ECP Standard Parallel Interface for DSP56300 Devices

Application Note

by

Mihai V. Micea
Dan Chiciudean
and Lucian Muntean

AN2085/D
Rev. 0, 11/2000
This document contains information on a new product. Specifications and information herein are subject to change without notice.

Motorola reserves the right to make changes without further notice to any products herein. Motorola makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Motorola assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters which may be provided in Motorola data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals" must be validated for each customer application by customer's technical experts. Motorola does not convey any license under its patent rights nor the rights of others. Motorola products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support life, or for any other application in which the failure of the Motorola product could create a situation where personal injury or death may occur. Should Buyer purchase or use Motorola products for any such unintended or unauthorized application, Buyer shall indemnify and hold Motorola and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Motorola was negligent regarding the design or manufacture of the part.

Motorola and are registered trademarks of Motorola, Inc. Motorola, Inc. is an Equal Opportunity/Affirmative Action Employer.

All other tradenames, trademarks, and registered trademarks are the property of their respective owners.

How to reach us:

USA/EUROPE Locations Not Listed: Motorola Literature Distribution; P.O. Box 5405, Denver, Colorado, 80217
1-303-675-2140 or 1-800-441-2447

JAPAN: Motorola Japan, Ltd.; SPS, Technical Information Center, 3-20-1, Minami-Azabu, Minato-ku, Tokyo 106-8573 Japan. 81-3-3440-3569

ASIA/PACIFIC: Motorola Semiconductors H.K. Ltd., Silicon Harbour Centre, 2 Dai King Street, Tai Po Industrial Estate, Tai Po, N.T., Hong Kong. 852-26668334

Customer Focus Center: 1-800-521-6274

The data communication between the DSP and a host computer is one of the important issues in designing DSP-based systems directly connected to PCs which are commonly used as process controllers or as interactive user interfaces. This application note proposes the implementation of a high-performance, yet relatively simple, parallel data communication protocol for a DSP connected to a PC.

This document focuses on the implementation of the Extended Capabilities Port (ECP) parallel communication standard on the DSP56300 family processors.

Specific ECP communication protocols are described along with the characteristic hardware and software implementation details, both on the DSP and on the PC side. The performance evaluation routines designed for the ECP data link are also presented, along with the corresponding results and conclusions.

1 Introduction ................................................. 1

2 DSP-Host Communication Tradeoffs .......................... 1

3 ECP Standard Specifications ................................. 2

  3.1 ECP Hardware Description ................................. 2

  3.2 ECP Handshaking Protocols .............................. 3

    3.2.1 ECP Forward Data Cycle ............................. 4

    3.2.2 ECP Forward Command Cycle .......................... 4

    3.2.3 ECP Reverse Data Cycle ............................. 5

    3.2.4 ECP Reverse Command Cycle .......................... 6

  3.3 ECP Software Registers ................................. 6

4 ECP Interface Implementation—Hardware .................... 8

5 ECP Interface Implementation—Software .................... 9

  5.1 Implementation on the DSP Side .......................... 9

    5.1.1 Initialization ......................................... 9

    5.1.2 Forward Data ......................................... 10

    5.1.3 Reverse Data ......................................... 12

  5.2 ECP Programming on the Host Side ....................... 14

6 Performance Evaluation of the ECP Interface ............... 16

  6.1 Single-Byte Transmission ............................... 16

  6.2 Buffered Transmission ................................... 17

7 Boosting Performance with a DMA Controller ............... 18
1 Introduction

An important set of real-life digital signal processing applications involves a host computer functioning as a system controller or interactive graphical user interface. In these applications, reliable, high-speed data communication between the DSP and the host computer is an important design and implementation issue.

This document describes the implementation of a high-performance, yet relatively simple, parallel data communication protocol for a DSP connected to a PC. The paper focuses on the implementation of the Extended Capabilities Port (ECP) parallel communication standard on the DSP56300 family processors. Specific ECP communication protocols are described, as well as hardware and software implementation details for both the DSP and the PC. Performance evaluation routines designed for the ECP data link are also presented, along with corresponding results and conclusions.

Details of the ECP standard can be found in the Microsoft document, Extended Capabilities Port: Specifications, Revision 1.06.

2 DSP-Host Communication Tradeoffs

The DSP56300 family features various data communication interfaces, suitable for a large variety of system interconnections, through its built-in peripheral ports.

The Serial Communication Interface (SCI) provides a full-duplex port for serial data transfers. Using three dedicated signals (Data Transmit, Data Receive, and Serial Clock), this interface supports industry-standard asynchronous bit rates and protocols up to 115200 bps, as well as high-speed synchronous data transmission up to 8.25 Mbps for a 66 MHz clock. The primary disadvantage of a SCI-based connection with a host computer is that the maximum data transfer rate is limited to 115200 bps by the computer’s serial interface. Another disadvantage is that many DSP-based evaluation modules use the DSP serial port for code development and debugging, making it difficult to develop a DSP program that can initiate serial data transactions autonomously with a host computer.

For high-performance parallel data transfers, the DSP56300 family provides a full-duplex, double-buffered parallel port called the Host Interface. This interface is either 8-bits or 32-bits wide, depending on the particular DSP device selected. The HI08 or HI32 can connect directly to the data bus of a host processor or computer with minimal glue logic, allowing direct data links to microcontrollers such as the Motorola HC11 or the Intel 8051 family, as well as to microprocessors such as the Motorola 68k family or Intel x86 family. The Host Interface can easily connect with the high-speed ISA or PCI bus on a PC to form a communication channel that is substantially faster than a serial connection (up to 16 MBps for the ISA bus, for example). However, this solution requires a very close physical connection, which drastically reduces overall system flexibility.

A good compromise between maximum data transfer speed and system flexibility is the parallel cable interconnection. The standard PC protocol for high-performance parallel communication is the Extended Capabilities Port (ECP). This document describes the hardware and software implementation of the ECP Standard with the HI08 port of a DSP56300 family device.
3 ECP Standard Specifications

The Extended Capabilities Port is a fast, bidirectional parallel interface developed by Microsoft and Hewlett-Packard. It is backward-compatible with the existing PC standard parallel port (SPP) configurations, using the existing parallel connectors and cables.

Pre-existing parallel communication methods utilized a wide variety of hardware and software interfaces, each with unique and incompatible signaling schemes. The ECP standard was developed to provide a standard, open path for communications between computers and more intelligent printers and peripherals. The availability of this standard bidirectional protocol encourages the development of new peripherals that can return both data and status to the host computer.

The ECP protocol generates handshake signals identical to those of the Extended Parallel Port (EPP), and runs at the same speed as EPP. It also supports Run Length Encoding (RLE) to achieve data compression ratios up to 64:1.

In summary, the ECP provides the following features:

- High performance half-duplex forward and reverse channel
- Interlocked handshake for fast, reliable data transfer
- Optional single-byte RLE compression for improved throughput
- Channel addressing for low-cost peripherals
- Active output drivers
- Adaptive signal timing
- Peer-to-peer capability

The ECP provides three operational modes for compatibility with various systems:

- **Compatible mode**—asynchronous, byte-wide forward channel (host-to-peripheral), parallel port interface. Data and status lines are used according to the original SPP and EPP definitions.
- **Nibble mode**—asynchronous, byte-wide reverse channel (peripheral-to-host), parallel port compatible with all existing PC hosts. Data bytes are transmitted as two sequential 4-bit nibbles using four peripheral-to-host status lines.
- **Byte mode**—asynchronous, byte-wide reverse channel using the eight data lines of the interface for data and the control/status lines for handshaking, compatible with IBM PS/2 hosts.

In addition, the ECP offers:

- **ECP mode**—fast bidirectional channel, with or without RLE compression. This mode requires custom hardware.

3.1 ECP Hardware Description

The Extended Capabilities Port uses the industry-standard DB25 connector and is backward-compatible with the SPP and EPP parallel standards. When the ECP operates in SPP or EPP mode, the data and handshake lines operate according to the SPP and EPP definitions.

When the port operates in ECP mode, each of the DB25 connector pins has a unique function, as listed in Table 1.
### 3.2 ECP Handshaking Protocols

The ECP can be fully implemented by hardware with custom parallel communication controllers. It can be implemented in software as well.

The ECP handshaking protocol features the following four cycles:

- **Forward Data**
- **Reverse Data**
- **Forward Command**
- **Reverse Command**

#### Table 1. Pin Assignments for Extended Capabilities Parallel Port Connector

<table>
<thead>
<tr>
<th>Pin</th>
<th>SPP Signal</th>
<th>ECP Signal</th>
<th>Input/Output</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Strobe</td>
<td>HostCLK</td>
<td>Output</td>
<td>A low on this line indicates valid data at the host. When this pin is de-asserted, the +ve clock edge should be used to shift the data into the peripheral.</td>
</tr>
<tr>
<td>2–9</td>
<td>Data 0–7</td>
<td>Data 0–7</td>
<td>I/O</td>
<td>Bidirectional data bus</td>
</tr>
<tr>
<td>10</td>
<td>Ack</td>
<td>PeriphCLK</td>
<td>Input</td>
<td>A low on this line indicates valid data at the peripheral. When this pin is deasserted, the +ve clock edge should be used to shift the data into the host.</td>
</tr>
<tr>
<td>11</td>
<td>Busy</td>
<td>PeriphAck</td>
<td>Input</td>
<td>In reverse channel operation, a high on this pin indicates a data cycle, while a low indicates a command cycle. In forward channel operation, the pin functions as PeriphAck.</td>
</tr>
<tr>
<td>12</td>
<td>Paper Out / End</td>
<td>nAckReverse</td>
<td>Input</td>
<td>The peripheral pulls this pin low to acknowledge a reverse request.</td>
</tr>
<tr>
<td>13</td>
<td>Select</td>
<td>X-Flag</td>
<td>Input</td>
<td>Extensibility Flag</td>
</tr>
<tr>
<td>14</td>
<td>Auto Linefeed</td>
<td>Host Ack</td>
<td>Output</td>
<td>In forward channel operation, a high on this pin indicates a data cycle, while a low indicates a command cycle. In reverse channel operation, the pin functions as HostAck.</td>
</tr>
<tr>
<td>15</td>
<td>Error / Fault</td>
<td>PeriphRequest</td>
<td>Input</td>
<td>The peripheral pulls this pin low to indicate that reverse data is available.</td>
</tr>
<tr>
<td>16</td>
<td>Initialize</td>
<td>nReverseRequest</td>
<td>Output</td>
<td>This pin is pulled low to indicate that data is in the reverse direction.</td>
</tr>
<tr>
<td>17</td>
<td>Select Printer</td>
<td>1284 Active</td>
<td>Output</td>
<td>The host pulls this pin high to indicate 1284 transfer mode, and pulls the pin low to terminate.</td>
</tr>
<tr>
<td>18–25</td>
<td>Ground</td>
<td>Ground</td>
<td>GND</td>
<td>Ground</td>
</tr>
</tbody>
</table>

[Table 1. Pin Assignments for Extended Capabilities Parallel Port Connector]
3.2.1 ECP Forward Data Cycle

In the forward data cycle, the host sends a single byte of compressed data to the peripheral. This cycle consists of six steps:

1. Host places data on data lines.
2. Host asserts HostAck to indicate the start of a data cycle.
3. Host asserts HostClk to low to indicate valid data.
4. Peripheral asserts PeriphAck to acknowledge valid.
5. Host deasserts HostClk high. The +ve edge is used to shift data into the peripheral.
6. Peripheral de-asserts PeriphAck to acknowledge receiving the byte.

These steps are illustrated in Figure 1.

![Figure 1. ECP Forward Data Cycle](image)

3.2.2 ECP Forward Command Cycle

In the forward command cycle, the host sends a channel address to the peripheral. This cycle consists of six steps:

1. Host places data on data lines.
2. Host deasserts HostAck to indicate the start of a command cycle.
3. Host asserts HostClk low to indicate valid data.
4. Peripheral asserts PeriphAck to acknowledge valid data.
5. Host deasserts HostClk high. The +ve edge is used to shift data into the peripheral.
6. Peripheral deasserts PeriphAck to acknowledge receiving the byte.

These steps are illustrated in Figure 2.
3.2.3 ECP Reverse Data Cycle

In the reverse data cycle, the peripheral sends a single byte of data to the host. There are eight steps specified for the reverse data cycle:

1. Host sets nReverseRequest Low to request a reverse channel.
2. Peripheral asserts nAckReverse low to acknowledge reverse channel request.
3. Peripheral places data on data lines.
4. Peripheral pulls PeriphAck high to select data cycle.
5. Peripheral pulls PeriphClk low to indicate that valid data is present.
6. Host pulls HostAck high to acknowledge that valid data is present.
7. Peripheral pulls PeriphClk high. The +ve edge is used to shift data into the host.
8. Host deasserts HostAck low to acknowledgment receipt of the byte.

These steps are illustrated in Figure 3.
### 3.2.4 ECP Reverse Command Cycle

In the reverse command cycle, the peripheral sends a channel address to the host. There are eight steps specified for the reverse command cycle:

1. Host sets nReverseRequest Low to request a reverse channel.
2. Peripheral asserts nAckReverse low to acknowledge reverse channel request.
3. Peripheral places data on data lines.
4. Peripheral pulls PeriphAck low to select command cycle.
5. Peripheral pulls PeriphClk low to indicate that valid data is present.
6. Host pulls HostAck high to acknowledge that valid data is present.
7. Peripheral pulls PeriphClk high. The +ve edge is used to shift data into the host.
8. Host deasserts HostAck low to acknowledgment receipt of the byte.

These steps are illustrated in Figure 4.

![ECP Reverse Command Cycle Diagram](image)

**Figure 4. ECP Reverse Command Cycle**

### 3.3 ECP Software Registers

The software registers implemented in a standard PC on the host side of the ECP interface are listed in Table 2. The first 3 registers are identical to those in the Standard Parallel Port.
### Table 2. ECP Registers

<table>
<thead>
<tr>
<th>Address</th>
<th>Port Name</th>
<th>Read/Write</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Base + 0</td>
<td>Data (SPP)</td>
<td>Read/Write</td>
<td>Standard parallel port data register. Writing to this register in SPP mode drives data on the parallel port data lines. In all other modes the drivers can be tri-stated by setting the direction bit in the Control Register.</td>
</tr>
<tr>
<td></td>
<td>ECP Address FIFO (ECP mode)</td>
<td>Read/Write</td>
<td>Data written to this address is placed in the FIFO and tagged as ECP Address/RLE. ECP port hardware transmits the byte to the peripheral automatically.</td>
</tr>
<tr>
<td>Base + 1</td>
<td>Status Register</td>
<td>Read</td>
<td>Reflects the inputs on the parallel port interface.</td>
</tr>
<tr>
<td>Base + 2</td>
<td>Control Register</td>
<td>Read/Write</td>
<td>Directly controls several output signals, sets the direction of communication, and enables an interrupt on the rising edge.</td>
</tr>
<tr>
<td>Base + 400h</td>
<td>Data FIFO (Parallel Port FIFO Mode)</td>
<td>Read/Write</td>
<td>In Parallel Port FIFO Mode, any data written to the Data FIFO is sent to the peripheral using the SPP handshake; hardware generates the handshaking required.</td>
</tr>
<tr>
<td></td>
<td>Data FIFO (ECP Mode)</td>
<td>Read/Write</td>
<td>When data direction is 0 (output to peripheral), bytes written or DMAed from the system to this FIFO are transmitted to the peripheral by hardware handshake using the ECP parallel port protocol. When data direction is 1 (input from peripheral), bytes from the peripheral are read into this FIFO under automatic hardware handshake from ECP.</td>
</tr>
<tr>
<td></td>
<td>Test FIFO (Test Mode)</td>
<td>Read/Write</td>
<td>Data can be read, written, or DMAed to or from the system to this FIFO in any direction; data is not transmitted to the parallel port lines using a hardware protocol handshake but can be displayed on the parallel port data lines.</td>
</tr>
<tr>
<td></td>
<td>Configuration Register A (Configura-</td>
<td>Read/Write</td>
<td>Indicates if the card generates level- or edge-triggered interrupts and the bus widths within the card, and determines if there are any bytes left in the FIFO. Configuration Register A is accessible only when the ECP Port is in Configuration Mode.</td>
</tr>
<tr>
<td></td>
<td>tion Mode)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Configuration Register B (Configur-</td>
<td>Read/Write</td>
<td>Selects compression option (RLE) for outgoing data, returns the status of the IRQ pin, IRQ assignment and DMA Channel assignment. Configuration Register B is accessible only when the ECP Port is in Configuration Mode.</td>
</tr>
<tr>
<td></td>
<td>ation Mode)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Extended Control Register</td>
<td>Read/Write</td>
<td>Configures the ECP mode and returns the status of the ECP FIFO. Modes of operation include:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Standard Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Byte / PS/2 Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Parallel Port FIFO Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• ECP FIFO Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• EPP Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• FIFO Test Mode</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Configuration Mode</td>
</tr>
</tbody>
</table>
4 ECP Interface Implementation—Hardware

The hardware connection between the PC and DSP is a standard DB25 parallel cable. On the host side, the cable connects to the standard parallel port, configured for the ECP protocol. On the DSP side, the cable connects to a male DB25 header, which is directly connected to the HI08 port as illustrated in Figure 5.

In this implementation, all HI08 signals are configured as General Purpose I/O lines (GPIO) during the ECP initialization routine for the DSP. The Host Interface data lines (HAD0–HAD7) are assigned to the ECP data bus (D0–D7). The ECP handshake signals are connected to the HI08 port as shown in Table 3.

Table 3. ECP to HI08 Pin Assignments

<table>
<thead>
<tr>
<th>ECP Signal</th>
<th>Direction</th>
<th>HI08 Signal</th>
</tr>
</thead>
<tbody>
<tr>
<td>HostClk</td>
<td></td>
<td>HAS/A0</td>
</tr>
<tr>
<td>Data 0–7</td>
<td></td>
<td>HAD0–HAD7</td>
</tr>
<tr>
<td>PeriphClk</td>
<td></td>
<td>HA8/A1</td>
</tr>
<tr>
<td>PeriphAck</td>
<td></td>
<td>HA9/A2</td>
</tr>
<tr>
<td>nAckReverse</td>
<td></td>
<td>HRW/RD</td>
</tr>
<tr>
<td>HostAck</td>
<td></td>
<td>HDS/WR</td>
</tr>
<tr>
<td>nReverseRequest</td>
<td></td>
<td>HCS/A10</td>
</tr>
</tbody>
</table>

In some cases, the electrical signal coupling between the DSP and the host computer may require voltage level shifters to accommodate the 3.3V DSP56300 family I/O port voltage levels and the 5V (TTL) parallel interface signals of the host computer. Tests run with the ECP implementation on the DSP56303 issued satisfactory results without such voltage level interface buffers, but special care must be taken with each DSP56300 family device.
5 ECP Interface Implementation—Software

The ECP handshake signals required to control data transactions between the DSP and the host computer can be generated either in hardware, using a customized ECP controller, or in software. This section describes the latter approach for the DSP side, in which software polls the HI08 port to generate the ECP handshake signals. The host side employs a hardware controller, but does require some initialization software which is also described.

5.1 Implementation on the DSP Side

The ECP standard defines four main protocol phases: ECP forward data and reverse data cycles for data transactions, and ECP forward command and reverse command cycles to indicate single-byte data compression or channel address. This section presents DSP software for the two data cycles, which are the basic requirement for the ECP communication, as well as an initialization routine.

5.1.1 Initialization

The DSP initialization routine includes:

- Configuring the HI08 port as GPIO
- Setting the direction of each line used for handshaking
- Generating the correct logical levels on the output lines
- Initializing the HI08 lines that connect to the ECP data bus as inputs

The code for this routine is presented in Example 1.
### Example 1. ECP Initialization Routine

```assembly
Example 1. ECP Initialization Routine

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Address</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hpcr</td>
<td>$ffffc4</td>
<td>Host Port Control Register Address</td>
</tr>
<tr>
<td>hddr</td>
<td>$ffffc8</td>
<td>Host Data Direction Register address</td>
</tr>
<tr>
<td>hdr</td>
<td>$ffffc9</td>
<td>Host Data Register address</td>
</tr>
<tr>
<td>HOSTCLK</td>
<td>EQU 8</td>
<td>The next 6 equates are used for</td>
</tr>
<tr>
<td>PERIPHCLK</td>
<td>EQU 9</td>
<td>;addressing the handshaking lines in</td>
</tr>
<tr>
<td>PERIPHACK</td>
<td>EQU 10</td>
<td>;Host Data Register</td>
</tr>
<tr>
<td>nACKREVERSE</td>
<td>EQU 11</td>
<td></td>
</tr>
<tr>
<td>HOSTACK</td>
<td>EQU 12</td>
<td></td>
</tr>
<tr>
<td>nREVERSEREQUEST</td>
<td>EQU 13</td>
<td></td>
</tr>
</tbody>
</table>

```assembly
init_ecp
bset #0, x:hpcr
bclr #HOSTCLK, x:hddr
bset #PERIPHCLK, x:hddr
bset PERIPHCLK, x:hdr
bset PERIPHACK, x:hdr
bclr PERIPHACK, x:hdr
bset nACKREVERSE, x:hdr
bclr nACKREVERSE, x:hdr
bclr HOSTACK, x:hddr
bclr nREVERSEREQUEST, x:hddr
bsr line_in
rts

```assembly
line_in
move x:hddr,a
and #$ff00,a
move a,x:hddr
rts
```

### 5.1.2 Forward Data

For the ECP forward data cycle, the DSP implements a read routine, which is called each time a byte is to be transferred from the host to the DSP. Figure 6 is a flowchart of the read routine.
In accordance with ECP specifications, the read routine performs a blocking scan of the nReverseRequest (‘data direction’) and HostClk (‘data valid’) lines. If the host pulls the data direction line low, indicating a host request to read data from the DSP, a “time-out” condition occurs; the program exits the read routine and goes to the write routine. Otherwise, the read routine waits for a byte to be sent by the host, indicated by a high on the HostClk line.

The code for the read routine is given in Example 2 on page 12.
Example 2. ECP Read Routine

```
read
  jclr #nREVERSEREQUEST,x:hdr,*       ;Wait for nRevReq line to
  bset  #PERIPHCLK,x:hdr              ;go high and then sets
       ;the PeriphClk line.

  jclr #nREVERSEREQUEST,x:hdr,rd1     ;Check if nRevReq is low
  jset  #HOSTCLK,x:hdr,rd1            ;and if so, a time out
       ;condition occurred.
  bset  #PERIPHACK,x:hdr              ;If HostClk is low then
       ;sets PeriphClk and
       ;continue.

  jclr #nREVERSEREQUEST,x:hdr,rd2     ;Wait for HostClk to go
  bsr  data_in                        ;high and then samples
       ;the data bus.

  bclr  #PERIPHACK,x:hdr              ;Reset the PeriphAck line

  move x:hdr,a0                       ;Get one byte from data
  extractu #$8000,a,a                  ;bus and store it in a0.

rd1

  bset #PERIPHACK,x:hdr

rd2

  bclr #PERIPHACK,x:hdr

read_end

rts
```

5.1.3 Reverse Data

For the ECP reverse data cycle, the DSP implements a write routine in accordance with the corresponding ECP protocol specifications. Figure 7 is a flowchart of the read routine.
Figure 7. ECP Write Routine Flowchart
The write routine performs a non-blocking scan on the nReverseRequest (‘data direction’) handshake line. If the line is high, indicating that the host is in the Forward Data phase of ECP operation and is not ready to receive the data byte from the DSP, the program exits the write routine and goes to the read routine. Otherwise, the write routine sends the byte out on the ECP data bus.

The code for the write routine is given in Example 3.

---

**Example 3. ECP Write Routine Code**

```assembly
write
  jset #nREVERSEREQUEST,x:hdr,write_end ;If nRevReq line is low,
  bclr #nACKREVERSE,x:hdr          ;;exit from write routine,
  bsr data_out                     ;;else reset nAckRev,
                                ;;output the data on data
  bset #PERIPHACK,x:hdr            ;;lines.
  bclr #PERIPHCLK,x:hdr            ;Assert PeriphAck,
wr1  jset #nREVERSEREQUEST,x:hdr,time_out_wr
  jclr #HOSTACK,x:hdr,wr1          ;Wait for HostAck to go
  bset #PERIPHCLK,x:hdr            ;;high and set PeriphClk.
wr2  jset #nREVERSEREQUEST,x:hdr,time_out_wr
  jset #HOSTACK,x:hdr,wr2          ;Wait for HostAck to go
  bclr #PERIPHACK,x:hdr            ;;low.
  time_out_wr
  bsr line_in                      ;;Switch the data bus
                                ;;direction to in.
  bclr #PERIPHACK,x:hdr            ;Reset PeriphAck and
  bset #nACKREVERSE,x:hdr          ;;set nAckRev.
write_end
  rts

data_out
  move a0,x0                       ;Take the least
                                ;;significant byte from
  move x:hdr,a0                    ;;a0 and send it,
  insert $8000,x0,a                ;;without modifying
  move a0,x:hdr                    ;;the most significant
  move x:hdr,a1                    ;;byte from Host Data
  or $ff,a                         ;;Register.
  movev a1,x:hdr                   
  rts
```

---

### 5.2 ECP Programming on the Host Side

The host computer generates the ECP communication protocol with a hardware controller on the motherboard, so the only programming requirements are proper ECP controller initialization and a pair of routines for data transmission and reception between the ECP port and the user application.

Example 4 presents all the initialization routines needed for data communication through the host ECP port. User applications should call these functions in a similar manner to the communication evaluation program described in Section 6, “Performance Evaluation of the ECP Interface,” on page 16.
Example 4. Host ECP Initialization Routines

#define b_addr 0x378

unsigned char Read_cnfg(int reg){
    return inb(b_addr+0x400+reg);
}

void Write_cnfg(int reg,unsigned char x){
    outb(x,b_addr+0x400+reg);
}

Write_cnfgA(unsigned char x){
    Write_cnfg(0,x);
}

Write_cnfgB(unsigned char x){
    Write_cnfg(1,x);
}

unsigned char Read_cnfgB(void){
    return Read_cnfg(1);
}

//Initialize the parallel port in ECP mode.
void init_ecp(){
    outb(0x34,b_addr+0x402);
    outb(0xf4,b_addr+0x402);
    Write_cnfgA(0x14);
    Write_cnfgB(Read_cnfgB()&0x7f);
}

//Place the ECP in Forward phase.
void init_write(){
    unsigned char x;
    outb(0x34,b_addr+0x402);
    outb(inb(b_addr+2)&0x0C,b_addr+2);
    x=inb(b_addr+2);
    outb(0x34,b_addr+0x402);
    outb(0x75,b_addr+0x402);
    outb((x|4),b_addr+2);
    while ((!inb(b_addr+1))&(1<<6));
}

//Place the ECP in Reverse phase.
void init_read(){
    unsigned char x;
    outb(0x34,b_addr+0x402);
    outb(inb(b_addr+2)|0x20,b_addr+2);
    x=inb(b_addr+2);
    outb(0x75,b_addr+0x402);
    outb((x&0xfb),b_addr+2);
    while((!inb(b_addr+1))&(1<<5));
}

//Test if ECP FIFO is empty.
int fifo_empty(){
    return (inb(b_addr+0x402)&1);
}

//Test if ECP FIFO is full.
int fifo_full(){
    return (inb(b_addr+0x402)&2);
}
6 Performance Evaluation of the ECP Interface

This section describes a time-based, data counting performance evaluation program for the ECP interface, and presents results for forward and reverse data transfer modes.

To evaluate data communication performance through the ECP interface, programs must be developed for both the host and the DSP. The performance parameters of primary concern are the occurrence of errors during the transfer and the data transfer rate. To determine the maximum transfer rate of the proposed ECP implementation, a large amount of data must be sent through the parallel link with minimal interference from other activities during the data transfer.

6.1 Single-Byte Transmission

A first approach is to transmit one byte at a time. For reverse mode, the ECP test routine on the DSP side simply transmits a byte of data on the ECP in an infinite loop, incrementing the value sent after each transferred byte.

Example 5 lists the reverse mode ECP test routine on the DSP.

Example 5. DSP-Side Performance Evaluation Routine—Single-Byte Transfer, Reverse Mode

```
ECP_test
  bsr init_ecp
again
  move b,a
  jsr write
  inc b
  jmp again
```

A similar algorithm is used for the ECP forward mode.

The actual ECP performance evaluation program resides on the host computer. The host routine measures the time it takes to transfer a large amount of data through the ECP interface, checks the data, and counts the errors that occur. The routine for reverse mode evaluation includes the following steps:

1. Initialize the host parallel port controller for ECP mode.
2. Configure the ECP for reverse data mode to receive data from peripheral.
3. When the port receives the first data byte, start the communication timer and initialize the error counter.
4. After receiving a predefined number of data bytes from the DSP, calculate the overall data transfer speed and display the error count.

These steps are incorporated in the test routine in Example 6.
void main(){
    unsigned char ch,old_ch;
    int error_no=0,i;
    time_t t1,t2;
    long l;

    init_ecp(); //Initialize the ECP controller
    init_read(); //Place ECP in Reverse phase
    while(fifo_empty()); //Wait for ECP FIFO to receive a byte
    old_ch=inb(b_addr+0x400); //Read that byte
    t1=time(NULL); //Store the transfer begin time
    for (l=0;l<4096000;l++){
        while(!fifo_full()); //Wait for ECP FIFO to fill
        for(i=0;i<16;i++){
            ch=inb(b_addr+0x400); //Read a byte
            if (ch!=old_ch++) //Compare with previous value
                error_no++; //and increment the error counter
            old_ch=ch; //if an error occurred
        }
    }
    t2=time(NULL); //Store the transfer end time
    printf("Errors=%d",error_no); //Display the error counter and
    printf("Speed=%f",(double)4096000.0*16.0/(t2-t1)); //transfer speed
}

Again, a similar algorithm is employed for the ECP forward mode.

A test system incorporating the programs in Example 5 and Example 6 was run in both forward and
reverse mode, yielding results shown in Table 4.

<table>
<thead>
<tr>
<th>Transfer Direction</th>
<th>Byte Count</th>
<th>Transfer Rate</th>
<th>Error Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Forward Mode</td>
<td>62.5 Mega</td>
<td>590 KBps</td>
<td>0</td>
</tr>
<tr>
<td>Reverse Mode</td>
<td>62.5 Mega</td>
<td>451 KBps</td>
<td>0</td>
</tr>
</tbody>
</table>

The difference between the forward mode and reverse mode transfer rates in Table 4 is due to the fact that
the reverse data routine requires more DSP instruction cycles than the forward data routine.

6.2 Buffered Transmission

A second, more practical approach is to implement a buffered-type data transfer through the ECP interface.
The code in Example 7 on page 18 is a simple DSP routine for reverse mode that transfers an entire buffer
to the host computer. In this example, the ‘buffer’ variable represents the base address of the data buffer to
be sent and the ‘buf_len’ variable represents the buffer length. The routine scans the buffer in one-step
(word) increments and calls the ECP ‘write’ routine described in Example 3 on page -14 to send the least
significant byte of each word in the buffer to the host. These routines can be interrupted by other
asynchronous events such as timer interrupts, serial, or DMA transfers, etc.
Example 7. DSP-Side Performance Evaluation Routine—Buffer Transfer, Reverse Mode

```
Send_Buffer
    move    #buffer,r0          ;Initialize pointer to the start of the buffer.
    do      #buf_len,loop_sb    ;Initialize the Loop Counter (LC) Register with
                           ;buffer length.
    move    x:(r0)+,a0          ;Prepare the current data word (24-bits) to be sent.
    bsr     write               ; Send the 8 least significant bits of A0 register.

loop_sb
    rts
```

In the same manner, one can easily implement the corresponding ‘Receive_Buffer’ routine on the DSP using the ‘read’ routine described in Example 2 on page -12.

To transfer 16-bit words, the ‘Send_Buffer’ routine can be modified as shown in Example 8. Similar modifications can be used for forward transfers, and to send or receive 14-bit words.

Example 8. 16-bit Word Buffer Send Routine

```
Send_Buffer_16
    move    #buffer,r0
    do      #buf_len,loop_sb
    move    x:(r0)+,a0
    bsr     write
    extract $8008,a,a          ; Replace the 8 least significant bits in A0 with the
                           ; next 8 more significant ones, to be used further
                           ;; by ‘write’.
    bsr     write

loop_sb
    rts
```

7 Boosting Performance with a DMA Controller

The performance of an ECP-based communication system can be substantially enhanced by incorporating a dedicated ECP controller with a DMA interface into the DSP56300-based system. A typical connection of a DMA-capable ECP controller (such as the PPC34C60 from Standard Microsystem Corporation or the W91284PIC from Warp Nine Engineering) to a DSP56300 family processor is shown in Figure 8.
As the master of the ECP data link, the host computer uses the ECP control lines to select the type of data transfer (forward or reverse) and initiates the transactions. Thus, the ECP Controller always initiates the DMA cycles with the DSP by asserting DMAREQ (DMA transfer request) and DMADIR (direction of the transfer). The actual DMA transfer begins when the DSP asserts DMAACK (DMA transfer acknowledge) and CS. The DMA transaction ends when the DSP asserts the TC (terminal count) signal.

On the DSP side, some preliminary configuration and initialization is required to perform DMA transactions with the ECP controller, including the following steps:

- Port A Data lines are used for data transfers.
- Port A Address lines specify the location of the transferred data.
- Port A RD and WR signals command the read and write cycles, respectively.
- Port A AA (One of the four Address Attribute 0:3 lines functions as the chip select signal for the ECP controller.
- Interrupt line IRQ serves as the DMA request signal from the ECP controller.
- An interrupt handler for the DMA request is installed.
- Three HI08 (Port B) signals are configured as GPIO to function as follows:
  - DMAACK (output)
  - DMADIR (input)
  - TC (output)

When the host initiates an ECP reverse data cycle, (i.e., reads a data buffer from the DSP), the following steps are executed:

1. Host configures the ECP controller and initiates the ECP reverse data cycles.
2. ECP controller asserts DMAREQ.
3. The DSP interrupts the execution of its current program and handles the corresponding interrupt as follows:
a) Reads the DMADIR to determine the direction of data transfer.
b) Prepares the data buffer to be sent.
c) Configures the on-chip DMA controller with the start address of the data buffer and the total number of data words.
d) Starts the DMA transfer and asserts DMAACK.

4. DSP resumes the current program execution while the DMA transfer is performed in parallel.
5. When the buffer is transferred, the on-board DMA controller issues an internal interrupt.
6. The DSP interrupted to assert TC (end of DMA transfer) and reset the internal DMA controller.
7. DSP resumes the execution of the current program.

A similar operation is executed for an ECP forward data cycle initiated by the host.

8 Conclusion

The ECP interface enables medium to high-speed parallel data transfers between a host computer and a DSP-based application without the physical limitations of a direct connection between the data buses of the host and DSP. This solution is easy to implement, requiring little or no additional hardware beyond the standard DB25 parallel connector and minimal software development. System performance can be enhanced by using DMA transfers between the host ECP controller and the DSP.
About the Authors

Mihai V. Micea is a lecturer at the Computer Software and Engineering Department at the Politehnica University of Timisoara, and Executive Director of the DSP Applications Lab Timisoara (DALT) sponsored by Digital DNA from Motorola.

Dan Chiciudean and Lucian Muntean are students at the Automation and Computer Science Faculty at the Politehnica University of Timisoara, and members of the research and development team at DALT.

Contacts:
- micha@dsplabs.utt.ro
- http://dsplabs.utt.ro/dalt/

References

[8] Interfacing the PC. Interfacing the Extended Capabilities Port, Craig Peacock, February 28, 2000, Internet Resource.