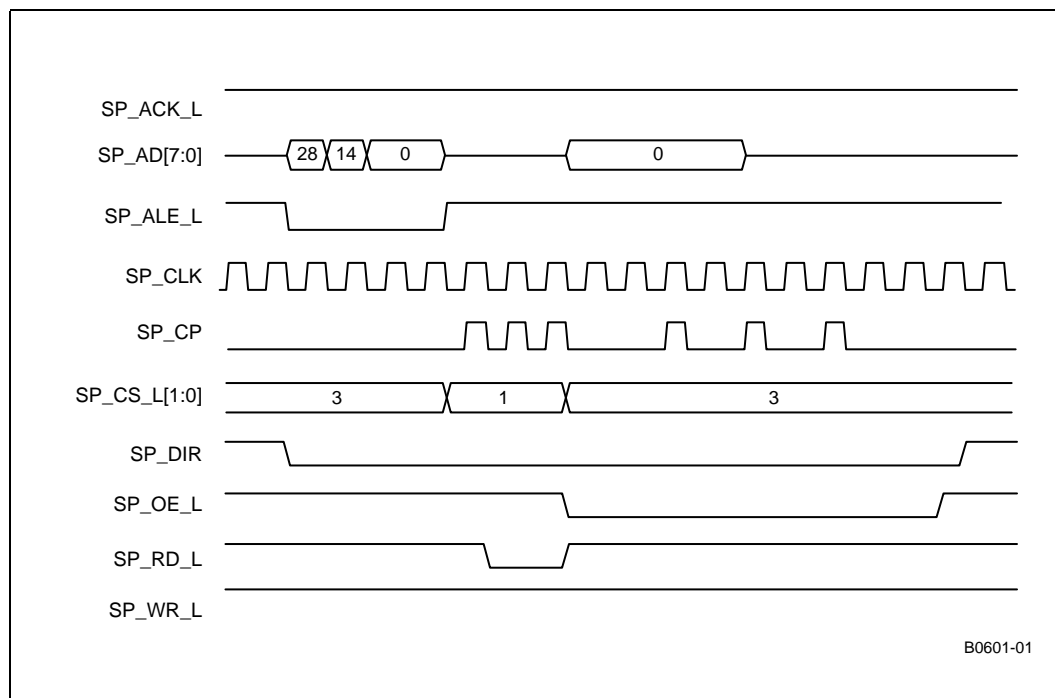
In the code in [Example 3](#), the 32-bit register that is used for latching the write data is also used for latching the read data; this decreases the number of gates needed for implementing the glue logic. If desired, separate registers for latching the write and read data can be used — in which case, the rising edge of the SP\_RD\_L signal is used to capture the read data, and the SP\_CP signal is not used.

### 3.2.2 Read Transactions

As with the writes, the read transactions to 16- and 32-bit devices will need to be unpacked – eight bits per cycle prior to the data being sent back to the IXP2800 via the SP\_AD bus. Again, we will use the SP\_CP, SP\_OE\_L, and SP\_DIR signals to implement the control logic needed to perform the unpacking of the read data. As with the writes, this example will focus on the logic required for Mode 3.

The glue logic is responsible for latching the 32-bit read data coming from the downstream device on the rising edge of the read signal or as specified by the device. [Figure 6](#) shows the timing of the slowport interface signals presented to the glue logic during a read transaction.

**Figure 6. Mode 32-Bit Read Transfer**



The glue logic latches all 32 bits of read data on the rising edge of SP\_CP when SP\_RD\_L is asserted. Note that this may pulse multiple times but this will not matter in our case, as we will latch all the data on each rising edge and hence will always get the last 32 bits of data presented on the data bus. Alternately, the rising edge of the SP\_RD\_L signal could be used in place of SP\_CP to latch the read data; however this requires separate registers for the read and write data, as explained in [Section 3.2.2](#).

The read latching logic is shown in [Example 3](#) as it is combined with the data packing logic and shares the same 32-bit register. After the data is captured, the logic must also unpack the data back to the IXP2800, at eight bits per transfer (MSB to LSB) onto the SP\_AD. This is accomplished by the glue logic driving the first byte of the read data back to the IXP2800 on the sp\_rd\_out bus when the signal shift\_en is asserted.

**Note:** The shift\_en signal is used in combination with a two-bit counter named pack\_cnt, which is used to steer the appropriate byte, based on the count, onto the sp\_rd\_out bus. The pack\_cnt counter is incremented on every rising edge of the SP\_CP if the shift\_en control signal is asserted and it is cleared if shift\_en is de-asserted. This ensures that the pack\_cnt will be zero at the beginning of each read cycle. As the count increments, the appropriate byte is driven onto the sp\_rd\_out bus as SP\_CP is pulsed to complete the 32-bit transfer.

The shift\_en control signal detects if the SP\_OE\_L signal is asserted, if the SP\_RD\_L signal is de-asserted, and that the SP\_DIR signal is de-asserted, which indicates that this is a read cycle and that the glue logic owns the bus. Ultimately, the sp\_rd\_out bus connects to the output port of a bidirectional buffer, which will connect to the SP\_AD bus. The output enable for this buffer can be controlled by SP\_DIR or the shift\_en control signal. The Verilog\* code in [Example 4](#) depicts an example implementation of the shift\_en control signal and the data unpacking logic.

**Note:** In the B stepping of the IXP2800, the SP\_CP signal may pulse after the SP\_RD\_L signal has been deasserted; hence, only the SP\_OE\_L, SP\_DIR, and SP\_CP signals should be used to control the unpacking of the data.

Note that on the clock cycle after the read signal is de-asserted from the downstream device, the glue logic will drive the first byte of data onto the SP\_AD bus, as the IXP2800 does not pulse the SP\_CP signal to promote the first read. The remaining three bytes of data are shifted out on the rising edge of the SP\_CP signal to complete the 32-bit transfer.

#### Example 4. 32-bit Read Un-Packing Logic Implementation

```
// control logic for shifting the 32-bit read data back to IXP, 8-bits
// per cycle
```

```
assign shift_en = (~sp_oe_l && ~sp_a[1]);
```

The Verilog\* code below depicts an example implementation of the data unpacking logic:

```
// implementation of pack_cnt control logic, the count is incremented on each
// rising edge of sp_a[0] if shift_en is active
```

```
always @(posedge sp_a[0]) begin
    if (~rst_l) begin
        pack_cnt <= 2'b00;
    end

    else begin
        if (shift_en)begin
            pack_cnt <= pack_cnt + 1;
        end
    end
end
```

```

        else if (~shift_en) begin
            pack_cnt    <= 2'b00;
        end

        end // else: !if(~rst_l)

    end // always @ (posedge sp_clk)

// Implementation of read data mux, pack_cnt is used to determine which byte
// should be driven onto the sp_rd_out bus

always @ (pack_cnt) begin
    //data is shifted during four consecutive cycles
    case (pack_cnt) //synopsis full_case parallel_case
        2'b00:    sp_rd_out    <= #1 data[31:24];
        2'b01:    sp_rd_out    <= #1 data[23:16];
        2'b10:    sp_rd_out    <= #1 data[15:8];
        2'b11:    sp_rd_out    <= #1 data[7:0];
    endcase // case(pack_cnt)

end

```

## 4.0 Summary

The address latching control logic explained for the PROM and microprocessor ports is essentially the same, with the only difference being it is always a fixed size for the PROM port (24 bits) and is latched during three consecutive clock cycles.

For the microprocessor port, the size of the address is programmable via the SP\_ADC register. Depending on the programmed size of the address, this could be 8 to 25 bits, which will require one to four cycles to latch. In our example, we chose the 25-bit case because this is a complete implementation and can be easily modified to support the other three values.

The data pack/unpacking logic examples were chosen to provide an example of how to interface to an Intel/AMCC\* device. While the other modes have subtle protocol and interface signal differences, the logic used for address latching and data pack/unpacking should be essentially the same. Again, in our example we chose the 32-bit case because this is a complete implementation and can be easily modified to support 8- or 16-bit devices.

Some minor changes may be required for the control logic; however the general concept remains unchanged as it uses the same combination of F377 flip-flops and F646 Octal registers for the glue logic in all modes.