HEWLETT-PACKARD

HP 82940A

# GPIO Interface

OWNER'S MANUAL

SERIES 80

HEWLETT
PACKARD

# HP 82940A
# GPIO Interface

## Owner's Manual

## Series 80

**September 1982**

Reorder Number
82940-90011

# Contents

Section 1

# General Description

## Introduction

The Hewlett-Packard GPIO Interface is a general-purpose interface that provides 16-bit or dual 8-bit data exchange between HP Series 80 Personal Computers and peripheral devices. The interface can transfer data in the "half-duplex" mode, which requires either an input or output handshake.

The hardware and software characteristics of the interface are extremely flexible. The data exchange timing and logic can be configured by switches or dynamically under program control to meet a wide variety of peripheral requirements.

This manual contains the general description, installation, hardware theory of operation and software information and operating instructions.

## I/O ROM

In order to make use of the capabilities of the HP 82940A GPIO Interface, an I/O ROM is required. The I/O ROM (P/N 00085-15003 for the HP-85/83 or P/N 00087-15003 for the HP-87) plugs into an HP 82936A ROM Drawer which is installed into a port at the back of the HP Series 80 Personal Computer.

## Specifications

The HP 82940A GPIO Interface provides two 8-bit, low-power, bi-directional ports (A and B), and two 8-bit, output only ports (C and D). Each port uses a two wire handshake for I/O data, and each port can be operated independently of the others in byte mode. In word mode, the interface can also be configured for 16-bit word mode operation. Data transferred via ports A and B use the A or B handshake lines. Data transferred via ports C and D may also use the CTL0/ST0 or the CTL1/ST1 handshake lines.

Two general purpose control bits can be written, and two status bits can be read. Depending on the application, port A or port B handshake lines can also be used as general purpose control and status bits.

The interface provides eight different hardware configurations for eight-bit ports, all of which may be selected by program statements.

The HP 82940A GPIO Interface consists of one circuit board mounted in a case. The case plugs into any one of the computer I/O ports. A 24-conductor, shielded cable is provided for connection to the peripheral device. The cable is approximately 4 meters (13 feet).

**Table 1-1. Hardware Specifications**

| | |
|---|---|
| **Data Lines:** | 16 low-power bi-directional, DA0-DA7 (port A), DB0-DB7 (port B), and 16 output only, DC0-DC7 (port C), DD0-DD7 (port D) |
| **Control Lines:** | 4 Control, 2 Flag, 2 Status |
| **Signal Lines:** | TTL and open collector |
| **Dimensions** | Approximately 16.7 × 12.7 × 1.5 cm (6.59 × 5 × 0.59 in). |
| **Weight** | 0.5 kg (1.1 lb) |
| **Cable Length** | Approximately four meters. (13.12 feet) |
| **Operating Temperature** | 0° to 55°C (32° to 131°F). |
| **Power Requirements** | The computer mainframe supplies all power for the interface via the plug-in module ports on the back panel. |

Section 2

# Installation

## Unpacking and Inspection

If the shipping carton is damaged, ask the carrier's agent to be present when the interface is unpacked. If the interface is damaged or fails to meet electrical specifications, immediately notify the carrier and the nearest HP sales and service office. Retain the shipping carton for the carrier's inspection. The sales and service office will arrange for the repair or replacement of your interface without waiting for the claim against the carrier to be settled.

## Installation

The complexity of the installation procedure depends on the device to be interfaced to the computer. The interface will have to be configured, and its cable properly prepared to meet your needs. This involves cutting the cable to the proper length, proper termination of the cable wires and the setting of default and select code switches located within the interface case. The following paragraphs will instruct you in configuring and preparing your interface.

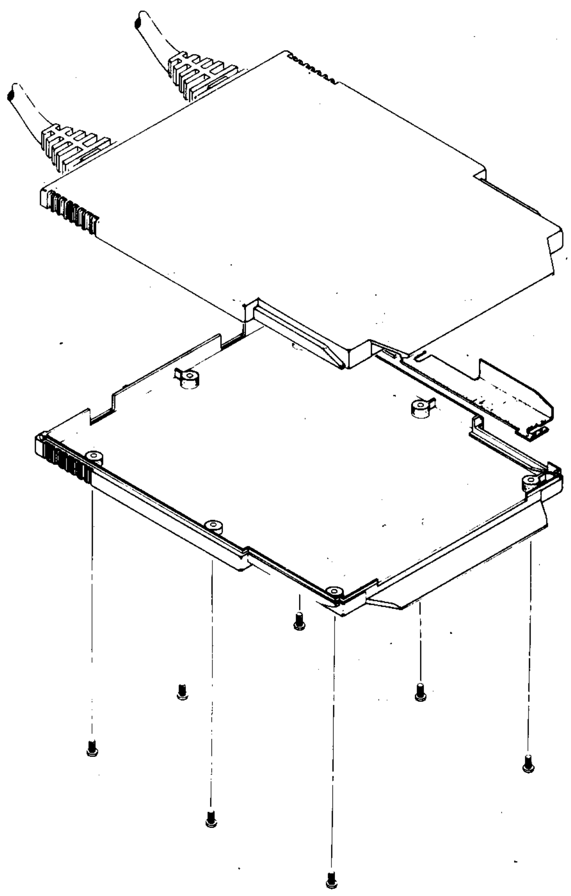


**Figure 2-1. GPIO Interface Disassembly**

## Disassembly and Preparation of the GPIO Interface Cable

The GPIO interface is shipped with one 4-meter-long, 24-conductor, shielded cable with both ends terminated to the interface.

The following steps are given for preparing the interface cable prior to installation. Refer to figure 2-7 for the connector pin-outs and wire color codes. Reverse the disassembly procedure to reassemble the interface.

1. Place the interface on a flat surface with the bottom cover facing up (screw heads showing).
2. Remove the seven screws shown in figure 2-1 and lift the cover, being careful not to lose the ground clip.
3. Carefully lift the circuit board out of the top cover and place on a flat surface, with the component side up, and the cable on the left (as shown in figure 2-2).
4. Determine the required cable length or lengths for your particular application. Remember that cable A contains the port A and port C data and handshake lines, and cable B contains the port B and port D data and handshake lines.

   The maximum supported cable length (from card to peripheral after installation) is 4 meters. After installation, this will give one cable that is 4 meters long, or two cables that are 2 meters long, or any combination in between. If two cables longer than 2 meters each is required, another 4-meter cable can be ordered. The cable part number is 8120-3190.
5. Cut the cable, being sure to leave some extra length for slack.

**Note:** After the cable has been cut, all unused wires must be properly terminated.

6. If an entire cable (A or B) is not to be used, either remove the unused portion of that cable by removing the connector, or sleeve each wire in that unused cable to prevent shorting of the wires.

**Note:** Removal of an unused cable will leave a hole at the rear of the interface case after reassembly. This hole should be covered with electrical tape to keep out foreign material.

7. Determine the select code and default configuration that your peripheral will require and set the appropriate switches, as shown in figures 2-3 and 2-4.
8. After the select code and default switches have been set and the cable has been cut, re-install the circuit board in the case with the ground clip properly positioned. Reverse steps 1 through 3 of the disassembly instructions to re-assemble the interface.

   Refer to figure 2-6 for instructions 9, 10 and 11.

9. Strip off the outer plastic jacket about 10 cm (4 inches).
10. Cut off the outer shield even with the end of the outer plastic jacket.
11. Cover the end of the jacket and outer shield with heat-shrink tubing or electrical tape.
12. For the following conditions, sleeve the lines individually and leave them floating:

    - If some or all of the lines in an output only port (port C and/or port D) are not used.
    - If all of the lines in a bi-directional port are not used.

13. If some of the lines of a bi-directional port (used for input only) are not used and a low level is required on the unused lines, the unused lines should be pulled low by a 1 kΩ resistor to ground. Use one resistor per line.

Note: When some of the lines of a bi-directional port are tied low, an output operation on that port should not be attempted.

The connector pin-outs for the GPIO are shown in figure 2-5.

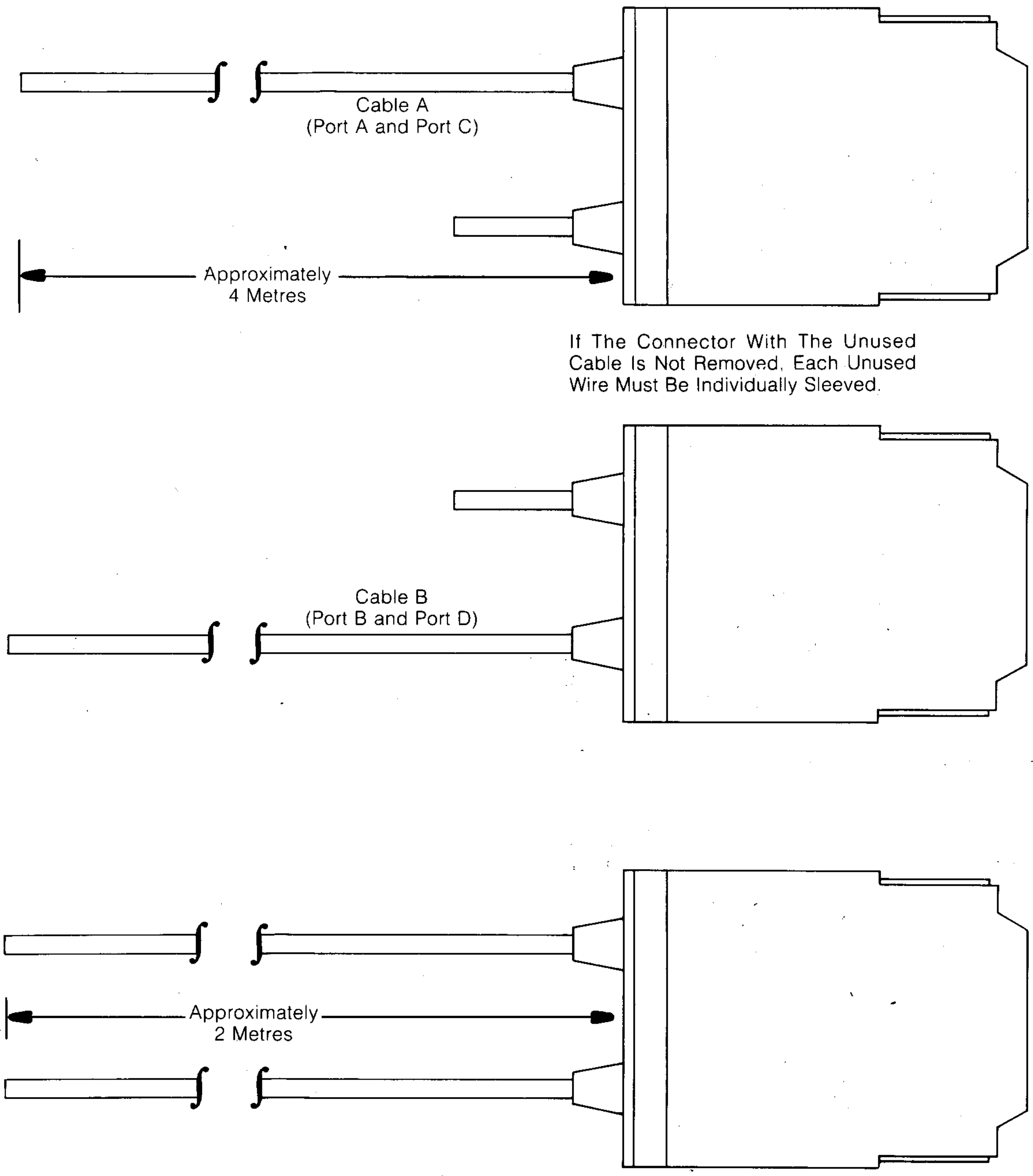

Cable A
(Port A and Port C)
Approximately
4 Metres
If The Connector With The Unused
Cable Is Not Removed, Each Unused
Wire Must Be Individually Sleeved.
Cable B
(Port B and Port D)
Approximately
2 Metres

**Figure 2-2. Preparing the GPIO Interface**

The select code switches and default switches are set at the factory as follows:

| | |
|---|---|
| Select Code | = 4 |
| Device Address | = 6 |
| Handshake Mode | = Full |
| Output Enable Register | = Not Enabled |
| Data, Flag/Status and CTL Logic Sense | = Positive True Logic |

Verifying or changing the select code or default conditions requires disassembly of the interface housing. If this is necessary, refer to the following disassembly procedure and to the discussion of the function requiring change covered in this section.

## Select Code

The select code is preset to "4" at the factory. To change this setting, it is necessary to change the position of the switch segments 2, 3, and 4 of switch S1. Segment 1 of S1 is not used. Select codes 3 through 10 may be set with these switches. Select codes 1 and 2 are reserved for the computer CRT and internal printer, respectively.

The select code and configuration (default) switches found on your interface card may be either the slide or rocker type. To set the switch segments to "1", either depress the "1" side (as marked on the circuit card) of the rocker switch or slide the tab of the slide switch to the "1". To do this, use the point of a pencil or other similar object. The select code, which is preset at the factory, is "4" and shown selected in figure 2-3 for both switch types.

**Note:** If you change any of the factory settings, make sure that you change the proper switch segments. Do not disturb the settings of adjoining switches. The small tip of a pencil or similar object is recommended for this purpose.

### Select Code Switches

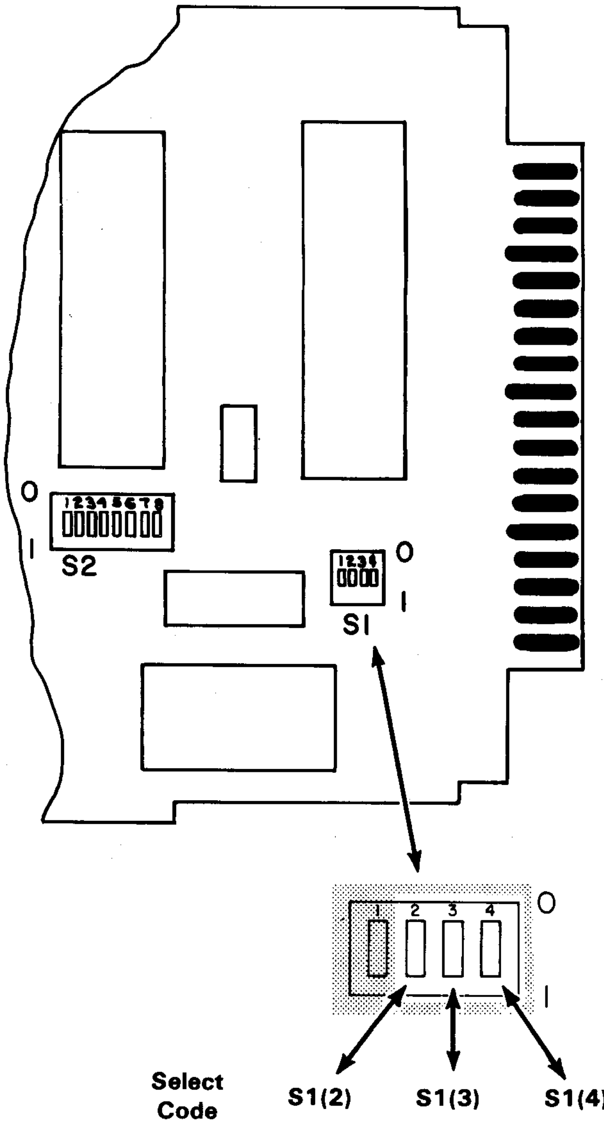
Switch segments 2, 3 and 4 of switch S1 are preset at the factory for select code "4" as follows:

- Switch segment 2 set to "0".
- Switch segment 3 set to "0".
- Switch segment 4 set to "1".

The "0" and "1" positions are labeled on the circuit board.

Select codes 3 through 10 may be set with these three switch segments. To change or verify the factory setting, orient the circuit board as shown in Figure 2-3 and locate switch segments 2, 3 and 4. Next, identify the "0" and "1" switch positions on the circuit board. You may verify that select code "4" is properly set by comparing the actual positions of switch segments 2, 3 and 4 with those illustrated. They should be the same.

To change the select code, refer to the table and set switch segments 2, 3 and 4 as required for the select code chosen. For example, if select code "3" is to be set, the three switch segments must all be set to "0". In this case, if the switches are the slide type, slides 2, 3 and 4 must all be set to the "0" position; if they are the rocker type, all three rockers must be pressed down toward the "0" position.



The select code is preset to "4" at the factory and is shown below.

To change the select code, locate the desired code in the table and set the corresponding switch segments.

Set the rocker switch segment by depressing the "0" or "1" side with a pointed object. Depressing the "0" side selects "0" for that segment.

Set the slide switch segment by moving the slide tab toward the "0" or "1" side with a pointed object. Sliding the tab to the "0" side selects "0" for that segment.

| | Select Code | S1(2) | S1(3) | S1(4) |
|---|---|---|---|---|
| | 3 | 0 | 0 | 0 |
| PRESET → | 4 | 0 | 0 | 1 |
| | 5 | 0 | 1 | 0 |
| | 6 | 0 | 1 | 1 |
| | 7 | 1 | 0 | 0 |
| | 8 | 1 | 0 | 1 |
| | 9 | 1 | 1 | 0 |
| | 10 | 1 | 1 | 1 |

0 = On/Closed    1 = Off/Open

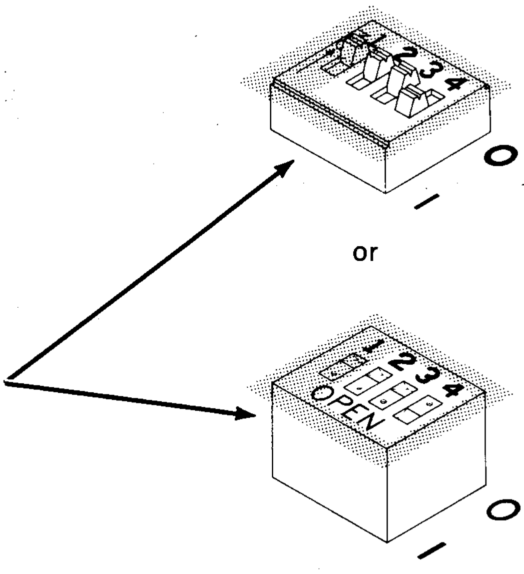


Figure 2-3. Select Code Switch Settings

## Device Address Default Switches

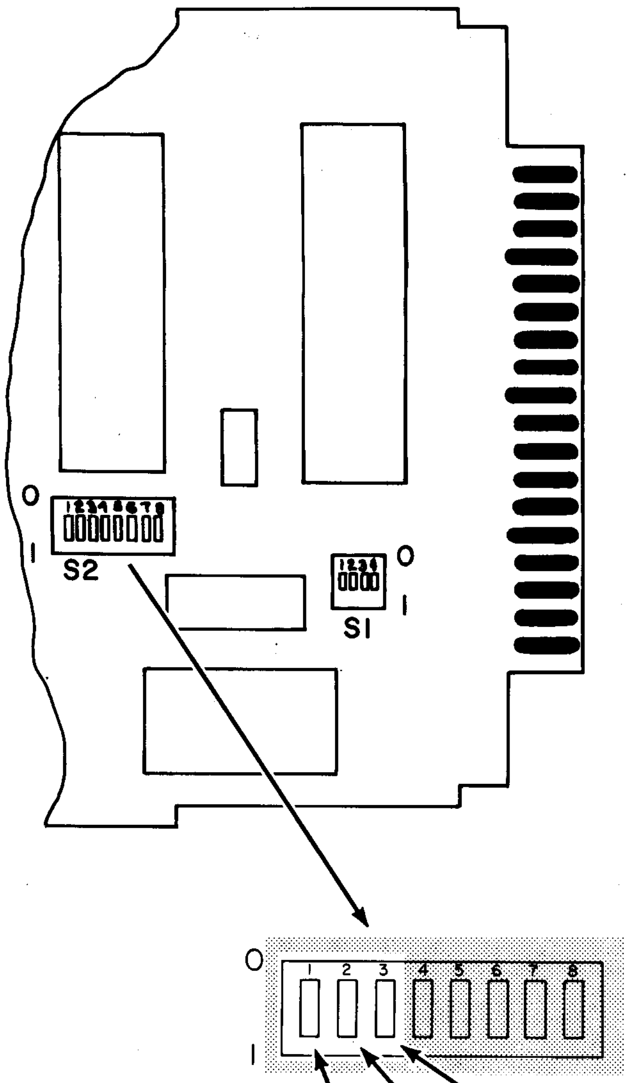
Switch segments 1 through 3 of switch S2 are preset at the factory for device address "06" as follows:

- Switch segment 1 set to "1",
- Switch segment 2 set to "1",
- Switch segment 3 set to "0".

The "0" and "1" positions are labeled on the circuit board.

Any device address from 0 through 7 may be set with these three switch segments. To change or verify the factory setting, orient the circuit board as shown in figure 2-4 and locate switch segments 1 through 3. Next, identify the "0" and "1" switch positions on the circuit board. You may verify that device address "06" is properly set by comparing the actual positions of switch segments 1 through 3 with those illustrated. They should be the same.

To change the factory setting, refer to the table and set switch segments 1 through 3 as required for the device address chosen. For example, if device address "7" is chosen, switch segments 1, 2 and 3 must be set to "1". In this case, if the interface is equipped with slide switches, move the slides for segments 1, 2 and 3 to the "1" position. If they are rocker switches, press the rockers down toward the "1" position for segments 1, 2 and 3.



The device address switches are preset to "06" at the factory and are shown below.

To change the device address, locate the desired address in the table and set the corresponding switch segments.

Set the rocker switch segment by depressing the "0" or "1" side with a pointed object. Depressing the "00" side selects "0" for that segment.

Set the slide switch segment by moving the slide tab toward the "0" or "1" side with a pointed object. Sliding the tab to the "0" side selects "0" for that segment.

|  | Address | S1(1) | S1(2) | S1(3) |
|---|---|---|---|---|
|  | 00 | 0 | 0 | 0 |
|  | 01 | 0 | 0 | 1 |
|  | 02 | 0 | 1 | 0 |
|  | 03 | 0 | 1 | 1 |
|  | 04 | 1 | 0 | 0 |
|  | 05 | 1 | 0 | 1 |
| PRESET→ | 06 | 1 | 1 | 0 |
|  | 07 | 1 | 1 | 1 |

0 = On/Closed 1 = Off/Open

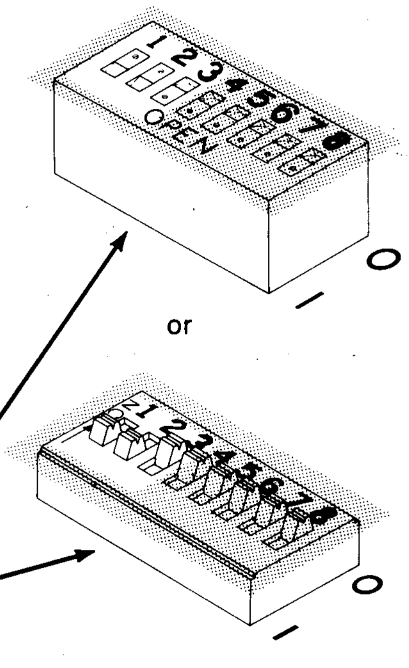


Figure 2-4. Device Address Default Switch Settings

## Default Switch Settings

The configuration switch selections determine the configuration (default condition) at power-on and reset. Refer to table 2-1 for available configurations. Refer to figure 2-5 for setting the following switch segments:

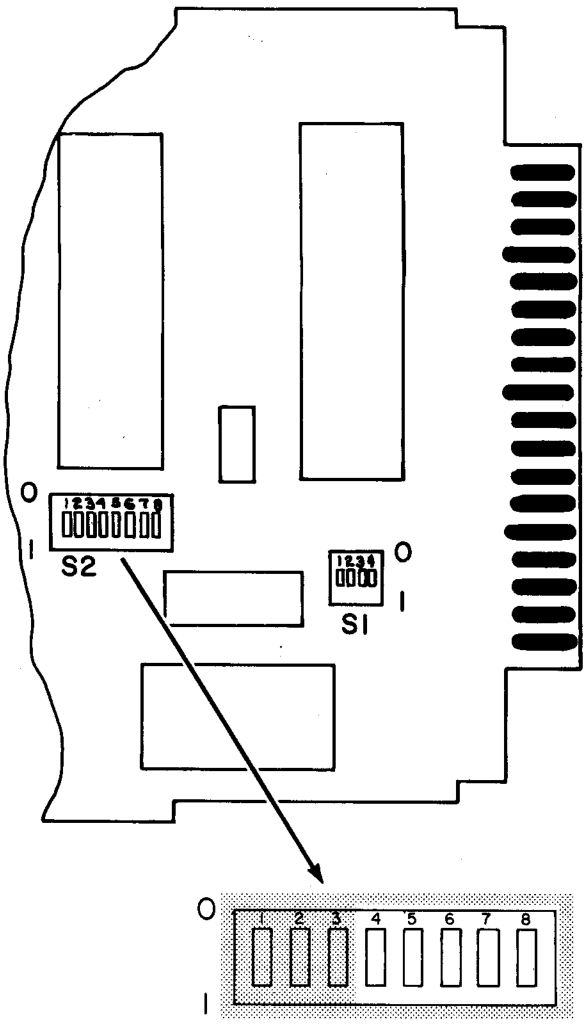
**Table 2-1. Default Switch (S2) Selections**

| Switch Segment | Function |
|---|---|
| S2(1), S2(2), S2(3) | Determines data port configuration (refer to table 2-2 and figure 2-4). Device addresses 00 through 07 only may be configured by the switches. Addresses 0 through 15 may be selected by writing to Register 5. |
| S2(4) | Protects the output drivers. Factory setting disables output ports (0 = not enabled, 1 = enabled). When this switch is set to 1, Register 8 (output enable register) can be written to enable output capability on ports A and B. |
| S2(5) | Determines handshake mode; 1 = partial, 0 = full. |
| S2(6) | Determines data line logic sense; 0 = positive-true logic, 1 = negative-true logic. |
| S2(7) | Determines FLG- and ST- logic sense; 0 = Busy-High, Ready-Low; 1 = Busy-Low, Ready-High. |
| S2(8) | Determines CTL- logic sense; 0 = Set-High, Clear-Low; 1 = Set-Low, Clear-High. |

With the exception of S2(4) these switches can be over-ridden by programming. The data port configuration is programmed by modifying Register 5. The handshake mode is programmed via Register 4 and the logic sense modes are controlled through the contents of Registers 3 and 4.

The Output Enable switch S2(4) is factory preset to the "0" (not enabled) position to protect the interface until the cable is cut and properly prepared. Once the cable(s) have been finished this switch should be moved to the 1 (enabled) position so that the interface may output data. If there is any question concerning proper cable preparation, leave this switch at "0" and attempt input operations to confirm correct switch/cable configuration.

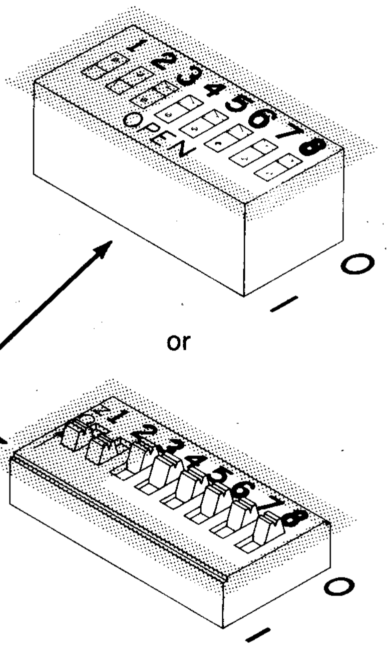
**Note:** Device address (port configuration) may be 00 through 15 by modifying the contents of register 5. Only device addresses 00 through 07 may be selected by setting segments 1, 2 and 3 of switch S2 (default condition).



To change the default condition, locate the desired condition in the table and set the corresponding switch segments.

Set the rocker switch segment by depressing the "0" or "1" side of the switch with a pointed object. Depressing the "0" side selects "0" for that segment.

Set the slide switch segment by moving the slide tab with a pointed object. Sliding the tab to the "0" selects "0" for that segment.



| | S2(4) | S2(5) | S2(6) | S2(7) | S2(8) |
|---|---|---|---|---|---|
| PRESET→ | 0 | 0 | 0 | 0 | 0 |

| S2 Segment | Function | "1" Setting | "0" Setting |
|---|---|---|---|
| 4 | Output Enable Register | Enabled | Disabled |
| 5 | Handshake Mode | Partial | Full |
| 6 | Data Line Logic Sense | Negative True | Positive True |
| 7 | Flag/Status Logic Sense | Busy = Low, Ready = High | Busy = High, Ready = Low |
| 8 | CTL — Logic Sense | Set = Low, Clear = High | Set = High, Clear = Low |

**Figure 2-5. Default Switch Settings**

**Table 2-2. Port Configurations**

| Device Address | Selected Data Port | Data Transfer Configuration | Handshake Lines | # of Bits | Transfer Direction Control Line |
|---|---|---|---|---|---|
| 00 | Port A | Bi-directional port | CTLA/FLGA | 8 | OUTA line shows direction |
| 01 | Port B | Bi-directional port | CTLB/FLGB | 8 | OUTB line shows direction |
| 02 | In port A Out port C | 2 uni-directional ports | CTLA/FLGA | 8/8 | OUTA line shows direction |
| 03 | In port B Out port D | 2 uni-directional ports | CTLB/FLGB | 8/8 | OUTB line shows direction |
| 04 | Port C | Output only port | CTL0/ST0 | 8 | Does not use OUT-line |
| 05 | Port D | Output only port | CTL1/ST1 | 8 | Does not use OUT-line |
| 06 | Ports A/C | Bi-directional ports A & C tied together | CTLA/FLGA | 8 | OUTA line shows direction |
| 07 | Ports B/D | Bi-directional ports B & D tied together | CTLB/FLGB | 8 | OUTB line shows direction |
| 08 | Ports A & B | Bi-directional ports (port A LSBs, port B MSBs) | CTLA/FLGA | 16 | OUTA line shows direction |
| 09 | Ports A & B | Bi-directional ports (port A LSBs, port B MSBs) | CTLB/FLGB | 16 | OUTB line shows direction |
| 10 | In ports A & B Out ports C & D | 2 uni-directional ports (ports A & C LSBs, ports B & D MSBs) | CTLA/FLGA | 16/16 | OUTA line shows direction |
| 11 | In ports A & B Out ports C & D | 2 uni-directional ports (ports A & C LSBs, ports B & D MSBs) | CTLB/FLGB | 16/16 | OUTB line shows direction |
| 12 | Ports C & D | Output only ports (port C LSBs, port D MSBs) | CTL0/ST0 | 16 | Does not use OUT-line |
| 13 | Ports C & D | Output only ports (port C LSBs, port D MSBs) | CTL1/ST1 | 16 | Does not use OUT-line |
| 14 | Ports A & B/C & D | Bi-directional ports (ports A & C tied — LSBs, ports B & D tied — MSBs) | CTLA/FLGA | 16 | OUTA line shows direction |
| 15 | Ports A & B/C & D | Bi-directional ports (ports A & C tied — LSBs ports B & D tied — MSBs) | CTLB/FLGB | 16 | OUTB line shows direction |

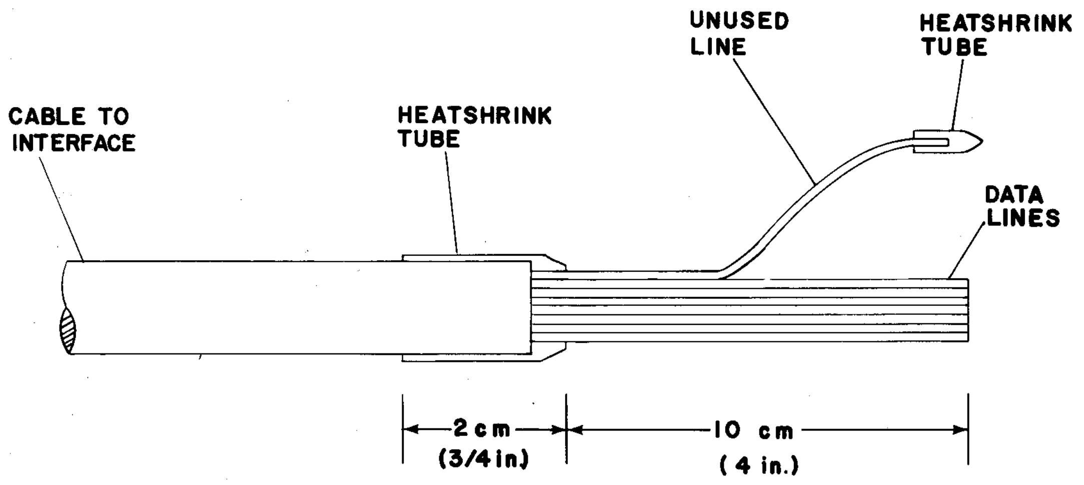
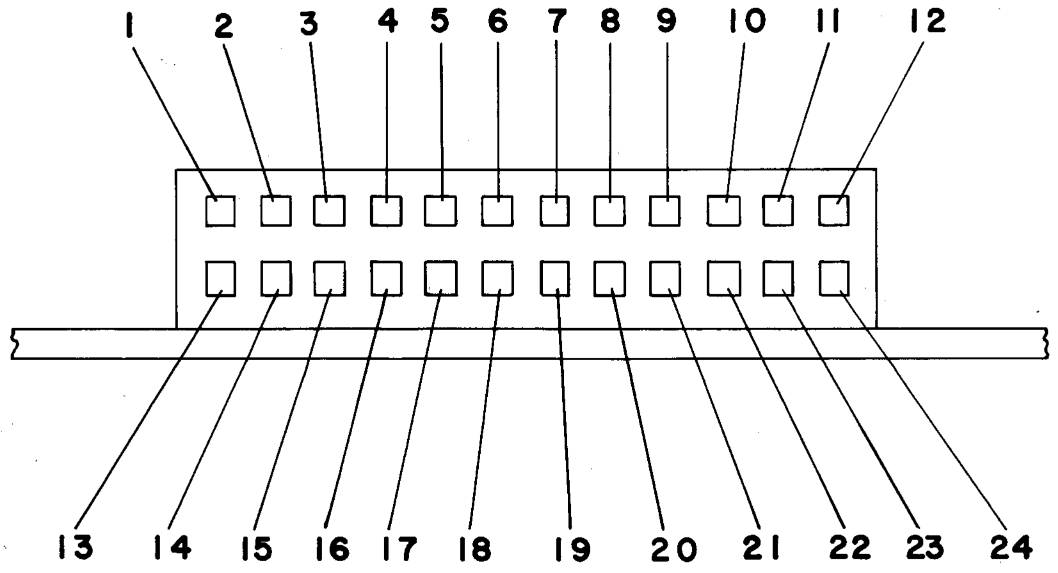


**Figure 2-6. Cable Preparation**



| J1 Pin Assignment | J2 Pin Assignment | Wire Color | |
|---|---|---|---|
| 1 GND | 1 GND | Black | (0) |
| 2 DC5 | 2 DD5 | White/Blue | (96) |
| 3 DC7 | 3 DD7 | White/Grey | (98) |
| 4 DC2 | 4 DD2 | White/Orange | (93) |
| 5 DC0 | 5 DD0 | White/Brown | (91) |
| 7 OUTA | 7 OUTB | White/Black/Red | (902) |
| 8 CTLA | 8 CTLB | White/Black | (90) |
| 9 DA7 | 9 DB7 | Grey | (8) |
| 10 DA5 | 10 DB5 | Blue | (6) |
| 11 DA0 | 11 DB0 | Brown | (1) |
| 12 DA2 | 12 DB2 | Orange | (3) |
| 13 GND | 13 GND | White | (9) |
| 14 DC4 | 14 DD4 | White/Green | (95) |
| 15 DC6 | 15 DD6 | White/Violet | (97) |
| 16 DC3 | 16 DD3 | White/Yellow | (94) |
| 17 DC1 | 17 DD1 | White/Red | (92) |
| 18 FLGA | 18 FLGB | White/Black/Yellow | (904) |
| 19 RESA | 19 RESB | White/Black/Orange | (903) |
| 20 CTL0 | 20 CTL1 | White/Black/Brown | (901) |
| 21 DA6 | 21 DB6 | Violet | (7) |
| 22 DA4 | 22 DB4 | Green | (5) |
| 23 DA1 | 23 DB1 | Red | (2) |
| 24 DA3 | 24 DB3 | Yellow | (4) |

**Figure 2-7. GPIO Interface Connector Pin-Outs**

## Recommended Peripheral Driver and Receiver Circuits

Sixteen of the data I/O lines (ports A and B) are connected to I/O expanders. These lines have bi-directional data transfer capability. The input voltage to these lines must not exceed 5 volts.

Here are typical specifications:

- $V_{in}$ max = 5.0V
- $V_{in}$ high = > 2.0V, <5.0V
- $V_{in}$ low = $\leqslant$ 0.8V
- $I_{out}$ low $\leqslant$ 4.5 mA @ 0.45V
- $I_{out}$ high $\geqslant$ 240 $\mu$V @ 2.4V

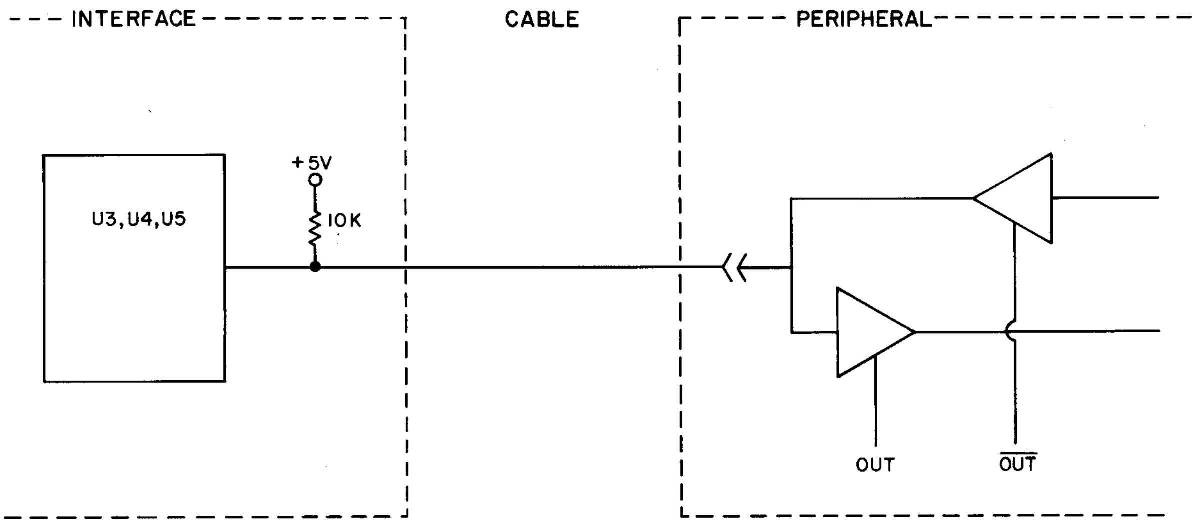


**Figure 2-8. Recommended Peripheral Transceiver Circuit (Bi-directional Data Transfer on Ports A and B)**

The port A and B data lines may also be connected for input only or output only data transfer as follows:

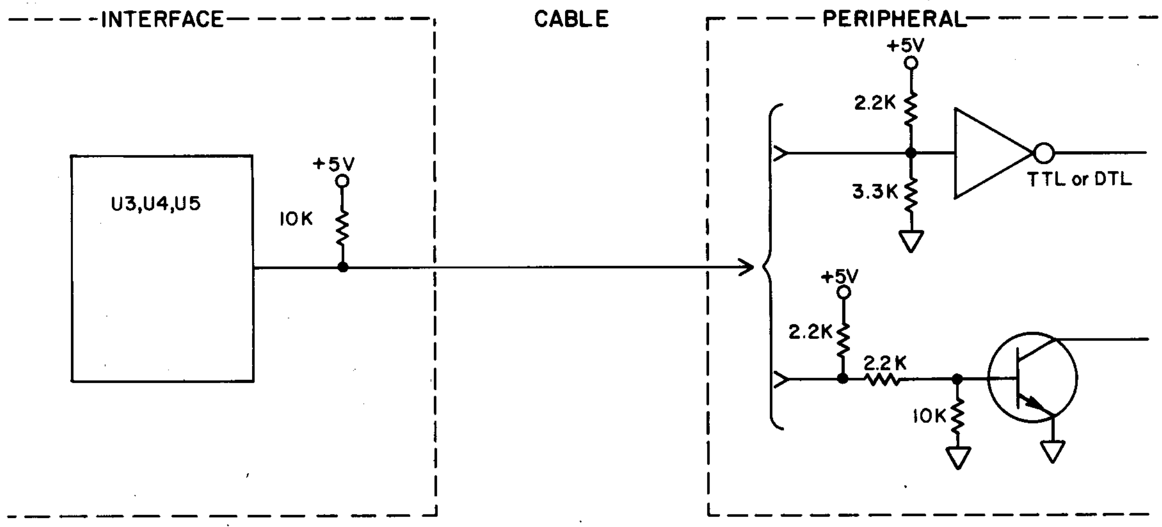


**Figure 2-9. Recommended Peripheral Driver Circuit (Ports A and B Output Only)**

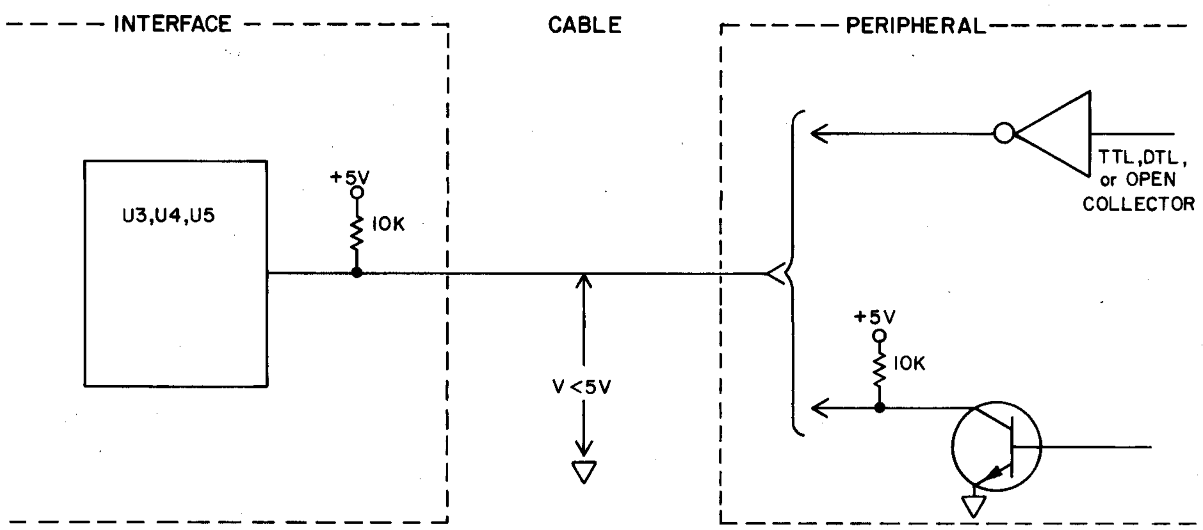


**Figure 2-10. Recommended Peripheral Reciever Circuits (Ports A and B Input Only)**

Ports C and D utilize open collector output devices as line drivers and are therefore capable of output data transfer only.

Here are typical specifications:

- $I_{out}$ low = 20 mA for each line
- $V_{out}$ low = $\leqslant 0.5V$
- $V_{out}$ high = 5V maximum

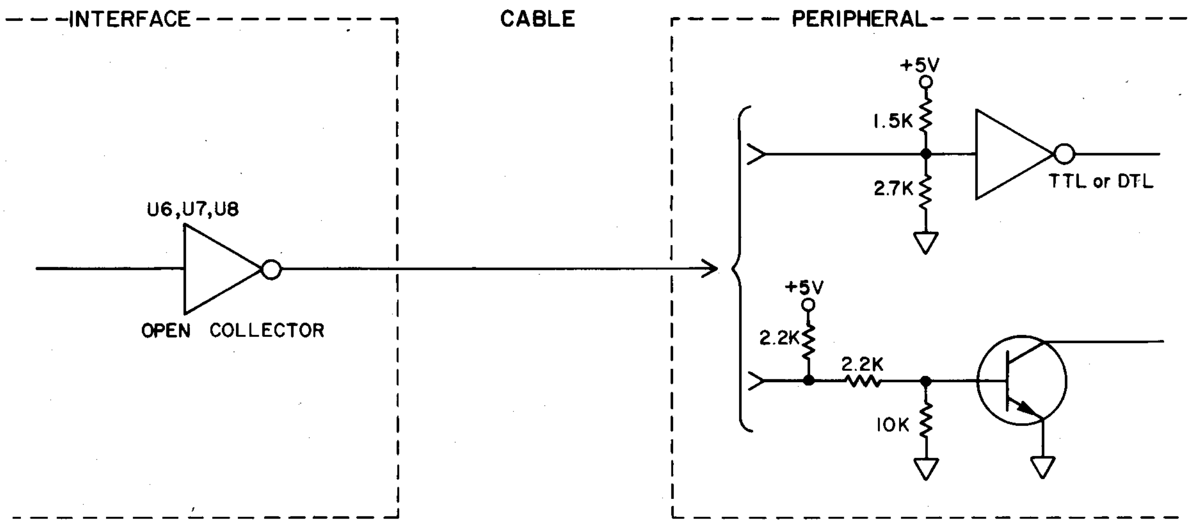


**Figure 2-11. Recommended Peripheral Receiver Circuits (Ports C and D, Lines CTL0 and CTL1, Output Only)**

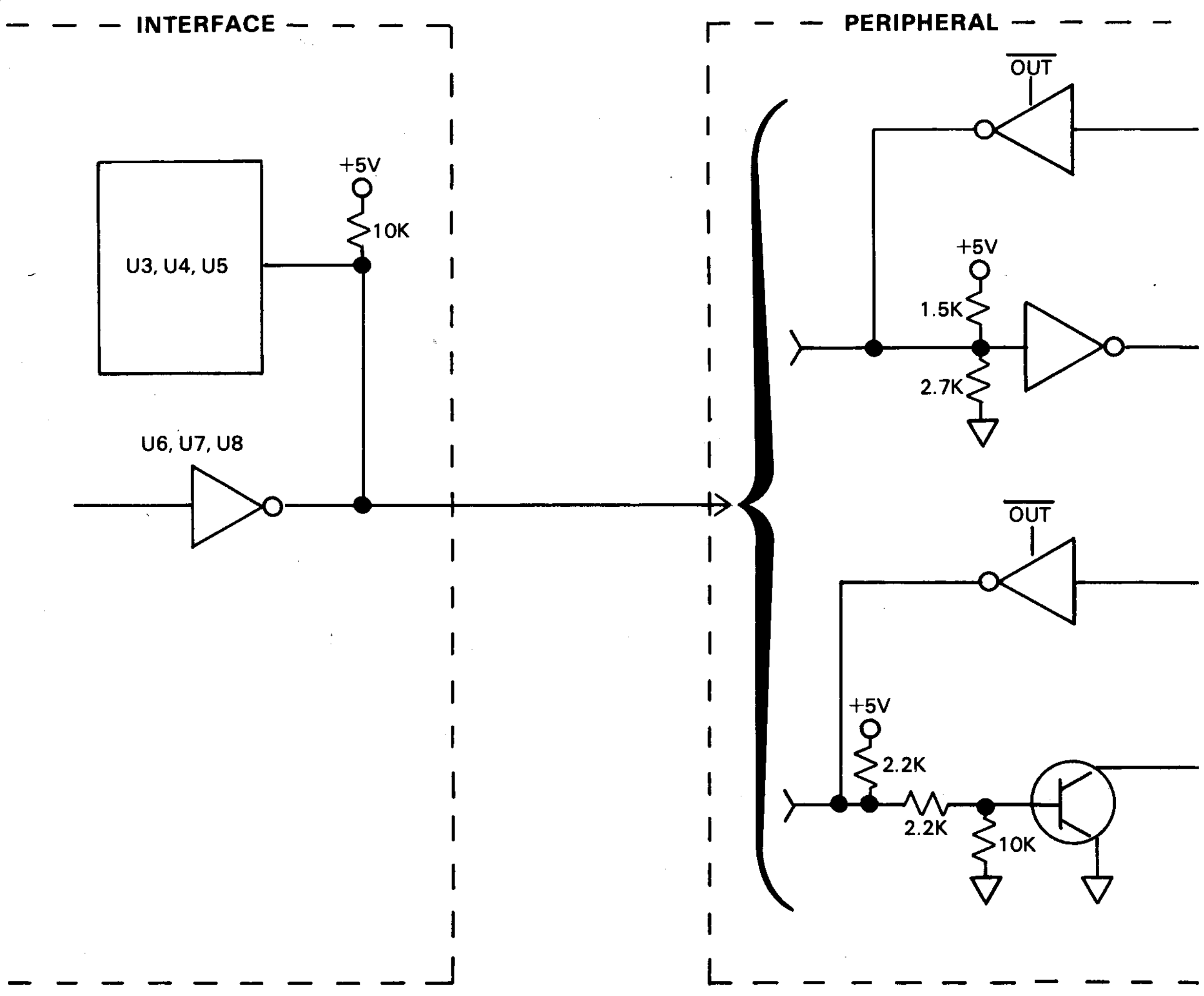


**Figure 2-12. Recommended Peripheral Transceiver Circuits (Ports A and C, Ports B and D, Wired Together)**

When using tied-together ports (device addresses 06, 07, 14 and 15) the above configurations are typical. Integrated circuits that perform both functions are available such as the HP-IB driver/receiver (75XX160). This configuration is recommended for long interface cable lengths due to the high noise immunity of low impedance drivers.

## Installing the Interface and Connecting the Peripherals

Make sure you read and understand this entire section before you install the interface or connect a peripheral device to it.

### Safety Precautions

Manufacturers of peripheral devices often use different grounding techniques. In some instances, logic ground is allowed to float with respect to earth ground in an effort to reduce ground return interference with digital signals. This may cause a voltage level between the two grounds to be high enough to be hazardous. Therefore, care should be taken when you are installing the interface or when peripherals are being connected to or disconnected from the interface.

On the HP 82940A GPIO Interface, the ground contact connects to the cable shield, and, when the interface is installed in the HP Series 80 Personal Computer, the ground contact connects to earth ground and to logic ground. On the peripheral end of the cable, the shield is not terminated to any ground. By not connecting the cable shield on the peripheral end, two things are accomplished:

- The possiblity of a ground loop problem is greatly reduced.
- If the peripheral's ground is floating or defective, touching the ground contact when the interface is removed from an I/O port cannot result in a shock hazard.

You should keep in mind, however, if the interface is removed from an I/O port while it is connected to a peripheral, logic ground does appear on the edge connector of the interface. Unless you know the voltage level of logic ground with reference to earth ground, never touch the edge connector while the interface is terminated to a peripheral.

When the HP 82940A GPIO Interface is installed in the computer, earth ground and logic ground become connected together. Thus, if logic ground on a peripheral is never connected to earth ground or, if it is defective, it may have a voltage level considerably different than logic ground on the interface. This level may be high enough to be hazardous unless peripherals are connected to the bus in an exacting manner.

If you don't know the grounding technique used on a peripheral, check with the manufacturer of the device. After verifying that suitable grounding techniques have been used in your peripheral, use the following steps in the order given to install the interface and peripherals to the computer's bus.

**WARNING**

To avoid personal injury and equipment damage, read and understand the preceding safety precautions and do not deviate from the order of the following steps to install the interface and peripheral.

1. Turn the power switch, located on the back of the HP Series 80 Personal Computer, to the off position. However, make sure the power cord is plugged into a grounded (3-wire) ac outlet.
2. Refer to figure 2-3 and install the interface into one of the I/O ports.
3. Make sure the power switch on the peripheral to be connected to the HP 82940A GPIO bus is in the off position.
4. Connect the interface cable to the peripheral.
5. After the peripheral has been connected to the HP 82940A GPIO bus, turn the computer and the peripheral power switches on.

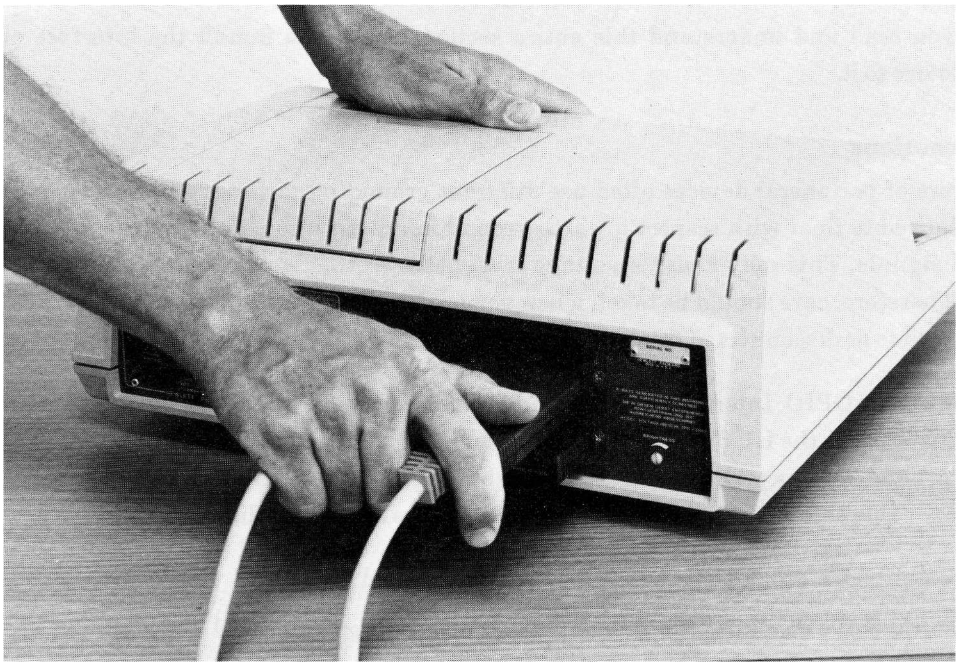


**Figure 2-13. Installing the GPIO Interface Installation**

## Removing Peripherals/Disconnecting the Interface

Use the following steps in the order given to remove peripherals from the bus or to disconnect the interface from the HP Series 80 Personal Computer.

> **CAUTION**
>
> Do not remove the interface from the computer with the power switch on. Doing this may cause damage to either the interface, computer or both.

1. Turn the power switch on the peripheral connected to the HP 82940A GPIO bus to the OFF position.
2. Disconnect the bus cable from the peripheral. If you intend to disconnect the interface from the computer first make sure that the bus device power switch is in the off position. Then remove the interface cable from the peripheral and proceed with step 3.
3. Turn the power switch on the back of the computer to the OFF position. Make sure the interface cable is not connected to the peripheral and proceed with step 4.
4. Remove the interface from the I/O port.

Section 3

# Using Your GPIO Interface

## Introduction

The HP 82940A GPIO Interface allows your HP Series 80 Personal Computer to communicate with a wide variety of devices through the use of parallel data exchanges. A **parallel** interface sends or receives an entire byte or word of data in one operation. This is the most basic, and most versatile, method of I/O. However, it is the inherent versatility of this interface that makes it appear somewhat confusing at first. Don't be overwhelmed. Consider each interface characteristic of your peripheral device and deal with these characteristics one at a time. For example, there are 16 primary addresses to choose from on this interface. But if you know that your only requirement is the input of 8-bit data, you can eliminate 14 of the 16 choices. By using this "process of elimination" approach, you can master a parallel interfacing task in short order.

This section explains the use of the GPIO interface from a programming point of view. The emphasis is on accessing the capabilities of the interface using program statements. Unlike the HP-IB interface however, a basic parallel interface does not isloate you from the hardware. Many references to the characteristics of the hardware are necessary to properly explain the various features available to you. If your background is solely in software, you will probably want to solicit the help of a person with some hardware background. In fact, some technical facility is practically a necessity during the installation of a parallel interface because there aren't any connectors wired to the HP 82940A GPIO Interface when you receive it.

Throughout this section, the abbreviation "GPIO" is used when referring to the HP 82940A parallel interface. This stands for "General Purpose Input and Output" and reflects the flexible nature of the interface.

## Essentials of a Parallel Interface

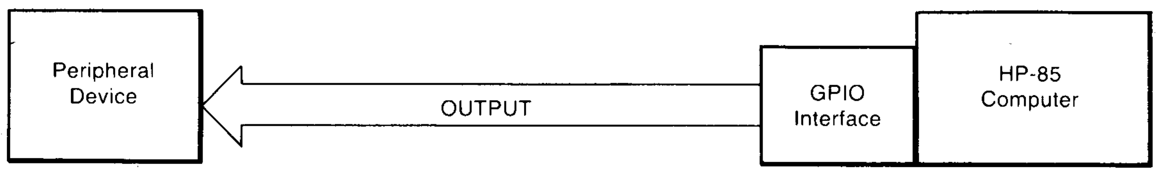
The introduction recommended that you consider each interface characteristic individually whenever possible. What are these essential characteristics? In most cases, a parallel interface will be successful if each of the following characteristics has been properly determined:

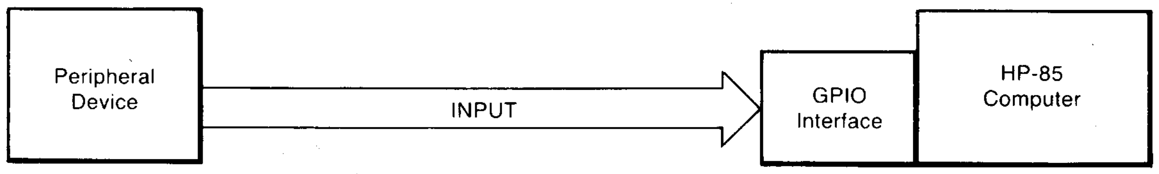
- Direction of data flow
- Number of bits in a unit of data
- Method and timing of handshake
- Logical polarity of data and control lines
- Type of I/O statement used in the program

Note that these five categories represent only the essentials of a parallel interface. There may be other factors to consider in individual applications, such as parity, end-of-line sequence, and creative use of interrupts. But no amount of attention to parity or end-of-line sequence will get an interface working if the handshake or polarity is wrong! Therefore, deal first with the five factors listed above. Extra features and special capabilities can be added after the GPIO is properly handling the basic communication task.

## Direction of Data Flow

Because an interface connects to both the computer and the peripheral device, it is important to avoid confusion about the direction of data flow. The output of the computer is the input to the peripheral device. All references to data direction in this section are given with respect to the computer. This is shown in the following diagrams.





There are four basic choices when selecting data direction with the GPIO interface. They involve direction of data flow and drive capability. Two kinds of output ports are available. One kind has a small drive capability of about two standard TTL loads. The other kind has a larger drive capability of about 12 standard TTL loads. A list of the data direction choices available is shown below. A detailed description of each choice is given in the following paragraphs. Typical drivers and receivers for use with these options are shown in the Installation section, page 00.

1. Bidirectional—small output drive (choose this for **input-only** applications)
2. Bidirectional—large output drive
3. Input and output on separate lines—large output drive
4. Output only—large output drive.

Choice #1 is a bidirectional port with a small output drive capability. This type of port is recommended for input-only operations and for bidirectional interface to light loads. A "light" load is a circuit that sources less than 4.5 mA. Some examples are NMOS interface chips, one TTL gate with a 2.2 kΩ pull-up resistor, or CMOS gates with a 10 kΩ pull-up resistor.

Choice #2 is a bidirectional port with increased output drive capability. This type of port is recommended for bidirectional interface to heavier loads. The output drivers on this port type are open-collector transistors rated to sink 20 mA. Any bidirectional load that sources more than 4.5 mA must be connected to this port type.

Choice #3 is similar to choice #2, but there is a significant difference. The bidirectional port (choice #2) uses a common data bus for input and output. The port type of choice #3 uses one data bus for input and a separate data bus for output. This type of port is useful when interfacing to a device that has separate input and output lines which cannot be connected together for electrical reasons.

Choice #4 is for output-only applications. This port type uses open-collector drivers rated to sink up to 20 mA.

## Number of Bits and Ports

The GPIO interface allows the selection of either 8-bit or 16-bit ports. The number of ports available depends upon the size you choose and the data direction requirements. If you are using 8-bit ports, there can be a maximum of four independent ports. These are two bidirectional ports with small output drive and two output-only ports. Note that other configurations yield fewer ports. For example, if you need bidirectional ports with large output drive, there can only be two. The reason for this will become apparent as you read the next topic on addressing and configuration.

A similar situation exists with 16-bit ports. You can have two of them if one is output only and the other is bidirectional with small output drive. However, any other configuration is limited to one 16-bit port.

## Using Primary Addresses

You select the type of port by specifying a primary address in your `OUTPUT`, `ENTER`, or `TRANSFER` statements. In essence, each type of port is treated as a separate device and is accesssed by using a device selector. There are other ways to access a port, but they are all related to primary addresses. The primary address can be written directly into register 5, and the default configuration switches allow any primary address for an 8-bit port to be selected automatically at power-on or reset. However, the simplest way to avoid surprises and confusion is to include the desired primary address in your device selector when performing I/O operations.

If you do not specify a primary address in the device selector (e.g. `OUTPUT 4;X`), the last primary address specified is used. If no primary address has been specified since the last power-on or reset, the address set by the default configuration switches is used.

The following tables summarize the port options available. The tables also indicate which lines are used for handshake and direction indicator for each port type. The handshake lines are discussed at length in Handshake Methods (covered next). The direction indicator is a line used with a bidirectional port to indicate in which direction the data is currently flowing. This line is often used for the control of tri-state gates or selector circuits. It presents a logic low when the interface is outputting and a logic high when the interface is inputting.

**Table 3-1. 8-Bit Ports**

| Data Direction | Primary Address | Port Description | Handshake Lines | Direction Indicator |
|---|---|---|---|---|
| Bidirectional: small output drive | 00 | Port A | CTLA/FLGA | OUTA |
| | 01 | Port B | CTLB/FLGB | OUTB |
| Input and output on separate lines; large output drive | 02 | Input to Port A Output from Port C | CTLA/FLGA | OUTA |
| | 03 | Input to Port B Output from Port D | CTLB/FLGB | OUTB |
| Output only; large output drive | 04 | Port C | CTL0/ST0 | none |
| | 05 | Port D | CTL1/ST1 | none |
| Bidirectional; large output drive | 06 | Port A and Port C wired together | CTLA/FLGA | OUTA |
| | 07 | Port B and Port D wired together | CTLB/FLGB | OUTB |

Table 3-2. 16-Bit Ports

| Data Direction | Primary Address | Port Description | Handshake Lines | Direction Indicator |
|---|---|---|---|---|
| Bidirectional; small output drive | 08 | LSB* on Port A<br>MSB* on Port B | CTLA/FLGA | OUTA |
| | 09 | same | CTLB/FLGB | OUTB |
| Input and output on separate lines; large output | 10 | LSB input on Port A<br>MSB input on Port B<br>LSB output on Port C<br>MSB output on Port D | CTLA/FLGA | OUTA |
| | 11 | same | CTLB/FLGB | OUTB |
| Output only; large output drive | 12 | LSB on Port C<br>MSB on Port D | CTL0/ST0 | none |
| | 13 | same | CTL1/ST1 | none |
| Bidirectional; large output drive | 14 | Port A and Port C wired together (LSB)<br>Port B and Port D wired togehter (MSB) | CTLA/FLGA | OUTA |
| | 15 | same | CTLB/FLGB | OUTB |

## Handshake Methods

A **handshake** is a sequence of electrical events used to synchronize a transfer of data. There is a brief overview of the handshake process in the *I/O ROM Owner's Manual.* With the GPIO interface, you have four basic methods of handshake to choose from (with some variations, of course). These basic choices are:

- Full handshake
- Partial handshake
- Strobe handshake
- No handshake

The handshake lines on the interface are called **FLAG** (FLG) and **CONTROL** (CTL). The FLAG line is used to sense the handshake signal coming from the peripheral device, and the CONTROL line is used to send a handshake signal from the interface to the peripheral device. (The output-only ports use a line called **STATUS** (ST) to perform the same function as the FLAG line.) Exactly what signals are sent and received depends upon the handshake mode that you select. Let's look at the details of each method.

* As it is used here, the abbreviation "LSB" stands for "Least Significant Bits". These are bit 0 thru 7 of the 16-bit word. Likewise, "MSB" stands for "Most Significant Bits". These are bit 8 thru bit 15 of the 16-bit word.

## Output Handshakes

Output handshakes are somewhat simpler than input handshakes, so they are presented first. The following timing diagrams show only the essential action of the data and handshake lines. The line used to indicate data direction has been left out for the sake of simplicity, and not all timing relationships have been given numeric values. More complete timing information is available in Theory of Operation, appendix A. These diagrams are intended to clarify the concept of the handshake methods. The important factors to note are the order of events and the causal relationship of events.

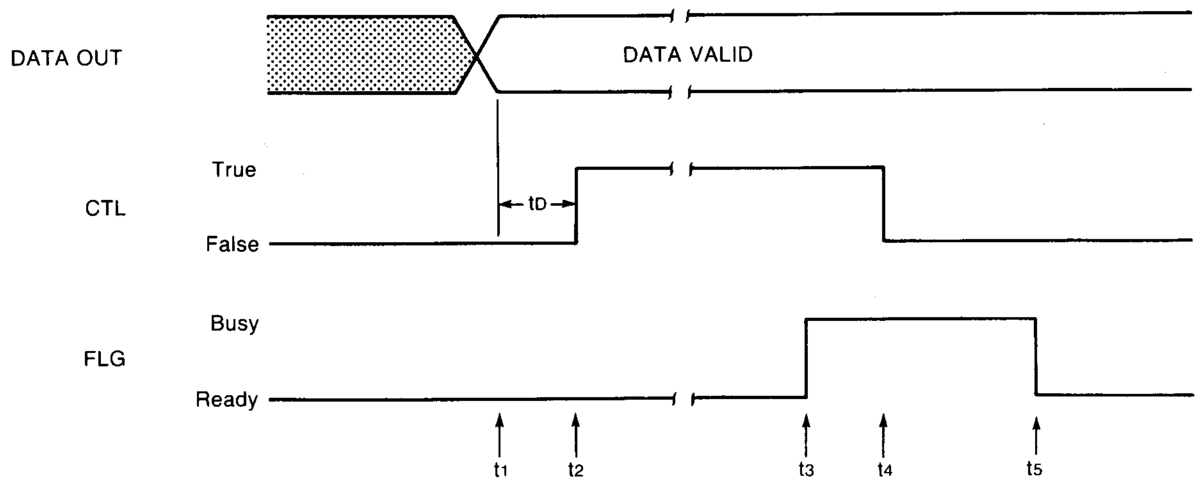


**Figure 3-1. Output: Full Handshake Timing**

When the full handshake cycle starts, the interface checks for a READY indication on the FLG line. If the line is READY, the interface places a new word of data on the DATA lines (t1). After a programmable delay time (tD), the interface places the CTL line in the true state (t2). This signals the peripheral device that the data is valid. The delay time is provided to ensure that the DATA lines are stable and valid before CTL is asserted. This delay time is set by Register 6, which is explained later. When the peripheral device sees the true state of the CTL line, it does whatever is necessary to input the data presented to it. The peripheral device indicates that it is busy inputting data by placing FLG in the BUSY state (t3). This serves as an acknowledgment to the interface that he CTL signal was received. Therefore, when the interface sees FLG go BUSY, it can return the CTL signal to the false state (t4). When the peripheral device has finished inputting data, it returns the FLG line to the READY state to indicate that it is ready for the next cycle (t5).

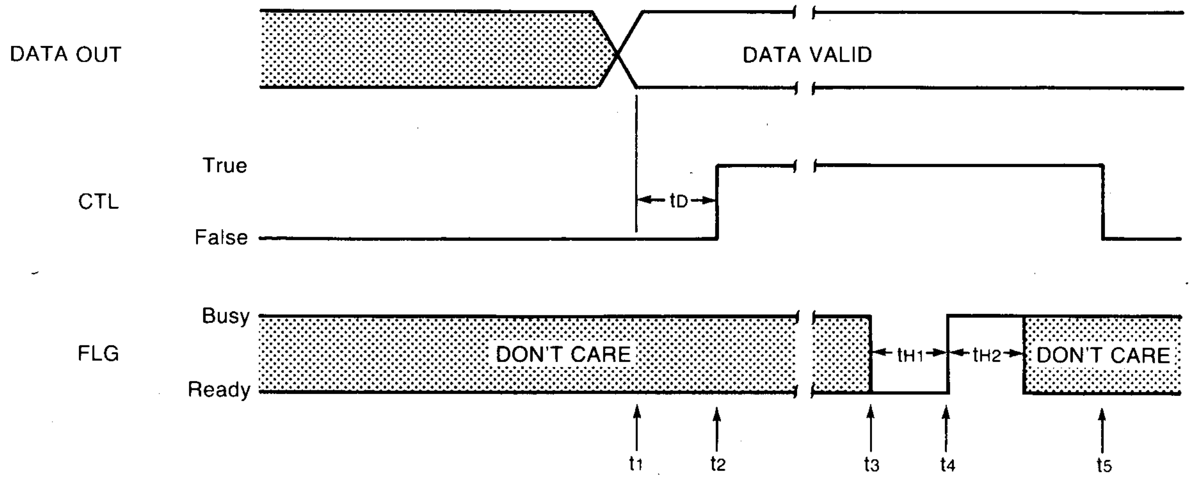


**Figure 3-2. Output: Partial Handshake Timing**

The key difference between full and partial handshake is that partial handshake does not check the FLG line before it outputs the data. The output cycle can begin with the FLG line BUSY or READY. As in the full handshake, the interface outputs the data (t1) and sets CTL to the true state (t2) after a programmable delay (tD). This signals the peripheral device that the data is valid. The interface then waits for the peripheral device to indicate that it has received the data. This is the reason for the name **partial handshake**. The interface does not require a READY signal to start the transfer, but it does require an acknowledgment to complete it. The peripheral device inputs the data and supplies the **data accepted** signal by holding FLG in the READY state (t3) for at least 30 μs (tH1) and then in the BUSY state (t4) for at least 35 μs (tH2). Note that although this action greatly resembles an edge-triggered event, it really is not. The minimum state times mentioned are necessary for the interface to sense the READY to BUSY transition. Once the interface senses the FLG signal from the peripheral device, it can return the CTL signal to the false state (t5).

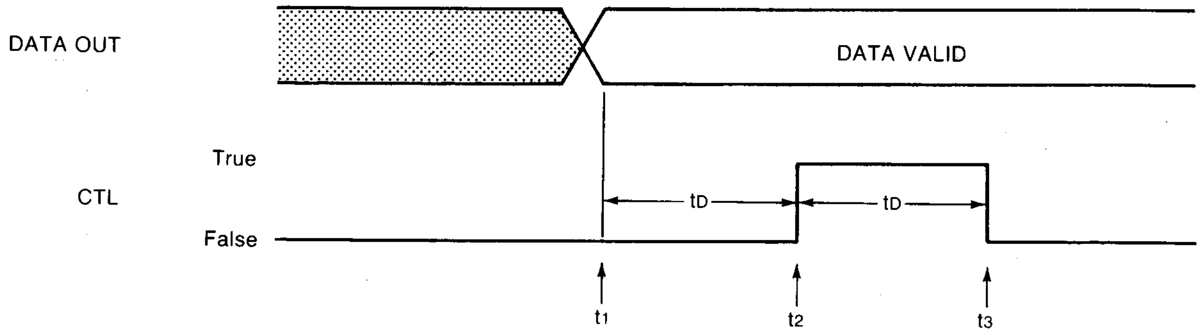


**Figure 3-3. Output: Strobe Handshake Timing**

The strobe handshake for output is a very simple sequence. It is probably the most common handshake method used in devices that do not implement full handshake. This method assumes that the peripheral device is always ready and the FLG line is not used. (If your device is not always ready, then the "Output Inhibit" feature can be used. This is explained in Selecting the Handshake Method.) The cycle starts with the output of data (t1). After a programmable delay (tD), the interface sets CTL to the true state (t2). This state is held for the delay time, then CTL is returned to the false state (t3). In other words, the interface supplies data, followed by a strobe pulse to indicate that the data is valid.

## No Handshake

The "no handshake" option does not need a timing diagram. The interface simply places new data on the DATA lines when it becomes available. The FLG and CTL lines are not used. The ASSERT and STATUS statements can be used to supply your own handshake in this mode (see Direct Use of Control Lines).

## Input Handshakes

One reason that the input handshakes are more complex than the output handshakes is that each input handshake has two options for the timing of the interface's read operation. These options are called "READY to BUSY" and "BUSY to READY".* In the following diagrams, both options are shown on the same drawing. The upper part of each diagram shows the timing that is common to both options and the READY to BUSY timing. The lower part of each diagram shows the timing changes that pertain to the BUSY to READY option.

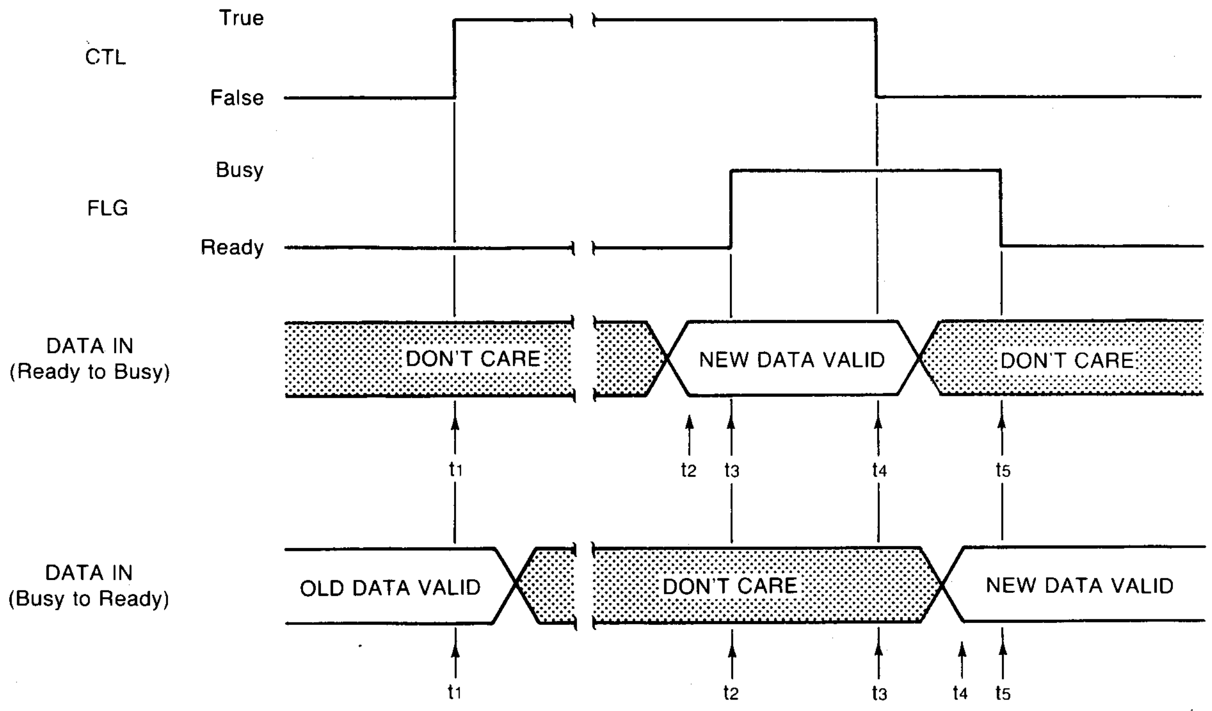


**Figure 3-4. Input: Full Handshake Timing**

**READY to BUSY:** When the full handshake cycle starts, the computer checks for a READY indication on the FLG line. If the FLG line is READY, the interface requests data by setting CTL to the true state (t1). The peripheral device sees this request and places data on the DATA lines (t2). The peripheral device then signals that the data is valid by placing the FLG line in the BUSY state (t3). When the interface sees this signal, it inputs the data (sometime between t3 and t4). The interface then signals that it has received the data by returning CTL to the false state (t4). When the peripheral device sees that the data has been received, it returns FLG to the READY state to prepare for the next cycle (t5).

* These names were derived from the state change on the FLG line that triggers the read operation in full handshake mode. However, the terms are not meant to imply that FLG lines are edge-triggered. They are not. Also, these names are used to describe the timing choices for all input handshakes, even though strobe handshake does not use a FLG line.

**BUSY to READY:** When the full handshake cycle starts, the computer checks for a READY indication on the FLG line. If the FLG line is READY, the interface requests data by setting CTL to the true state (t1). This signal tells the peripheral device that it can place new data on the DATA lines. The peripheral acknowledges the CTL signal by placing FLG in the BUSY state (t2). The interface then acknowledges the FLG signal by returning CTL to the false state (t3). After all these acknowledgments are taken care of, the peripheral device places new data on the DATA lines (t4). The peripheral device then signals that the data is valid by returning FLG to the READY state (t5). After the interface sees this signal, it inputs the data (sometime between t5 of this cycle and t1 of the next cycle.)

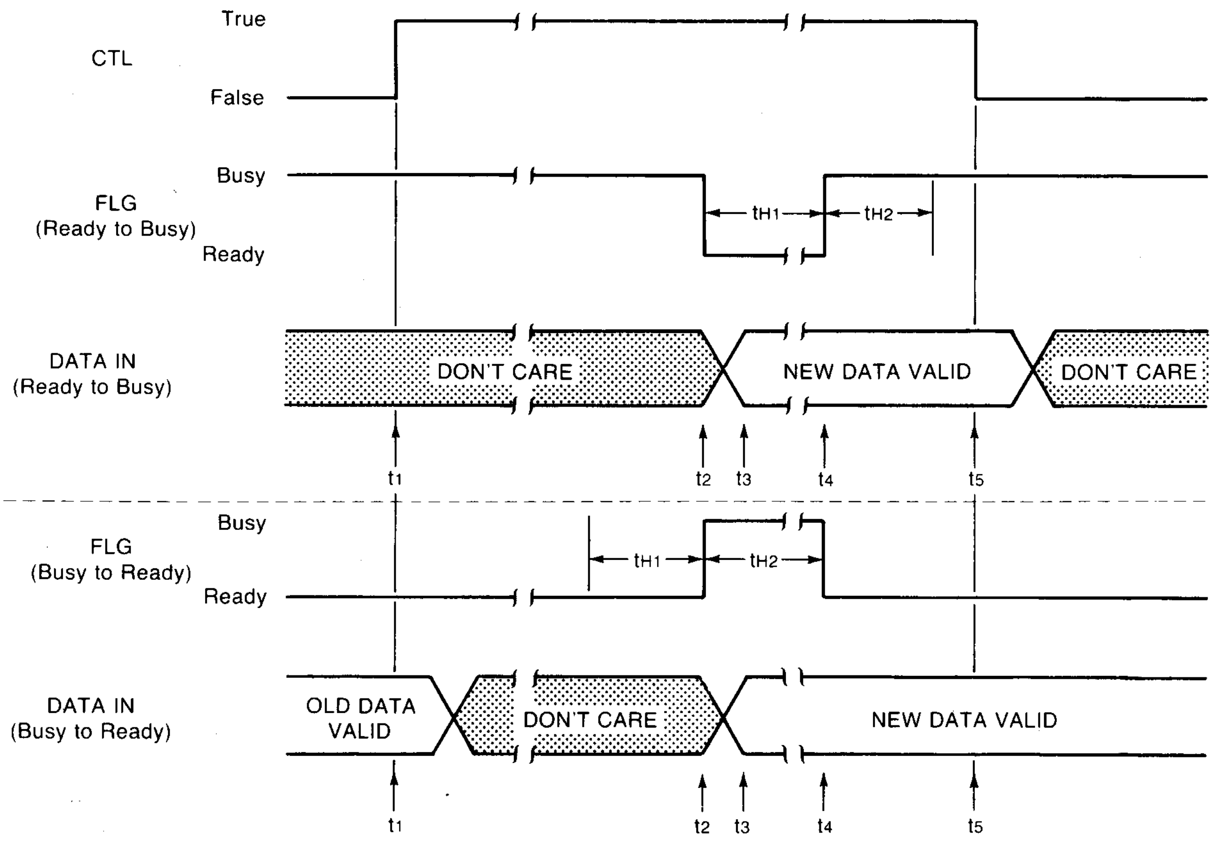


**Figure 3-5. Input: Partial Handshake Timing**

**READY to BUSY:** The primary use of this handshake method is to input data that is being sent with a strobe handshake from the peripheral device. Partial handshake does not wait for the FLG line to be READY before starting the cycle. Regardless of the state of the FLG line, the request for data is made by setting CTL to the TRUE state (t1). It does not matter if the peripheral device outputs the data first or starts the strobe pulse first. The important thing is that the data should be valid before the end of the strobe pulse. This diagram shows the strobe pulse starting first as the peripheral device places the FLG line in the READY state (t2). The data becomes valid before the end of the pulse (t3). Then the peripheral signals that the data is valid by placing the FLG line in the BUSY state (t4). The minimum state times of 30 μs (tH1) and 35 μs (tH2) are necessary for the interface to detect this READY to BUSY transition. When the interface sees this transition, it inputs the data (sometime between t4 and t5). Note that the difference between this option and the "BUSY to READY" option is timing of the input operation with respect to the end of the CTL pulse, not the polarity of the FLG pulse. Either option can be used with any polarity of FLG pulse (see Setting the Logic Polarity).

**BUSY to READY:** This is a variation of the previous method. Regardless of the state of the FLG line, the request for data is made by setting CTL to the true state (t1). It does not matter if the peripheral device outputs the data first or starts the strobe pulse first. The important thing is that the data should be valid before the end of the strobe pulse. This diagram shows the strobe pulse starting first as the peripheral device places the FLG line in the BUSY state (t2). The data becomes valid before the end of the pulse (t3). Then the peripheral signals that the data is valid by placing the FLG line in the READY state (t4). The minimum state times of 30 μs (tH1) and 35 μs (tH2) are necessary for the interface to detect the READY to BUSY transition.

After the pulse on the FLG line is finished, the interface returns CTL to the false state (t5). The interface then inputs the data (sometime between t5 of this cycle and t1 of the next cycle). Note that the difference between this option and the "READY to BUSY" option is timing of the input operation with respect to the end of the CTL pulse, not the polarity of the FLG pulse. Either option can be used with any polarity of FLG pulse (see Setting the Logic Polarity).

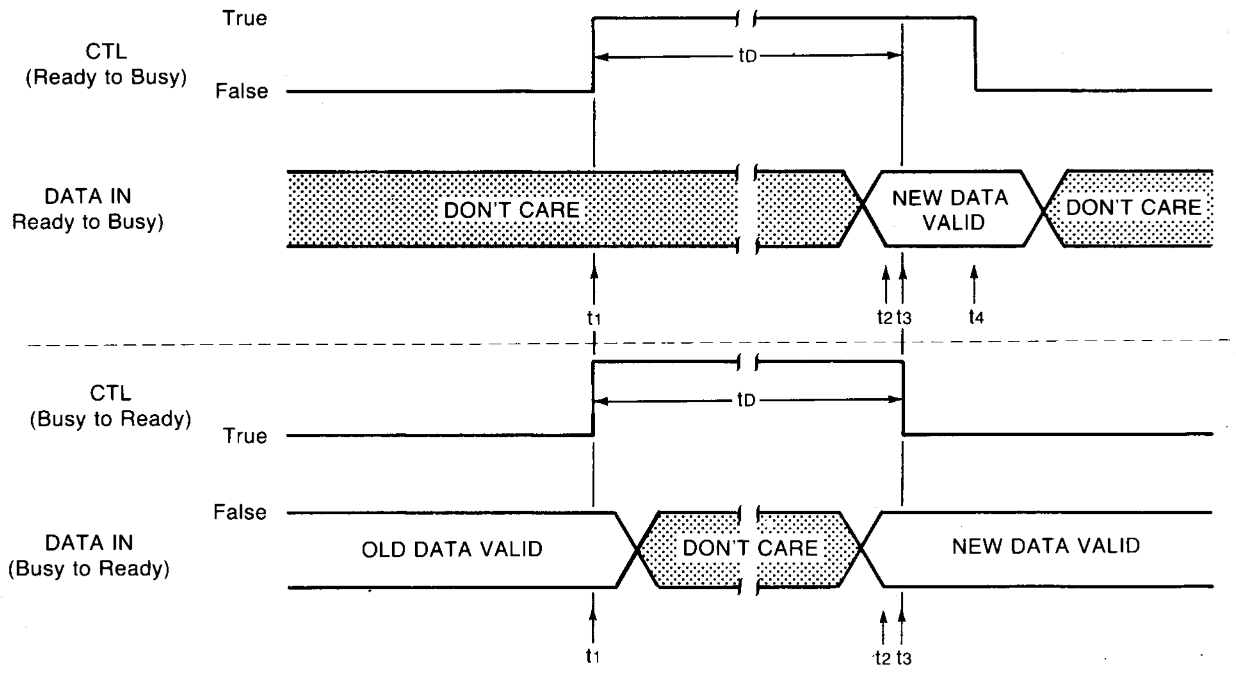


**Figure 3-6. Input: Strobe Handshake Timing**

**READY to BUSY:** This is a simple handshake method that can be used when you are sure that your peripheral device can provide valid data in a fixed amount of time after a request signal. The peripheral device is not given an opportunity to acknowledge any signals or control the handshake timing in any way. Therefore, the FLG line is not used. The cycle starts when the interface requests data by setting CTL to the TRUE state (t1). The peripheral device then places new data on the DATA lines (t2). After a programmable delay (tD), the interface inputs the data (sometime between t3 and t4). The interface completes the cycle by returning CTL to the false state (t4).

**BUSY to READY:** This is a variation of the previous method. The cycle starts when the interface requests data by setting CTL to the true state (t1). The peripheral device then places new data on the DATA lines (t2). After a programmable delay (tD), the interface returns CTL to the false state (t3). Then the interface inputs the data (sometime between t3 of this cycle and t1 of the next cycle).

### No Handshake

The "no handshake" option does not need a timing diagram. The interface simply inputs new data whenever a data input statement is executed. The FLG and CTL lines are not used. The ASSERT and STATUS statements can be used to supply your own handshake in this mode (see Direct Use of Control Lines).

## Selecting the Handshake Method

The handshake characteristics of the GPIO interface are selected by writing various codes to interface control registers. The registers of interest are Control Registers 4, 6, and 9. These are accessed by using the CONTROL and STATUS statements.

**Register 4 - Data Normalization and Handshake Control**

Most Significant Bit | Least Significant Bit

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| Handshake Method | | 0 = Ready to Busy 1 = Busy to Ready | Not Used | Data Polarity (see "Selecting the Logic Polarity") | | | |
| Value = 128 | Value = 64 | Value = 32 | Value = 16 | Value = 8 | Value = 4 | Value = 2 | Value = 1 |

Register 4 has two primary functions. The lower four bits are used to select either positive-true or negative-true data for each 8-bit port. This is explained in Selecting the Logic Polarity. The top three bits are used to select the handshake method. The primary selection of handshake method is done with bit 6 and bit 7 as follows:

| Bit 7 | Bit 6 | Handshake |
|---|---|---|
| 0 | 0 | Full |
| 0 | 1 | Partial |
| 1 | 0 | Strobe |
| 1 | 1 | None |

Bit 5 of this register is used to select the input timing option. Its states are defined as follows:

| Bit 5 | Data Input Timing |
|---|---|
| 0 | READY to BUSY |
| 1 | BUSY to READY |

The meaning of all these options is discussed at length in Handshake Methods. The following is a summary of all the choices, listed with the decimal value of the control byte used to select each choice.

| Decimal Value of Bit 5 thru Bit 7 | Handshake Method |
|---|---|
| 0 | Output: Full Handshake |
| 64 | Output: Partial Handshake |
| 128 | Output: Strobe Handshake |
| 0 | Input: Full Handshake; READY to BUSY |
| 32 | Input: Full Handshake; BUSY to READY |
| 64 | Input: Partial Handshake; READY to BUSY |
| 96 | Input: Partial Handshake; BUSY to READY |
| 128 | Input: Strobe Handshake; READY to BUSY |
| 160 | Input: Strobe Handshake; BUSY to READY |
| 192 | Input or Output: No Handshake |

It is possible to write the values shown directly into Register 4. However, if you do that, you will also clear all the data normalization bits. This gives all data ports a positive-true logic sense. If that does not cause any problems, statements like the following can be used. These, and all other example statements in this section, assume that the interface select code is 4.

```
CONTROL 4,4 ; 128 ! Set strobe handshake
```

isc reg# control byte

```
CONTROL 4,4 ; 96 ! Partial handskake, input BUSY to READY
CONTROL 4,4 ; 192 ! Turn off handshake
```

You are encouraged to get into the habit of using comments on statements like these. The word CONTROL followed by a bunch of numbers can be very mysterious when you look at a program some months after it was written. Anyone who needs to support a program will be very thankful for a little bit of information about the action of a cryptic CONTROL statement. Remember, that support person just might be you!

Now suppose that you are concerned about affecting the normalization bits. There are two approaches to this problem. First, and most common, is to set all the options in Register 4 with the same statement. This simply means that you determine the value of the bits used for handshake control, determine the value of the bits used for normalization, add those two values together, and use the sum as your control byte.

If for some reason you need to change the handshake bits after the normalization bits have been set, that's OK. The value of bits previously in the register can easily be maintained by using a couple extra statements. The general technique is to read the current register contents, mask out the old handshake bits, OR in the new handshake bits, then place the result back into the register. The following example shows the details of this process.

```
100 STATUS 4,4 ; C ! Read current value
110 C=BINAND(15,C) ! Clear B4 thru B7
120 C=BINIOR(64,C) ! Set partial handshake mode
130 CONTROL 4,4 ; C ! Write new value
```

Another register affecting handshake is control Register 6. This register establishes the delay time between data output and the setting of CTL, and it establishes the width of a strobe pulse. These times are shown in Handshake Methods as "tD". The value in Register 6 is used to estblish a delay time which is added to a minimum time that is always present. The minimum times are generally around 60 µs, but there are exceptions. Refer to appendix A for more specific details.

**Register 6 - CTL Delay and Strobe Pulse Duration**

Most Significant Bit | Least Significant Bit

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| Increment 0 = 10 µs 1 = 1 ms | Delay: Number of Increments | | | | | | |
| Value = 128 | Value = 64 | Value = 32 | Value = 16 | Value = 8 | Value = 4 | Value = 2 | Value = 1 |

The control byte in this register contains a delay value (bit 0 thru bit 6) and a range selection bit (bit 7). When bit 7 is clear, the value of the other bits is multiplied by 10 µs to determine the additional delay time. When bit 7 is set, the value of the other bits is multiplied by 10 ms to determine the additional delay time. In other words, the lower seven bits specify how many time intervals to use, and bit 7 defines the size of each interval. This system yields two overlapping ranges that include times from 10 µs to 127 ms. Here are some examples:

```
CONTROL 4,6 ; 20 ! Add 200 us of CTL delay
CONTROL 4,6 ; 130 ! Set 2 ms CTL delay
CONTROL 4,6 ; 128+50 ! Set 50 ms strobe pulse
```

The final handshake-related register is Register 9. This register has only one active bit. It is used to select the Output Inhibit function. Although this feature is most often used with certain strobe handshake devices, it can be used with any handshake method. When the Output Inhibit function is disabled, all output handshakes work exactly as described in Handshake Methods.

If the Output Inhibit function is enabled, the output sequence is slightly modified. Enabling this function causes an additional handshake line to be assigned as an **inhibit** line. If the output port is using CTLA as a handshake line, then ST0 becomes the inhibit line. If the output port is using CTLB as a handshake line, then ST1 becomes the inhibit line. **If the output port is already using ST0 or ST1 as a normal handshake line, then the Output Inhibit function cannot be used.** The action of the Output Inhibit function is simple. Before starting an output cycle, the interface first checks the inhibit line. If the inhibit line is FALSE, the handshake proceeds in the normal manner. If the inhibit line is true, the interface waits until inhibit returns to the false state before proceeding with the output operation.

This function makes it easy to interface to devices that have a general **busy** line that is not part of the normal handshake sequence. An example is a printer with an internal buffer. This type of device often uses a strobe handshake, since all the internal buffer needs is a pulse to latch the valid data. However, once the buffer is full, or a line ending is received, the printer **goes busy** and prints the entire buffer. Buffers of this type usually cannot print and receive data at the same time. The GPIO interface must halt its output while the printer is busy, hence the use of the Output Inhibit function.

**Register 9 - Output Inhibit Function**

Most Significant Bit — Least Significant Bit

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| Not Used | | | | | | | Enable Output Inhibit |
| Value = 128 | Value = 64 | Value = 32 | Value = 16 | Value = 8 | Value = 4 | Value = 2 | Value = 1 |

Bit 0 is the only bit used in this register. When bit 0 is clear (value = 0), the Output Inhibit function is disabled and no inhibit line is used. When bit 0 is set (value = 1), the Output Inhibit function is enabled and the inhibit line is assigned and monitored as described two paragraphs ago. Access to this register is shown in the following example statements:

```
CONTROL 4,9 ; 1 ! Enable Output Inhibit
CONTROL 4,9 ; 0 ! Don't Inhibit Output
```

## Setting the Logic Polarity

Register 3 allows you to individually determine the logic sense of each handshake line. Register 4 allows you to individually determine the logic sense of each data port. This is a tremendous amount of flexibility. The term **logic sense** (or logic polarity) refers to whether the lines are treated as positive-true or negative-true. A positive-true line interprets a logic low as a "0" and a logic high as a "1". A negative-true line interprets a logic low as a "1" and a logic high as a "0". Note that if you change the normalization of any CTL line, the line changes states immediately after the normalization bit is changed in the control register.

**Register 3 - Handshake Line Normalization**

Most Significant Bit — Least Significant Bit

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| Invert ST1 | Invert STO | Invert FLGB | Invert FLGA | Invert CTL1 | Invert CTLB | Invert CTLO | Invert CTLA |
| Value = 128 | Value = 64 | Value = 32 | Value = 16 | Value = 8 | Value = 4 | Value = 2 | Value = 1 |

Each bit in this register corresponds to one handshake line. When a bit is "0", its corresponding line is positive-true. When a bit is "1", its corresponding line is negative-true. In other words, each bit is used to enable or disable an inversion for its corresponding line. The following table shows the polarity definitions of the handshake lines.

| Line Type | Normalization Bit | Logic Sense |
|---|---|---|
| FLG (or ST) | 0 | Logic HI = BUSY<br>LOGIC LO = READY |
| | 1 | Logic HI = READY<br>Logic LO = BUSY |
| CTL | 0 | Logic HI = TRUE<br>Logic LO = FALSE |
| | 1 | Logic HI = FALSE<br>Logic LO = TRUE |

Here are some example statements:

```
CONTROL 4,3 ; 15 ! Invert CTL lines
CONTROL 4,3 ; 128+8 ! Invert Port D hndsk lines
CONTROL 4,3 ; 16+32 ! Invert FLGA and FLGB
```

**Register 4- Data Normalization and Handshake Control**

Most Significant Bit | Least Significant Bit

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| Handshake Control (see "Selecting the Handshake Method") | | | Not Used | Invert Port D Data | Invert Port C Data | Invert Port B Data | Invert Port A Data |
| Value = 128 | Value = 64 | Value = 32 | Value = 16 | Value = 8 | Value = 4 | Value = 2 | Value = 1 |

This is the same Register 4 discussed in Selecting the Handshake Method. This time we are interested in the lower four bits. Each of these bits corresponds to one of the data ports. When a normalization bit is "0", all the data lines on the corresponding port are positive-true. When a normalization bit is "1", all the data lines on the corresponding port are negative-true.

Because this register contains normalization bits and handshake control bits, you should pay particular attention to what you are doing when you write to it. As was mentioned in Selecting the Handshake Method, the cleanest approach is to set all the options with one statment. This simply means that you determine the value of the bits used for handshake control, determine the value of the bits used for port normalization, add those two values together, and use the sum as your control byte. For example, assume that you wanted to output negative true data from Port A using strobe handshake. The handshake control value is 128. To invert the data lines on Port A, a value of "1" is used. The total of these two values is 129. Therefore, the following statement would be used:

```
CONTROL 4,4 ; 129 ! Strobe handshake; invert Port A data
```

isc reg# control byte

If for some reason you need to change the normalization bits after the handshake bits have been set, you can use the masking technique discussed in Selecting the Handshake Method. The following examples show two methods of isolating normalization bits. The first example sets Port A and Port C to negative true, sets Port B and Port D to positive true, and leaves the handshake control bits unchanged.

```
100 STATUS 4,4 ; C ! Read current value
110 C=BINAND(240,C) ! Clear B0 thru B3
120 C=BINIOR(5,C) ! Invert Port A & C
130 CONTROL 4,4 ; C ! Write new value
```

The second example shows how the normalization of a single port can be changed without affecting any other bits in the register.

```
250 STATUS 4,4 ; X ! Read current value
260 X=BINAND(X,BTD("11111101")) ! Port B = pos. true
270 CONTROL 4,4 ; X ! Write new value
```

## Enabling Output

So far you have seen how to set the method, timing, and polarity of handshake, how to set the logic polarity of the data, and how to address a port of the proper size, direction, and drive capability. This is enough information to get most of the ports working, but there is an extra little "trick" needed to activate the output drivers on Port A and Port B.

The GPIO interface has protection mechanisms built in that must be satisfied before Port A or Port B will output. The reason for this is to ensure that the output drivers are not accidentally activated while they are grounded or connected to current-sourcing circuitry. If the GPIO is trying to drive a device that is also trying to drive the GPIO, an electrical conflict will result. In other words, don't try an output operation on an input port. Without safeguards, this could happen as the result of a simple typing error when entering a device selector.

It is unlikely that you will accidentally generate an output from Port A or Port B. To enable the output drivers of Port A or Port B, you must set an enable bit in Register 8. To set any enable bits in Register 8, switch 4 of the default configuration switches must be on. The details on accessing and setting this switch are in the Installation section. Trying to write to Register 8 without first setting this switch results in Error 115.

**Register 8 - Output Enable for Port A and Port B**

Most Significant Bit | Least Significant Bit

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| Not Used | | | | | | Enable Port B Outputs | Enable Port A Outputs |
| Value = 128 | Value = 64 | Value = 32 | Value = 16 | Value = 8 | Value = 4 | Value = 2 | Value = 1 |

Assuming that switch 4 has been properly set, the following example shows an output enable operation for Port B.

```
CONTROL 4,8 ; 2 ! Enable Port B output
```

## Choosing the Method of Transfer

The basic software link with the GPIO interface is the program statement used to perform the I/O operation. Assuming that you have studied the *I/O ROM Owner's Manual*, choosing the proper statement is usually a simple matter. Some of the factors to be considered are reviewed here.

If you are using an 8-bit port and dealing with ASCII data, most of your needs can be met with simple OUTPUT and ENTER statements. For example, string data is output to Port B by the following statement:

```
OUTPUT 401 ; A$
```

The following statement inputs a number that is being sent to Port A as an ASCII representation:

```
ENTER 400 ; N
```

If straight binary numbers are being transferred, you will probably want to suppress the end-of-line sequence and use binary formatting. An 8-bit binary number can be output to Port C as follows:

```
OUTPUT 404 USING "#,B" ; X
```

The following is an example of inputting a 16-bit binary number from Ports A & B using Port A handshake lines:

```
ENTER 408 USING "#,W" ; K
```

If you are using 16-bit ports without the "W" image, be careful to move an even number of bytes. Attempting to move an odd number of bytes through a 16-bit port generates Error 113. A common oversight is trying to output data using free-field format. If your 16-bit data is stored as a string, the string must contain an even number of bytes and a "K" image should be used.

All I/O formatting capabilities are available for use with the GPIO interface. An image is used in the following example to output a column of numbers using the output side of the Port B/Port D combination:

```
OUTPUT 403 USING "4Z.DD,/" ; A,B,C
```

The details of alternate transfer methods, such as fast handshake or interrupt transfer, are covered next.

## Advanced Capabilities

The first half of this section dealt with the basic characteristics of a parallel interface. This half covers the extensions to those capabilities that are provided by the HP 82940A GPIO Interface. These extra features are:

- Specialized transfers: fast handshake and interrupt.
- User-defined end-of-line (EOL) sequences.
- Sensing and controlling individual handshake lines.
- Parity generating and checking.
- Using external events to generate interrupts.
- A trigger-byte function for transfer control.

It may be necessary to review the statements made available by the I/O ROM; see the Syntax Summary in the *I/O ROM Owner's Manual.*

### FHS and INTR Transfers

In addition to the normal OUTPUT and ENTER statements, the GPIO interface can be used with TRANSFER...FHS and TRANSFER...INTR statements. No special configuration is necessary to use INTR transfers. Simply set the control registers as you would for a normal OUTPUT or ENTER. The GPIO interface accepts DELIM and COUNT terminating conditions for an input transfer, but EOI is not defined. Transfer speeds are limited to about 400 bytes per second using this method. The following example program shows the use of an interrupt transfer to input string data using full handshake and positive-true logic on Port A.

```
 10 DIM A$[80],B$[88]
 20 IOBUFFER B$
 30 ON EOT 4 GOSUB 140
 40 TRANSFER 400 to B$ INTR ; DELIM 10
 50 ! ASCII 10 IS A LINE FEED CHAR.
    :
    Dummy loop; indicatesother processing
    :
 80 K=1
 90 DISP K;@ K=K+1
100 GOTO 90
    :
    EOT Routine
    :
140 ENTER B$ ; A$ ! Empty buffer
150 PRINT A$
160 TRANSFER 400 TO B$ INTR ; DELIM 10 @ RETURN
```

Notice that the TRANSFER statements have a DELIM 10 parameter. Ten is the value of a line-feed character. This is a common delimiter when dealing with string data because it allows the transfer to terminate at the end of each line of incoming text. Without the DELIM parameter, the transfer would not terminate until the buffer was full. Another point to remember about buffers is shown in line 140. The incoming data could be viewed by simply printing B$. However, this approach would leave data in the buffer. When more data came in, it would be appended to the existing data until an ERROR 126 was generated. Since this is usually not the desired action, the buffer should be emptied after each transfer. Entering from the buffer is one way to empty it (as shown in the last example). Another way to empty a buffer is to re-execute the IOBUFFER statement. For example:

```
140 PRINT B$
150 IOBUFFER B$ ! Empty buffer
```

Performing an IOBUFFER is faster and simpler than performing an ENTER, but it also clears any CONVERT values you may have established. If this is not acceptable, the third choice is to perform a CONTROL to the buffer's pointers. This is the fastest way to empty a buffer, although the statement is somewhat cryptic. The following is an example:

```
140 PRINT B$
150 CONTROL B$,0 ; 1,0 ! Empty buffer
```

ABORTIO, ASSERT, HALT, RESET, and STATUS statements can be executed during an interrupt transfer. These statements take affect immediately and are the only statements that do so. All other interface-controlling statements (such as CLEAR, CONTROL, ENTER, OUTPUT, or another TRANSFER) wait until the end of the current transfer before taking effect. Therefore, the GPIO interface can only do one TRANSFER at at time (ENTER from a buffer and OUTPUT to a buffer are not interface-controlling statements and can be performed at any time).

The fast handshake transfer is a special case. In order to achieve maximum speed, certain constraints are placed on this transfer method. A FHS transfer is allowed for 8-bit data only. This transfer type must use the Port A/Port C bidirectional configuration (primary address 06). Only full handshake is allowed, with FLGA and CTLA as the handshake lines. Normalization changes for FLGA, CTLA, Port A and Port C are allowed, but alternate handshake methods or port configurations cannot be used. A COUNT terminating condition can be specified for an input transfer, but EOI is not defined. Transfer speeds of about 18,000 bytes per second can be achieved using this method.

Because a FHS transfer locks out all interrupts (including the RESET key) and a full handshake is used, the computer can be completely "hung" or "locked up" if the peripheral device stops handshaking before the transfer is complete. To help deal with this problem, a special feature has been added to the GPIO interface. A FHS transfer can be aborted by placing the ST0 line in the true state. This can be done by an operator with a pushbutton or by a control line from the peripheral device. If ST0 is asserted during a FHS transfer, the transfer is aborted, ERROR 114 is generated, and the program stops. If you wish to trap this event and keep the program running, use an ON ERROR statement.

## EOL Sequence

In its default state, the GPIO interface outputs a carriage return/line feed as the end-of-line (EOL) sequence. This sequence is sent at the end of each OUTPUT statement and whenever it is called for by an IMAGE statement. If you desire a different EOL sequence, any sequence up to seven characters long can be programmed by using the CONTROL statement. The EOL sequnce is controlled by Registers 16 thru 23. Register 16 holds a number 0 thru 7. This is the number of EOL characters in the sequence. The characters themselves are stored starting in Register 17. For example, the default sequence has the value 2 (number of characters) in Register 16 and the character values 13 (carriage return) and 10 (line feed) in Registers 17 and 18, respectively. Although these registers can be changed using the CONTROL statement, they cannot be examined with the STATUS statement. In other words, they are write-only registers.

The following program statement redefines the EOL sequence to be four control characters: bell, carriage return, line feed, DC3.

```
CONTROL 4,16 ; 4,7,13,10,19
```

It is important to note that the EOL sequence is also sent at the end of an outgoing transfer operation. If you don't pay close attention to the contents of the buffer, this can produce an extra, unwanted carriage return/line feed. For example:

```
10 IOBUFFER A$
20 OUTPUT A$ ; "Hello"
30 TRANSFER A$ TO 406 INTR
```

This program sequence outputs the word "Hello" followed by a carriage return/line feed *and* an EOL sequnce. The carriage return/line feed is placed in the buffer by the OUTPUT statement. The EOL sequence is added by the GPIO after it sends the buffer contents. To avoid this duplication, one of the EOL sequences must be suppressed. To suppress the carriage return/line feed in the buffer, the following change can be made. This change uses an OUTPUT image to place the string into the buffer without any EOL characters.

```
10 IOBUFFER A$
20 OUTPUT A$ USING "#,K" ; "Hello"
30 TRANSFER A$ TO 406 INTR
```

To suppress the EOL sequence sent by the GPIO, the following change can be used. This change writes a value of zero to Register 16, which tells the interface to output zero EOL characters. Unless you need an EOL sequence different from carriage return/line feed, this method is the most convenient of the two. It is especially handy when transferring strings and text, since blank lines (null strings) can be output without worrying about transferring an empty buffer. Note that this also eliminates the need to dimension the IOBUFFER larger by the number of EOL sequence characters.

```
10 IOBUFFER A$
20 CONTROL 4,16 ; 0 ! Use no EOL characters
30 OUTPUT A$ ; "Hello"
40 TRANSFER A$ TO 406 INTR
```

Note that the programmable EOL sequence is sent to 8-bit ports only. When 16-bit ports are used (primary addresses 08 thru 15), no EOL bit patterns are sent. You do not need to suppress the EOL sequence when using 16-bit data, just be aware that it is not sent.

## Direct Use of Control Lines

The ASSERT statement has a similar definition. If you set a bit to "1", the corresponding line is placed in the true state. The true state might be a logic high or a logic low, depending upon the normalization bits in Register 3. The RES lines do not have normalization bits and are always negative true. If you use a bit value of "1" to ASSERT a RES line, that line is placed in the logic low state.

**Register 2 (read) - Line Status**

Most Significant Bit — Least Significant Bit

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| ST1 | ST0 | FLGB | FLGA | CTL1 | CTLB | CTL0 | CTLA |
| Value = 128 | Value = 64 | Value = 32 | Value = 16 | Value = 8 | Value = 4 | Value = 2 | Value = 1 |

**Assertion Control and Register 2 (write)**

Most Significant Bit — Least Significant Bit

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| RESB | RESA | Not Used | | CTL1 | CTLB | CTL0 | CTLA |
| Value = 128 | Value = 64 | Value = 32 | Value = 16 | Value = 8 | Value = 4 | Value = 2 | Value = 1 |

The following simple example shows a method of checking a handshake line. This program reads Register 2 and checks the state of ST1. If ST1 is false, the program loops back and continues checking. When the ST1 line is true, the program takes an alternate path. This example just beeps and goes back to the main loop. In actual application, the BEEP statement would be replaced by whatever action is appropriate as a response to the line being monitored.

```
10 STATUS 4,2 ; X
20 IF NOT BIT(X,7) THEN GOTO 10
30 !
40 ! Your program goes here
50 !
60 BEEP
70 GOTO 10
```

Register 2 returns the logical state of the lines, not necessarily their electrical state. Therefore, a line state shows as "1" if that line is true. It does not matter if a logic high or a logic low has been defined as the true state (see Setting the Logic Polarity).

The ASSERT statement has a similar definition. If you set a bit to "1", the corresponding line is placed in the true state. The true state might be a logic high or a logic low, depending upon the normalization bits in register 3. The RES lines do not have normalization bits and are always negative true. If you use a bit value of "1" to ASSERT a RES line, that line is placed in the logic low state.

The following example shows the use of an ASSERT statement to produce a custom handshake sequence. This example sets the CTLA line true, outputs a byte, waits 500 ms, then sets CTLA false. Notice that to control the handshake line directly, the interface handshake options must be turned off.

```
10 CONTROL 4,4 ; 192 ! Turn off handshake
20 ASSERT 4;1 ! CTLA True
30 OUTPUT 406 USING "#,B" ; X
40 WAIT 500
50 ASSERT 4;0 ! CTLA False
```

The following example pulses RESA and RESB.

```
10 ASSERT 4;192
20 ASSERT 4;0
```

The outgoing control lines can also be set by using a CONTROL statement directed to Register 2. However, the action of a CONTROL statement does not take effect until the current process of the interface is completed. The ASSERT statement, like the STATUS statement, takes effect immediately.

## Parity

The GPIO interface provides both parity generation and parity checking. *Parity* is a simple method of detecting erroneous transmissions of data. It uses one bit of data as an indicator which is set or cleared according to known rules. If the sender and receiver are both using the same set of rules, any disagreement in the state of the *parity bit* indicates a possible transmission error. The GPIO can only use parity with 8-bit ports and data that has seven bits (or less). The most common data type that fits these requirements is the ASCII character set.

There are five parity choices on the GPIO.

- Zero parity
- One parity
- Even parity
- Odd parity
- No parity

If Zero parity is selected, all outgoing characters have "0" in their eighth bit. All incoming characters are checked to make sure that the eighth bit is "0". If any incoming character has "1" in the top bit, a parity error is generated. One parity is similar to Zero parity, except that the eighth bit is set to "1" for outgoing data and checked for "1" on the incoming data.

Even parity and Odd parity are more sophisticated than that. The object of these methods is to ensure that all data bytes have an even or odd number of "1" bits in them. For example, consider the letter "A" being output with odd parity. The bit pattern for this character is "01000001". There are two bits set to "1". When odd parity is applied to the character, the top bit is set so that there are an odd number of 1's. The resultant pattern is "11000001". Even parity operates in a like manner, except that the top bit is used to guarantee an even number of "1" bits. If an incoming byte does not have the proper number of 1's in it, a parity error is generated.

A parity error can be detected in two ways. If an incoming character generates a parity error, its eighth bit is set to "1". This can be detected by the BIT function. Also, if the character is printed or displayed on the computer, the top bit shows up as an underline. The second way to detect parity is to use an ON INTR statement. This is explained in Event Interrupts.

The parity function is selected by Control Register 0. The desired parity mode is specified by using a CONTROL statement to set one of the lower four bits. The current parity mode cannot be read by using a STATUS statement. Reading the status of Register 0 always results in the value "4", which is the interface ID code.

**Register 0 (write) - Parity Control**

Most Significant Bit | Least Significant Bit

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| Not Used | | | | Enable Odd Parity | Enable Even Parity | Enable One Parity | Enable Zero Parity |
| Value = 128 | Value = 64 | Value = 32 | Value = 16 | Value = 8 | Value = 4 | Value = 2 | Value = 1 |

Here are some examples of parity selection:

```
CONTROL 4,0 ; 8 ! Use Odd parity
CONTROL 4,0 ;   ! Turn Off parity
```

If more than one parity mode is selected, the one occupying the lowest bit position has preference. For example, if you try to select both Even and Odd parity, Even parity will be used. Also, if you try to use parity with 8-bit data, the top bit of the data will be replaced by the parity bit.

## Event Interrupts

There are two kinds of interrupts available with the GPIO interface. One kind is used for interrupt transfers. This kind is handled automatically by the computer and interface. All you need to do is specify INTR in your TRANSFER statement and provide an ON EOT statement. The other kind of interrupt is used for ON INTR programming. This kind requires more involvement from the programmer and is referred to as an *event interrupt.*

The following is a summary of the steps necessary to use event interrupts with the GPIO.

1. Have an interrupt service routine in your program that performs the desired tasks as a result of the interrupt.
2. Provide an ON INTR statement to direct the program to the service routine in the event of an interrupt.
3. Use an ENABLE INTR statement to select the event(s) that you want to cause an interrupt.
4. When an interrupt takes the program to the service routine, read the STATUS of Register 1. This lets you determine the cause of the interrupt and is a necessary part of the interrupt-handling protocol (allows another interrupt to be accepted).
5. If you are expecting further interrupts, do another ENABLE INTR on the same line as the RETURN statement in the interrupt service routine.

Events that can cause an interrupt are selected and detected by using Register 1. These events and their corresponding bit positions are shown in the following diagram.

**Register 1 - Interrupt Enable/Cause**

Most Significant Bit | Least Significant Bit

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| ST1 Interrupt | ST0 Interrupt | FLGB Interrupt | FLGA Interrupt | Not Used | | Parity Error Interrupt | Not Used |
| Value = 128 | Value = 64 | Value = 32 | Value = 16 | Value = 8 | Value = 4 | Value = 2 | Value = 1 |

The first example shows the simplest form of interrupt handling. This program counts pulses on the FLGA line by using interrupts. Each time FLGA goes true, an end-of-line branch directs the program to the interrupt service routine (line 120). This routine performs the necessary `STATUS 4,1` statement, waits for the end of the pulse, increments a counter, enables the interface for the next interrupt, and then returns to the main program. The main program simply monitors the counter and beeps when a count of 10 is reached. In actual applications, an action more significant than beeping would be included. For example, the interrupt might be generated by a photocell detector that is counting items moving on a conveyor into a packing carton. When the proper number of items is in the carton, the main program might send bit patterns to an output port that controls the folding and sealing of the carton.

```
 10 ON INTR 4 GOSUB 120
 20 ENABLE INTR 4;16 ! FLGA interrupt
 30 !
 40 ! Main program goes here
 50 !
 60 N=0
 70 IF N<10 THEN GOTO 70 ! Check counter
 80 BEEP @ GOTO 60
 90 !
100 ! Interrupt Service Routine
110 !
120 STATUS 4,1 ; S
130 ! Wait for end of FLGA pulse
140 STATUS 4,2 ; X
150 IF BIT(X,$) THEN 140
160 N=N+1 ! Increment counter
170 ENABLE INTR 4;16 @ RETURN
```

When dealing with interrupts in real-life applications, it is very important to consider the timing of the interrupt pulse. Both the width and the frequency of the pulse are important factors. The number of interrupts per second that can be handled depends greatly upon the amount of processing being done in the interrupt service routine and the amount of time spent on the lines in the main program. To get the fastest response time, keep the main program moving from line to line as fast as possible. This can be done by coding efficiently and by not using the @ symbol to combine statements on a line. In any event, it is unlikely that you will be able to handle interrupts that are more closely spaced than 25 ms or so.

Interrupt processing can be adversely affected by pulse widths that are too long or too short. An interrupt pulse must be over 100 $\mu$s long for the interface to see it under ideal conditions. If the interface is busy with other tasks, such as handling data or communicating with the computer, even longer pulses are necessary. However, if the interrupt pulse is too long, the same pulse can cause more than one interrupt. This happens if the computer makes it through the interrupt service routine and re-enables interrupts before the pulse has ended. This problem does have a solution, as shown by lines 140 and 150 in the preceding example. If you expect a relatively long pulse, you can use the `STATUS` statement to ensure that the pulse has ended before re-enabling for more interrupts. When this technique is used, the only disadvantage of long interrupt pulses is that they will slow down the program.

Not all event interrupts are based on pulses. A handshake action is sometimes employed. In these applications, the interrupting device holds its line true until the interface sends a signal that clears it. When this is the case, the interrupt service routine can use `ASSERT` statements to cause the peripheral device to drop its request before interrupts are re-enabled.

The next example shows the detection of more than one interrupt cause. Assume for this example that the computer is controlling a test station. This is a hypothetical test for circuit reliability at certain temperatures. The example program does two things. The main loop sends data to the circuit under test and reads the response. If the response is equal to the data sent, the circuit is operating properly. This operation uses full handshake on the Port A/Port C combination.

The interrupt service routine runs the temperature controller. An interrupt from ST0 indicates that the heater must be turned on, while an interrupt from ST1 tells the computer to turn the heater off. The heater is controlled by bit 0 in Port D. This operation uses strobe handshake.

Admittedly, a simple thermostat could be set up without involving the computer. This example merely demonstrates some principles of handling interrupt-driven events. In actual applications, you will probably be doing something more complex than turning a heater on and off.

```
 10 IMAGE #,B
 20 CONTROL 4,4 ; 0 ! Set full handshake
 30 ON INTR 4 GOSUB 170
 40 ENABLE INTR 4;(128+64) ! ST0 & ST1 interrupts
 50 !
 60 E=0 ! Reset error counter
 70 FOR K=0 TO 255
 80 OUTPUT 402 USING 10 ; K ! Test value to Port C
 90 ENTER 402 USING 10 ; X ! Response from Port A
100 IF X<>K THEN E=E+1 ! Count errors
110 NEXT K
120 PRINT E;"Errors for this test"
130 GOTO 60 ! Repeat continuously
140 !
150 ! Interrupt service routine
160 !
170 STATUS 4,1 ; S
180 IF BIT(S,6) THEN H=1 ! Heater on
190 IF BIT(S,7) THEN H=0 ! Heater off
200 CONTROL 4,4 ; 128 ! Go to strobe handshake
210 OUTPUT 403 USING 10 ; H ! Heater byte to Port D
220 CONTROL 4,4 ; 0 ! Return to full handshake
230 ENABLE INTR 4;(128+64) @ RETURN
```

Notice that a binary image is used for all I/O statements because actual binary values are being handled, not ASCII representations. Primary address 03 is used to access Port D so that FLGB and CTLB are used for handshake. If primary address 05 were used, there would be a conflict between the use of ST1 as an interrupt line and as a handshake line. Notice also that Register 4 defines the handshake mode for all ports on the interface. Line 200 sets up a strobe handshake for the Port D operation. Then line 220 re-establishes the full handshake mode that is needed by Port A and Port C in the main routine.

The previous example assumed that the two interrupts would never occur at the same time. This is not always the case. Many applications need to handle multiple interrupts that may occur in any order or in any combination. To deal with this situation, it is recommended that you employ a carefully structured polling routine. The need for proper polling is especially critical since the HP Series 80 Personal Computers do not have a priority system for interrupts (see Interactions and Permutations in section 7 of the *I/O ROM Owner's Manual).* The recommended polling technique is shown in the following example.

This program uses interrupts from ST0, ST1, and FLGB to demonstrate the concept of polling. When an interrupt occurs, the program branches to line 100. The required `STATUS` statement reads the interrupt cause register. Note that only one `STATUS 4,1` statement is used. The interrupt cause register is automatically cleared when it is read. Therefore, attempts to identify multiple causes by repeatedly reading Register 1 would be futile. This is not a problem since the register contents are placed in variable `S`, where they can be inspected as often as necessary. Variable `S` is tested by the `BIT` function to isolate the interrupt causes. The `IF ... GOSUB` structure is the simplest and least confusing method of dealing with multiple causes. This technique allows all causes to be tested before returning to the main program and yields independent subroutines for handling each cause. If two or more interrupts occur simultaneously, the one that is tested first in the polling routine will be serviced first.

```
 10 ON INTR 4 GOSUB 100
 20 E=128+64+32 ! IRQ conditions
 30 ENABLE INTR 4;E
 40 !
 50 ! Main program goes here
 60 GOTO 60
 70 !
 80 ! Interrupt polling routine
 90 !
100 STATUS 4,1 ; S
110 IF BIT(S,5) THEN GOSUB 180
120 IF BIT(S,6) THEN GOSUB 210
130 IF BIT(S,7) THEN GOSUB 240
140 ENABLE INTR 4;E @ RETURN
150 !
160 ! Interrupt service routines
170 !
180 DISP "FLGB Interrupt"
190 RETURN
200 !
210 DISP "ST0 Interrupt"
220 RETURN
230 !
240 DISP "ST1 Interrupt"
250 RETURN
```

**Note:** Do not use FLG or ST interrupts and the TRANSFER statement at the same time. During a transfer, the interface's resources are dedicated to the data movement operation. An external interrupt is not recognized during a transfer. If a FLG or ST interrupt occurs at the end of an incoming transfer, the computer and interface might *lock up.*

Parity error is the only event interrupt that is recognized during an interrupt transfer. Interrupts for FLG and ST lines should be disabled before a transfer starts and re-enabled (if desired) after the transfer completes. Any form of event interrupt may be used in conjunction with OUTPUT and ENTER statements, but remember that the end-of-line branch is not taken until the OUTPUT or ENTER statement completes.

## The Trigger Function

An elegant feature of the HP 82940A GPIO Interface is its ability to initiate actions upon the detection of a trigger byte. The GPIO interface can input and inspect data without interacting with the computer, thereby freeing computer time for other operations. The trigger byte is defined by a CONTROL statement. Incoming data can be tested for conditions "less than", "greater than", or "equal to" the trigger byte, or any combination of these conditions. When a trigger condition is detected, the interface can initiate an interrupt transfer or signal the peripheral device with a CTL line. The registers involved are Register 5 and Register 7.

The trigger byte itself is written into or read from Register 7. The following statement defines the binary value 127 as the trigger byte:

```
CONTROL 4,7 ; 127
```

The trigger actions are established by the top four bits in Register 5. The lower four bits of this Register hold the currently-assigned primary address. As was the case for Register 4, writing only high-order bit values will clear the lower bits. However, if primary addresses are included in the OUTPUT, ENTER, or TRANSFER statements, Register 5 will be updated automatically for every operation. This means that you don't have to worry about those lower four bits if the primary addresses are included in your program. If your are concerned about preserving the primary address in Register 5 for some special reason, use a masking technique like the one explained in Selecting the Handshake Method.

**Register 5 - Primary Address and Trigger Action**

Most Significant Bit | Least Significant Bit

| Bit 7 | Bit 6 | Bit 5 | Bit 4 | Bit 3 | Bit 2 | Bit 1 | Bit 0 |
|---|---|---|---|---|---|---|---|
| Trigger If Data <R7 | Trigger If Data =R7 | Trigger If Data >R7 | If Trigger, Pulse CTL | Primary Address | | | |
| Value = 128 | Value = 64 | Value = 32 | Value = 16 | Value = 8 | Value = 4 | Value = 2 | Value = 1 |

The trigger bits can be summarized as follows:

| Comparison Test | Register 5 Value |
|---|---|
| Data > Register 7 | 32 |
| Data ⩾ Register 7 | 96 |
| Data < Register 7 | 128 |
| Data ⩽ Register 7 | 192 |
| Data = Register 7 | 64 |
| Data ≠ Register 7 | 160 |
| No Trigger | 0 |

The primary trigger function is to initiate an interrupt transfer. The following example establishes an interrupt transfer that begins when the interface detects a DC2 control character (decimal value 18) and ends when the buffer is full. Port A of a Port A/Port C combination is used for the input.

```
 10 DIM B$[40]
 20 IOBUFFER B$
 30 CONTROL 4,5 ; 64 ! Start if = trigger
 40 CONTROL 4,7 ; 18 ! DC2 is trigger
 50 ON EOT 4 GOTO 140
 60 TRANSFER 402 TO B$ INTR ; DELIM 10
 70 !
 80 ! Dummy loop; indicates other processing
 90 !
100 K=1
110 DISP K;@ K=K+1
120 GOTO 110
130 !
140 ! EOT Routine
150 !
160 ENTER B$ ; X,Y
170 PRINT X,Y
180 END
```

In addition to initiating a transfer, the trigger function can also signal a peripheral device that the trigger byte has been detected. The signal is a short pulse on a CTL line. If FLGA/CTLA are being used as the handshake lines, the pulse is sent on CTL0. If FLGB/CTLB are being used as the handshake lines, the pulse is sent on CTL1. The line is pulsed from false to true and back to false, with normalization changes allowed. The pulse width is roughly 40 $\mu$s.

This function, called *auto response*, is enabled in Register 5. If bit 4 of Register 5 is set to "1", the appropriate CTL line will be pulsed when an incoming byte meets the trigger byte requirements. If bit 4 is "0", the auto response function is disabled. To enable this function, add 16 to the values shown in the preceding summary table.

Note that this feature is available in addition to the automatic start of a transfer. Auto response cannot be selected without setting up a transfer of at lease one byte. The following example shows the minimum software required to produce an auto response pulse. This example is inputting data from Port A and using CTL0 as the auto-response line. The pulse will be sent when the first control character (value<32) is input.

```
 10 IOBUFFER Z$
 20 CONTROL 4,7 ; 32 ! Trigger byte
 30 CONTROL 4,5 ; 128+16 ! IF < R7, pulse CTL
 40 ON EOT 4 GOSUB 100
 50 TRANSFER 400 TO Z$ INTR ; COUNT 1
 60 !
 70 ! Main program goes here
 80 GOTO 80
 90 !
100 ! EOT routine (does nothing)
110 !
120 RETURN
```

This example can be enhanced to produce a CTL0 pulse for every trigger condition detected, instead of just the first. Assume that you want an auto response pulse for each form-feed character (value=12). The EOT routine is changed to empty the buffer and start a new transfer.

```
 10 IOBUFFER Z$
 20 CONTROL 4,7 ; 12 ! FF is trigger
 30 CONTROL 4,5 ; 64+16 ! If = R7, pulse CTL
 40 ON EOT 4 GOSUB 100
 50 TRANSFER 400 TO Z$ INTR ; COUNT 1
 60 !
 70 ! Main program goes here
 80 GOTO 80
 90 !
100 ! EOT routine
110 !
120 IOBUFFER Z$ ! Empty buffer
130 TRANSFER 400 TO Z$ INTR ; COUNT 1 @ RETURN
```

Section 4

# Maintenance, Service, and Warranty

## Maintenance

There are no customer serviceable parts inside the HP 82940A GPIO Interface. It should not be necessary to clean the interface module or cable contacts. The action of installing the module in the port or plugging the cable into a peripheral is normally sufficient to clean contamination from the contacts.

## Service

If at any time you suspect that the interface may be malfunctioning, do the following:

1. Turn off the computer and all peripherals. After diconnecting all plug-in devices from the ports, turn on the computer. If the cursor appears and no error message is displayed, the computer is functioning properly.
2. Turn off the computer. After installing the interface module in question in any port, turn on the computer.
   - If `Error 110 : I/O CARD` appears, the interface module requires service.
   - If the cursor does not appear, the system is not operating properly. To help determine if the interface module is interfering with proper operation, repeat this step with the module installed in a different port.
3. If improper operation is indicated in either the interface module or the computer, repair service is required.

## Warranty and Repair Service Information

The warranty statement and procedures for obtaining repair service are contained on the Warranty and Service Information sheet shipped with your HP 82940A GPIO Interface. If you need additional information, please contact your authorized HP dealer or the nearest Hewlett-Packard sales and service facility.

If you have any questions concerning the warranty, please contact:

| | |
|---|---|
| **In the U.S.:** | One of the six Field Repair Centers listen on the Service Information Sheet packaged with your owner's documentation. |

| | |
|---|---|
| **In Europe:** | Hewlett-Packard S.A.<br>7, rue du Bois-du-lan<br>P.O. Box<br>CH-1217 Meyrin 2<br>Geneva<br>Switzerland<br>Tel. (22) 82 70 00 |
| **Other Countries:** | Hewlett-Packard Intercontinental<br>3495 Deer Creek Road<br>Palo Alto, California 94304<br>U.S.A.<br>Tel. (415) 857-1501 |

If your system malfunctions and repair is required, you can help assure efficient service by providing the following items with your unit(s):

1. A description of the configuration of the HP Series 80 Personal Computer system, exactly as it was at the time of malfunction, including ROMs, interfaces, and other peripherals.
2. A brief yet specific description of the malfunction symptoms for service personnel.
3. Printouts or any other materials that illustrate the problem area.
4. A copy of the sales slip or other proof of purchase to establish the warranty coverage period.

Each computer and peripheral carries an individual serial number. It is recommended that you keep a separate record of this number. Should your unit be stolen or lost, the serial number is often necessary for tracing and recovery, as well as any insurance claims. Hewlett-Packard does not maintain records of individual owner's names and unit serial numbers.

### General Shipping Instructions

Should you ever need to ship any portion of your computer system, be sure it is packed in a protective package (use the original case), to avoid in-transit damage. Hewlett-Packard suggests that the customer always insure shipments.

If you happen to be outside of the country where you bought your computer or peripheral, contact the nearest authorized HP-83/85 dealer or the local Hewlett-Packard office. All customs and duties are your responsibility.

## Radio Frequency Interference Statement

The HP 82940A GPIO Interface uses radio frequency energy and may cause interference to radio and television reception. The interface has been type-tested and found to comply with the limits for a Class B computing device in accordance with the specifications in Subpart J of Part 15 of the FCC Rules. These specifications provide reasonable protection against such interference in a residential installation. However, there is no guarantee that interference will not occur in a particular installation. If the interface does cause interference to radio or television, which can be determined by turning the computer on and off with the interface installed and with the interface removed, you can try to eliminate the interference problem by doing one or more of the following:

- Reorient the receiving antenna.
- Change the position of the computer with respect to the receiver.
- Change the position of the interface cables and peripherals with respect to the receiver.
- Move the computer away from the receiver.
- Plug the computer into a different outlet so that the computer and the receiver are on different branch circuits.

If necessary, consult an authorized HP dealer or an experienced radio/television technician for additional suggestions. You may find the following booklet by the Federal Communications Commission helpful: *How to Identify and Resolve Radio-TV Interference Problems.* This booklet is available from the U.S. Government Printing Office, Washington, D.C. 20402, Stock No. 004-000-00345-4.

Appendix A

# Functional Description

## Introduction

This appendix contains a description of the interface circuit operations. This includes a schematic diagram description of various interface components and signal lines. Although this information is provided, component level repair is not recommended due to the microcomputer based organization of the interface.

If the interface appears to be malfunctioning, it is recommended that you contact your nearest HP sales and service office for assistance.

## Theory of Operation

### Translator IC Description

The interface uses an 8049 microcomputer (μC) which requires a +5V TTL logic level. The microprocessor in the HP Series 80 Personal Computer (referred to here as the CPU) uses a +6V level. A special IC, known as a translator, permits communication between the two devices by providing level translation. The +5V and +6V power supplies and two +12V clock signals (Φ1 and Φ2) required by the translator are located on the computer mainframe. They are brought out to the interface via the I/O backplane when the interface is inserted into one of the four I/O ports.

- Handshaking of data and command information from the HP Series 80 Personal Computer to the μC.
- Handshaking of data from the μC to the computer.
- Interrupts issued to the computer by the μC.
- Interrupts issued to the μC by the computer.
- Fast handshake operation where the translator halts the computer with each data byte transfer to synchronize the flow of data.

It may be helpful to refer to the interface schematic diagram (figure A-7) while reading the theory discussed in this section.

### Select Codes

Three select code bits are used to define the I/O address of the interface. These bits (S2, S3, S4) are set via a switch by the user. The range of available select codes is 3 through 10.

The three select code bits and I/O address bits (A1, A2, A3) are sent to comparator circuitry within the translator. If they match, the interface is addressed. When this occurs, another bit (A0) is sent out by the comparator to specify one of two read/write locations. If A0 is low, the computer writes to the control register (CR) and reads the status register (SR). When A0 is high, the computer writes to the output buffer (OB) and reads the input buffer (IB). These four registers are the next topic of discussion.

## Translator I/O Registers

The computer sends data to the interface $\mu$C via the output buffer register (OB) and defines that data by setting appropriate bits in the calculator control register (CCR). Both registers are physically located within the translator IC. The OB is write-only by the computer and read-only by the $\mu$C. Except for bit 0 and bit 7, the CCR is also write-only by the computer and read-only by the $\mu$C. Different status bits are read by the $\mu$C for bit 0 and bit 7 than those written as bits 0 and 7 by the computer (CPU). This is illustrated in the following figure.

**$\mu$C Read CR**

| 7** | 6 | 5 | 4 | 3 | 2 | 1 | 0* |
|---|---|---|---|---|---|---|---|
| OBF | 0 | 0 | 0 | 0 | CED | COM | IBF |

**CPU Write to CR**

| 7* | 6 | 5 | 4 | 3 | 2 | 1 | 0* |
|---|---|---|---|---|---|---|---|
| RESET | 0 | 0 | 0 | 0 | CED | COM | INT |

| | |
|---|---|
| INT | Interrupt. This bit is routed directly to the T1 input of the $\mu$C which is used to interrupt. |
| COM | Command. |
| CED | End Data. Used by the computer to terminate a data transfer to the $\mu$C. |
| RESET | Resets the $\mu$C. This bit is routed directly to the reset input of the $\mu$C. |
| IBF | Input buffer full. |
| OBF | Output buffer full. |

**Figure A-1. CR Bit Assignments**

* Power-on state true.
** Power-on state false.

Two other registers within the translator are used when the computer receives data from the interface $\mu$C. These registers are the input buffer register (IB) used to handle the data and the status register (SR) which contains status bits to implement communication protocol. The IB is write-only by the $\mu$C and read-only by the computer. Except for bit 0 and bit 7, the SR is also write-only by the interface $\mu$C and read-only by the computer. Different status bits are read for bits 0 and 7 by the CPU than those written as bits 0 and 7 by the $\mu$C. This is illustrated in the following figure.

**$\mu$C Write to SR**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0* |
|---|---|---|---|---|---|---|---|
| HLTEN | TFLG | 0 | 0 | PACK | PED | BUSY | SERVICE |

**CPU Read SR**

| 7* | 6 | 5 | 4 | 3 | 2 | 1 | 0* |
|---|---|---|---|---|---|---|---|
| OBF | 0 | 0 | 0 | PACK | PED | BUSY | IBF |

| | |
|---|---|
| SERVICE | The $\mu$C sets this bit to initiate an interrupt to the CPU. |
| BUSY | Informs the CPU the state of the $\mu$C. When low, the $\mu$C is monitoring OBF to determine when the next command or data byte is available from the CPU. When OBF = 1, the $\mu$C sets BUSY to 1, reads the OB and CR, performs the necessary operation then returns BUSY to low. |
| PED | Processor ($\mu$C) End Data. Set high by the $\mu$C upon detection of programmed termination sequence (byte count or end character). |
| PACK | Processor ($\mu$C) Interrupt Acknowledge. The $\mu$C's interrupt service routine sets this bit after being interrupted by the CPU. |
| HLTEN | Used by the $\mu$C to halt the CPU based on the status of OBF and IBF during fast handshake data transfers. |
| IBF | Input buffer full. |
| OBF | Output buffer full. |
| TFLG | Set to tell the CPU that this interrupt is for a multiple byte transfer. |

**Figure A-2. SR Bit Assignments**

* Power-on state false.

## Interrupts

The interface interrupts the computer (CPU) as follows:

1. The interface μC sets bit 0 (Service) true in the Translator status register. This causes the interrupt line, IRLX, to go low, interrupting the CPU.
2. The CPU acknowledges the interrupt and obtains from the interface an interrupt vector. This tells the computer that this is an I/O interrupt as opposed to an interrupt from the internal keyboard chip.
3. After the interrupt is acknowledged, IRLX is returned high and all other interfaces are locked out from interrupting until re-enabled by the CPU.
4. The CPU then obtains the select code of the interrupting interface and processes the interrupt accordingly.
5. After the interrupting interface is serviced, the other interfaces are re-enabled to interrupt.

Since the computer can accommodate up to three interfaces, there must be a priority scheme if more than one requests service at the same time. This would most likely be a very rare occurrence. Essentially, the computer handles interrupts on a first-come, first-served basis. However, if more than one interface is pulling the IRLX line before the CPU acknowledges the interrupt request, the interface that gets serviced first will be the one occupying the upper-most I/O port.

The μC interrupt counter is preset to 255. The CPU initiates an interrupt by pulling CINT of the translator (T1 of the μC) low. This increments the interrupt counter from 255 to 0 (and overflow) and starts the interrupt sequence. Before the μC acknowledges the interrupt, it reads the CR to check the IBF bit. If IBF is set, it indicates the IB contains data. To prevent this data from being lost, the μC sets an input buffer restore flag. This will permit the IB to be restored after the interrupt is serviced.

The μC acknowledges the interrupt by setting the PACK bit in the SR. When PACK is set, the CPU reads the IB, which may or may not contain data, and discards it. The CPU then sets the COM (command) bit in the CR, writes an instruction into OB and returns CINT high. When CINT goes high, the μC accepts the instruction and performs the operation specified.

After the operation is performed, the CPU strobes CINT again, which increments the μC interrupt counter from 0 to 1. When this occurs, the interrupt counter gets reset to 255 and the SR is restored. If the input buffer restore flag is set, the IB is also restored. The μC then returns from interrupt.

Notice in the above sequence that CINT was strobed twice. The first time caused an overflow of the μC interrupt conter (from 255 to 0) which started the interrupt sequence. The second strobe incremented the counter from 0 to 1 which the μC detected to return from interrupt.

## Data Transfer

### GPIO Bus Lines

The standard GPIO signal lines are described next. The GPIO bus consists of two separate shielded cables (A and B) each containing 24 insulated conductors.

## Low-Power, Bi-directional Data Lines (DA0 – 7, DB0 – 7)

These data lines may be used to communicate all input and output data between the interface and the peripheral device. One character byte (when configured as separate 8-bit ports) is sent at a time in a byte serial, bit parallel fashion. In most instruments, characters are based on the 8-bit ASCII representation. As an example, if the interface is connected to a printer, the eight bits that represent one character are all sent at once in parallel. Then, the next character is sent and so on. Thus, the eight bits (one byte) defining a character are all sent at once (bit-parallel) while each character is sent serially (byte-serial). Port A and port B may be configured as one 16-bit, bi-directional port for word mode operation. Port A data lines (DA0 through DA7) are located within cable A and port B data line (DB0 through DB7) are located within cable B.

## Output Only Data Lines (DC0 — 7, DD0 — 7)

These data lines can be used to output data or address (word mode) to a peripheral device. These data lines DC0 through DC7 (port C) and DD0 through DD7 (port D) may be configured for 8-bit (byte) transfer or 16-bit (word) transfer. Port C data lines (DC0 through DC7) are located within cable A and port D data lines (DD0 through DD7) are located within cable B.

## Handshake Lines (CTLA, CTLB, CTL0, CTL1, FLGA, FLGB, ST0, ST1)

The control lines, two for each port, control the data transfer on the data lines. They allow asynchronous data transfer without timing restrictions being placed on any instrument connected to the bus. The transfer speed of each data byte is determined by the speed at which the slowest instrument is capable of sending or receiving data. The control (CTL-) lines may be written as general purpose control, and the flag (FLG-) and status (ST-) bits can be read as general purpose status bits depending on the application.

## OUTA/OUTB Lines

These lines indicate to the peripheral in which direction the data is to be transferred. Low indicates an output operation; high indicates an input operation.

## Reset Lines (RESA, RESB)

These lines, when taken low, reset the peripheral device.

## Data Handshake

Synchronization of data exchanged between the computer and the peripheral is referred to as the *handshake*. The handshake is accomplished via the CTLA/B, FLGA/B and OUTA/B lines. The peripheral receives information about the data exchange on the CTL and OUT lines and then responds on the FLG line. The OUT line tells the peripheral which direction the data is to be transferred.

Each port has two handshake lines; CTLA, FLGA (port A); CTLB, FLGB(portB); CL0, ST0 (port C) and CTL1, ST1 (port D). In the following discussion, port A is used as an example. All capabilities exist likewise for port B.

The GPIO is not an edge triggered interface. It does respond to level changes.

The handshake lines, their states and intended meanings are listed in the following table.

Table A-1. Handshake Lines

| Line | State or Mnemonic | Meaning |
|---|---|---|
| **Output** | | |
| OUTA | LOW | Computer output operation. |
| CTLA (From Interface) | CONTROL CLEAR<br>CONTROL SET | No new output data available.<br>New output data is available on output line. |
| FLGA (From Peripheral) | READY<br>BUSY | Peripheral is ready for next data transfer.<br>Peripheral is not ready for next data transfer. |
| **Input** | | |
| OUTA | HIGH | Computer input operation. |
| CTLA (From Interface) | CONTROL CLEAR<br>CONTROL SET | Computer is not requesting new data.<br>Computer is requesting new data. |
| FLGA (From Peripheral) | READY<br>BUSY | Peripheral is ready for next transfer.<br>Peripheral is not ready for next transfer. |

The reason that the state of the CTL and FLG lines are not referred to as being either high or low is that the logic level of these lines can be complemented by switch settings or dynamically under program control. The use of mnemonics for the state of these lines allows discussion of the handshake logic without referring to the specific switch settings/program chosen for your periphral.

## The Four Modes of Handshake

The four modes of handshake are referred to as full, partial, CTL- strobe and no handshake.

The user may select from one of four options for controlling handshake by modifying Register 4.

In the full mode, the CPU will check the peripheral FLG- line to ensure that it is in its ready state prior to setting the CTL- (control set) line to initiate another transfer. Refer to the Full Handshake Timing Diagram.

The user may program the interface (bit 5 of Register 4) to accept input data on FLG- going from either ready to busy or from busy to ready. The default mode at power on is to accept data on the ready to busy transition.

If burst transfer is used, only port A's handshake lines and full handshake will be used.

In the partial mode of handshake, the computer will not check the FLG- line prior to setting the CTL-line to control set. Applications which require this mode do not have true ready to busy levels on the flag line. Only a transition on the FLG- line is used to terminate the transfer. Refer to the Partial Handshake Timing Diagram, figure A-4.

In the CTL- strobe mode, no response is required on the FLG- line from the peripheral device. For either input or output operations, a strobe pulse of programmable width (within Register 6) occurs on the CTL-line. The pulse transition is from clear to set to clear again. Refer to the CTL- Strobe Timing Diagram, figure A-5. The width of this pulse is set by writing to Register 6.

When output with one of the above handshake modes is selected, a programmable delay (using Register 6) provides a delay for data to settle on long lines. The data is placed on the data lines followed by the delay and then the setting of CTL-.

In the no handshake mode, the user can control the CTL- directly by using the ASSERT statement or by writing to bits within Register 2. This mode of operation effectively provides extension of the four control bits (CTLA through CTL1). This mode also does not require a peripheral FLG- response.

## Modes of Operation

The interface has four operating modes which are available under program control. The modes are:

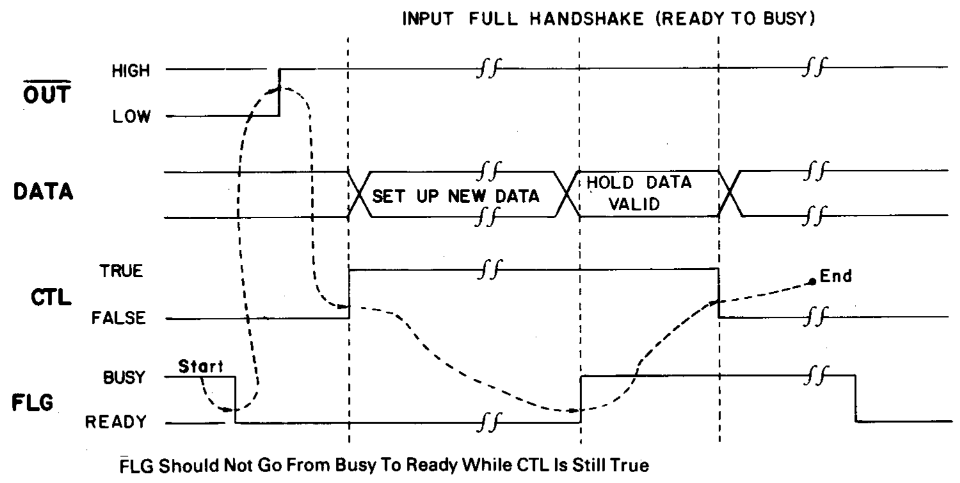
1. Byte mode
2. Word mode
3. Auto Response/Trigger mode
4. Enable Output Inhibit

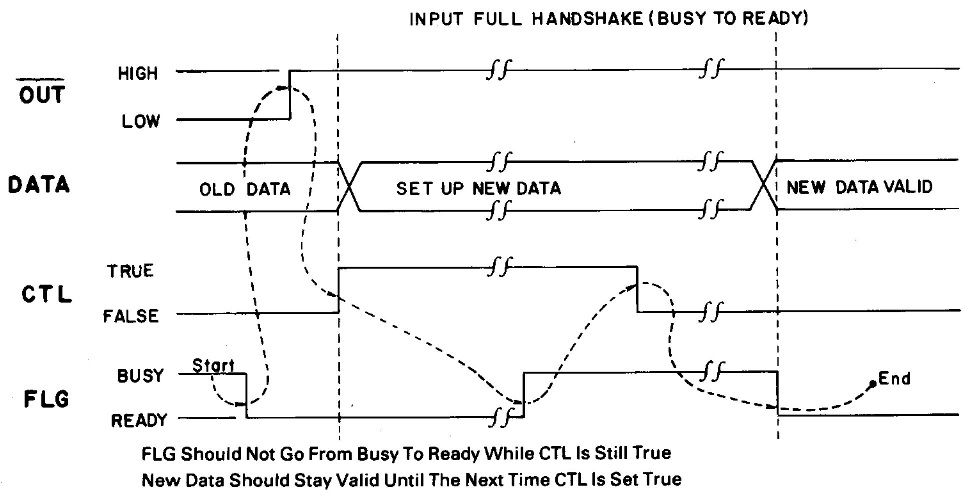
### Byte Mode

In the byte mode, ports A, B, C and D are totally independent.

### Word Mode

In word mode, ports A and B are treated as one 16-bit, bi-directional port. The user may program the interface to use either A's handshake lines (CTLA, FLGA) or B's handshake lines (CTLB, FLGB) by using port address 08 for A and 09 for B. The lines not used for handshaking may be used as general purpose control (CTL-) and status (FLG-) bits (Register 2). Ports C and D can be combined to form a 16-bit output only port.


INPUT FULL HANDSHAKE (READY TO BUSY)
OUT
HIGH
LOW
DATA
SET UP NEW DATA
HOLD DATA VALID
CTL
TRUE
FALSE
End
FLG
BUSY
READY
Start
FLG Should Not Go From Busy To Ready While CTL Is Still True


INPUT FULL HANDSHAKE (BUSY TO READY)
OUT
HIGH
LOW
DATA
OLD DATA
SET UP NEW DATA
NEW DATA VALID
CTL
TRUE
FALSE
FLG
BUSY
READY
Start
End
FLG Should Not Go From Busy To Ready While CTL Is Still True
New Data Should Stay Valid Until The Next Time CTL Is Set True

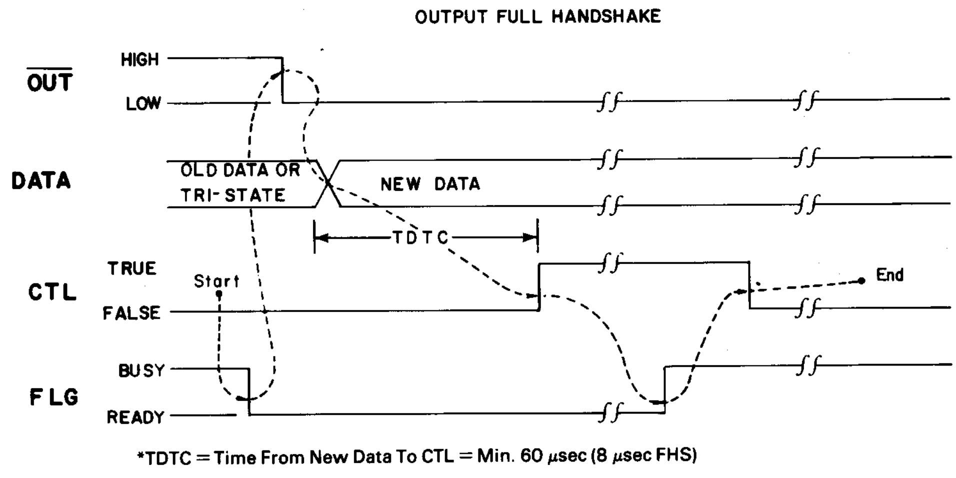
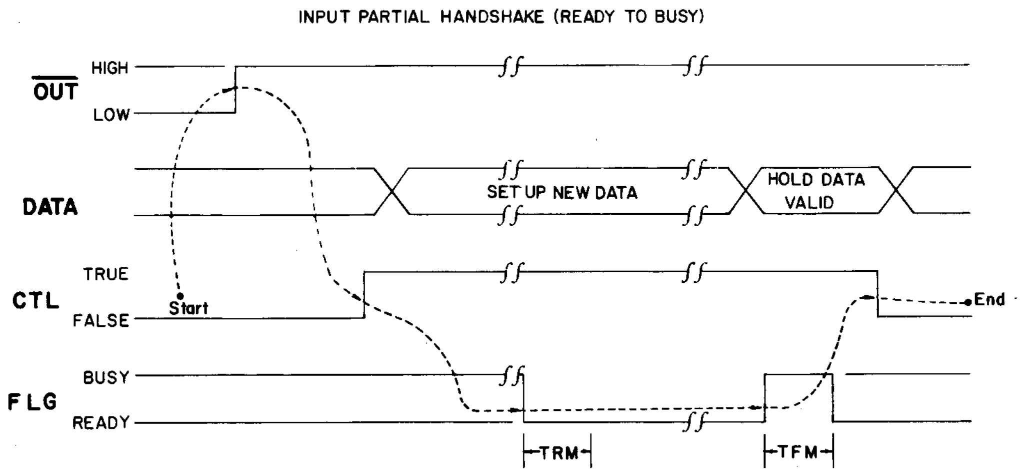
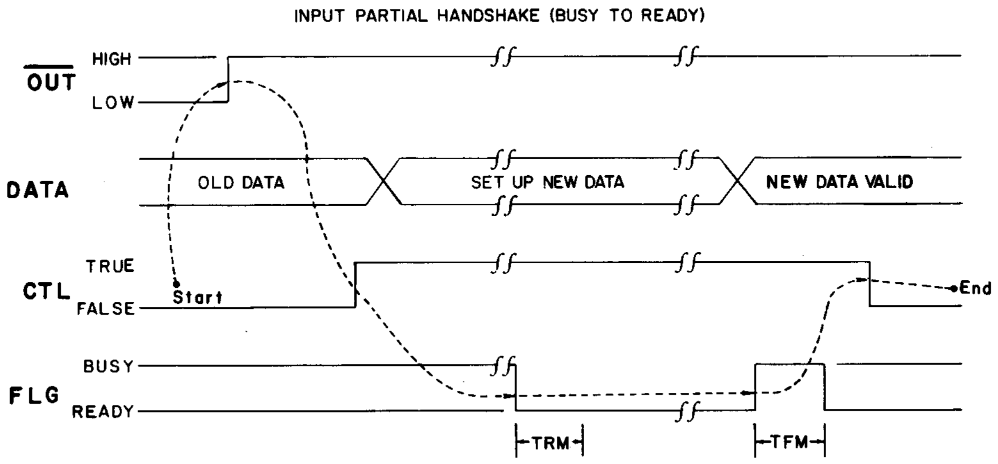


**Figure A-3. Full Handshake Timing Diagram**

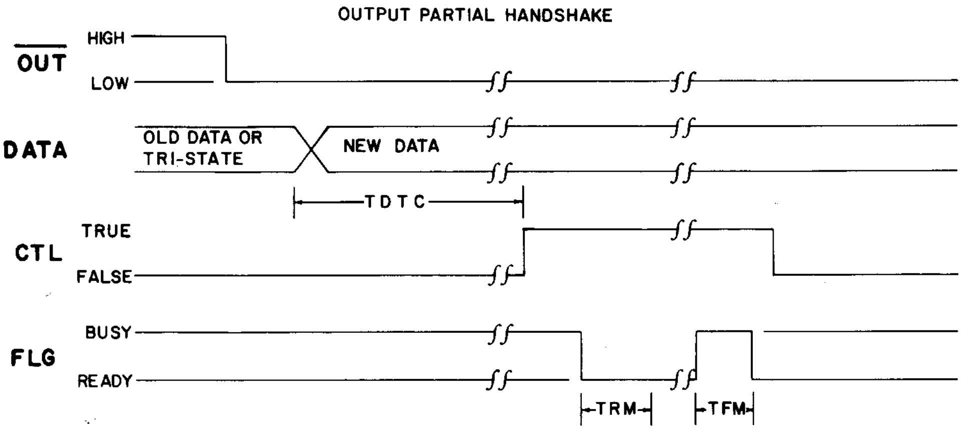
* Register 6 can be used to extend $T_{DTC}$ and $T_{CPD}$ for all handshakes except FHS.



TRM = Minimum Time FLG Must Hold Ready After CTL = 30 μsec
TFM = Minimum FLG Duration = 35 μsec



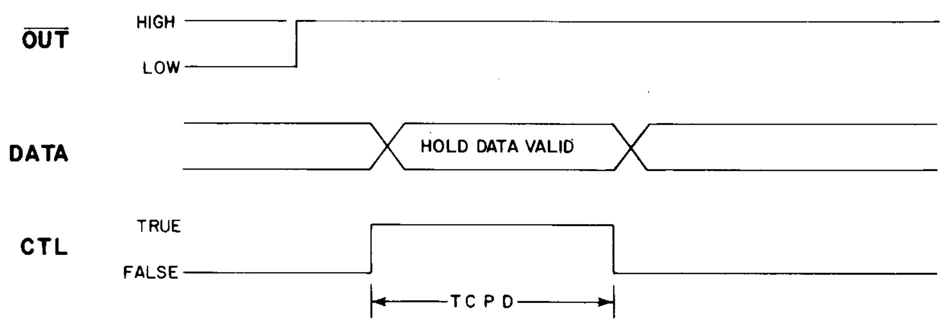
TRM = Minimum Time FLG Must Hold Ready After CTL = 30 μsec
TFM = Minimum FLG Duration = 35 μsec
New Data Should Stay Valid Until The Next Time CTL Is Set True.



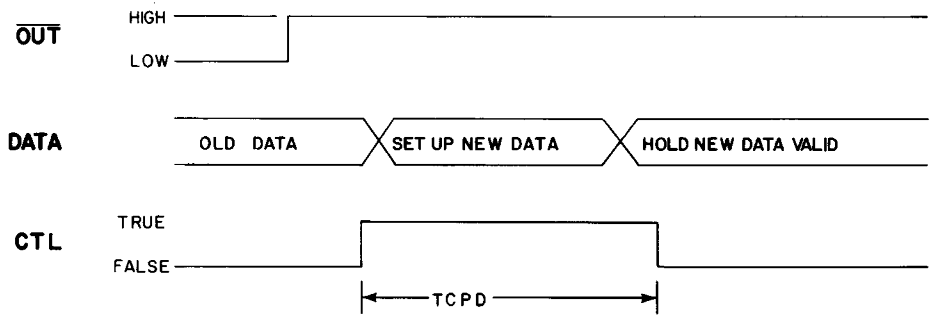
TRM = Minimum Time FLG Must Hold Ready After CTL = 30 μsec
TFM = Minimum FLG Duration = 35 μsec
*TDTC = Minimum Time From New Data To CTL = 65 μsec

**Figure A-4. Partial Handshake Timing Diagram**

INPUT STROBE (BUSY TO READY)

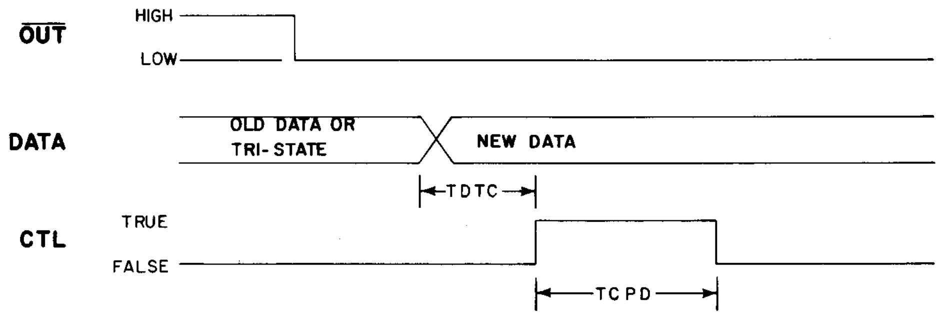


*TCPD = CTL Pulse Duration = Min 110 µsec



TCPD = CTL Pulse Duration = Min 60 µsec
New DATA Should Stay Valid Until The Next Time CTL Is Set True.

OUTPUT STROBE



*TDTC = Time from new data to CTL = 60 µsec min.
*TCPD = CTL Pulse Duration = Min 60 µsec min.
*Register 6 can be used to extend $T_{ODC}$ and $T_{CPD}$ for all handshakes except FHS.

**Note:** Timings are not valid if card is interrupted.

**Figure A-5. CTL- Strobe Timing Diagram**

### Auto Response/Trigger Mode

In this mode, the transfer begins upon detection of a character which meets the qualifications of Registers 5 and 7. Register 5 contains the >, =, or < delimiters and Register 7 contains the trigger value. The computer is interrupted when this character is detected. The data is then transferred until the count is satisfied or until the CPU sends a CED or until a character equal to the delimiter is received. This mode is restricted to 8-bit data transfer only.

Enable Output Inhibit is provided to prevent data output from the interface to the peripheral. The capability is enabled by setting bit 0 of Register 9. The inhibit may be used for 8- and 16-bit output operations. If ST1 or ST0 lines are used in the handshake, this feature is not available. When CTLA/FLGA handshake lines are used, ST0 will act as the inhibit line. When CTLB/FLGB handshake lines are used, ST1 will act as the inhibit line.

This feature is not available in burst output data transfer.

## 8049 Microcomputer (μ)

Most of the activities carried out by the 8049 μC have already been discussed in this appendix. The following discussion will summarize these activies and list all the input and output lines utilized.

The μC is the intermediary between the CPU and the GPIO bus. It implements interface protocol via its own self-contained ROM. The μC responds to instructions from the host CPU.

There are two 8-bit I/O ports. Port 1 (P10 through P17) is connected to the configuration switch S2 switch segments and port 2 (P20 through P26) is used for communicating with the I/O expanders.

There are eight other data bits (D0 through D7) for communication between the μC and the translator.

## Replaceable Parts

This appendix lists the HP 82940A GPIO Interface replaceable parts, and illustrates the GPIO Interface card hardware. The exploded view of figure A-6 shows the part numbers of the major assemblies. Table 4-1 lists the replaceable parts of the interface circuit board assembly.

The total quantity of a part is listed only the first time it is used on the assembly.

The number in the "CD" column, immediately preceding the part number, is the part number's check digit. Please include this number when ordering a part.

**Table A-2. HP 82940A GPIO Interface Replaceable Parts**

| Reference Designator | CD | HP Part No. | TQ | Description |
|---|---|---|---|---|
| A1 | 1 | 82940-60901 | 1 | Circuit Board Assembly, GPIO Interface |
| C1, C2 | 6 | 0180-0228 | 2 | C-F: 22$\mu$fd, 15V |
| C3 – C10 | 8 | 0160-4571 | 9 | C-F: .1$\mu$fd, 50V |
| C11 | 8 | 0167-4767 | 1 | C-F: 20Pfd, 200V |
| C12 | 8 | 0160-4571 | | C-F: .1$\mu$fd, 50 V |
| J1, J2 | 4 | 1251-5266 | 2 | Connector, 24-Pin |
| R1 | 4 | 1810-0278 | 1 | R-F: Network, 9 × 3.3k, 2%, .25W |
| R2 | 8 | 1810-0280 | 2 | R-F: Network, 9 × 10k, 2%, .25W |
| R3 | 2 | 1810-0276 | 3 | R-F: Network, 7 × 1.5K, 5%, .15W |
| R4 | 8 | 1810-0280 | | R-F: Network, 9 × 10k, 2%, .25W |
| R5, R6 | 2 | 1810-0276 | | R-F: Network, 7 × 1.5k, 5%, .15W |
| R7 | 3 | 1810-0368 | 1 | R-F: Network, 5 × 10.0k, 2%, .125W |
| R8 | 2 | 1810-0367 | 1 | R- F: Network, 5 × 4.7k, 2%, .125W |
| S1 | | 3101-2533 | 1 | Switch: 4 Segment, SPDT |
| S2 | | 3101-2534 | 1 | Switch: 8 Segment, SPDT |
| U1 | 3 | IMB5-0101 | 1 | IC: Translator |
| U2 | 7 | 1820-2440 | 1 | IC: 8049 Microcomputer |
| U3 – U5 | 7 | 1820-2177 | 3 | IC: 8243 I/O Expander |
| U6 – U8 | 0 | 1820-2138 | 3 | IC: DS887IN Driver |
| U9 | 8 | 1820-1112 | 1 | IC: 74LS74 |
| Y1 | 1 | 0410-1222 | 1 | Crystal: 11 MHz |
| | | 8120-3190 | 1 | Cable Assembly |

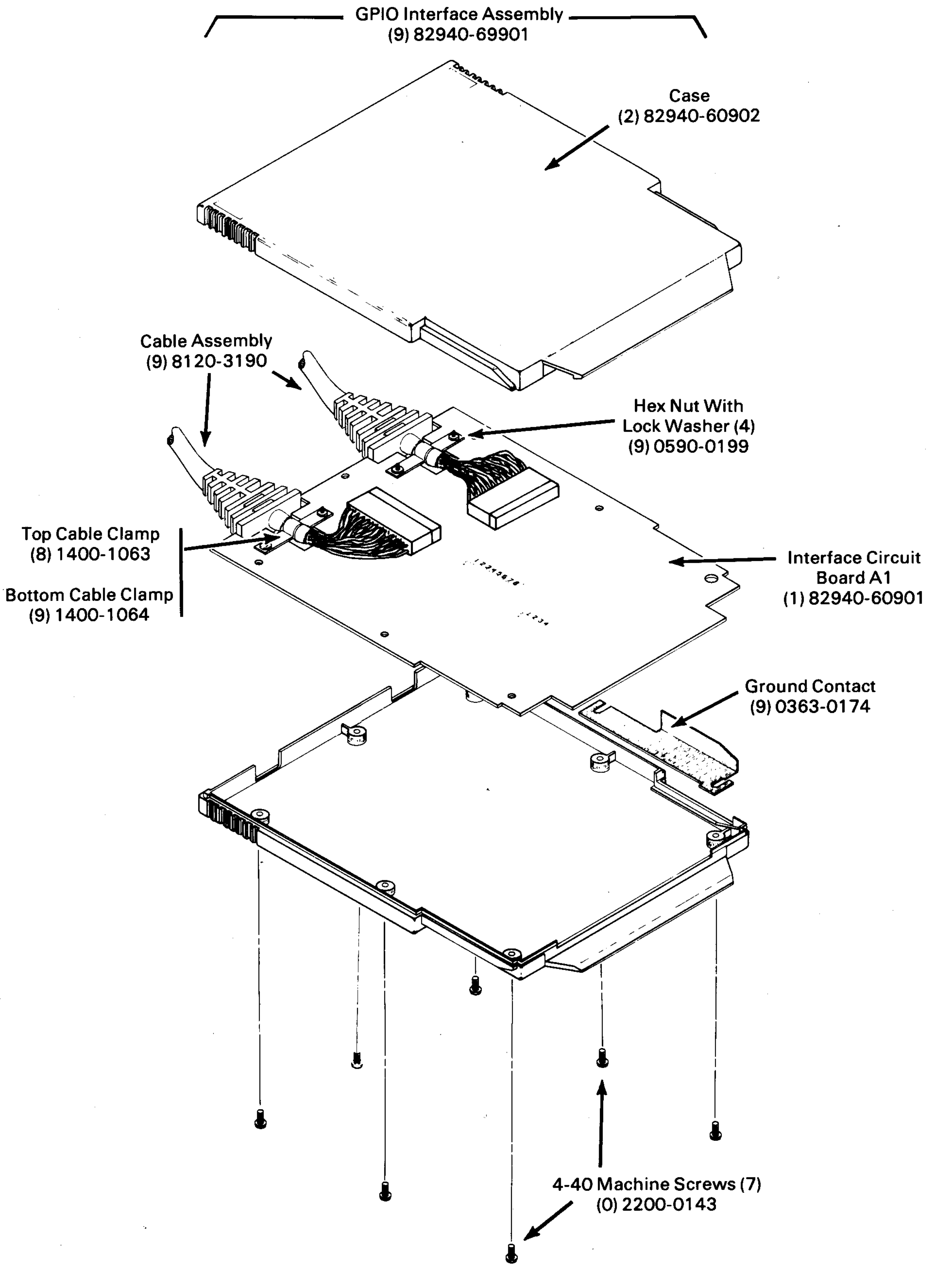
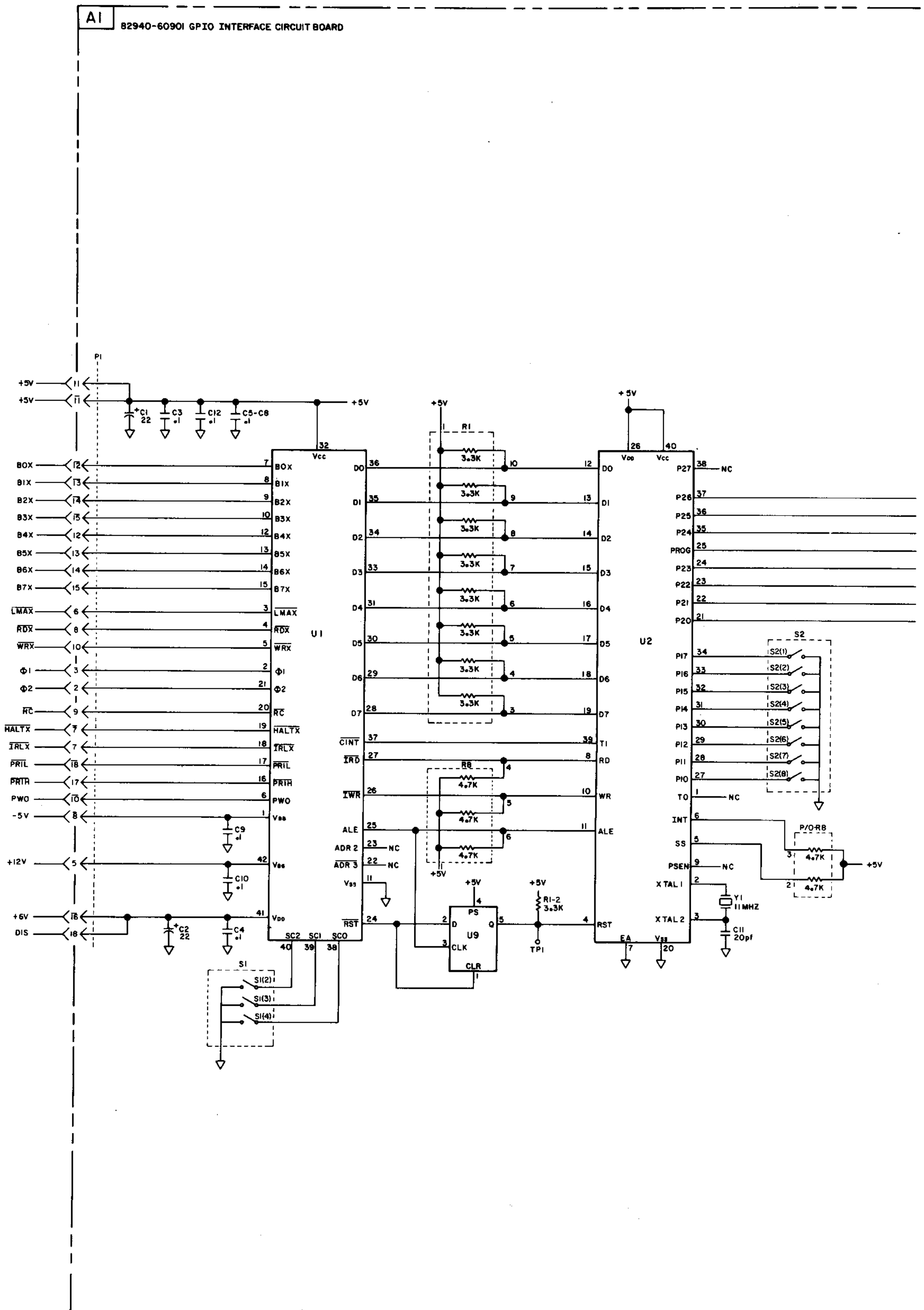


**Figure A-6. GPIO Exploded View**


A1
82940-60901 GPIO INTERFACE CIRCUIT BOARD
U1
U2
U9
R1
R8
S1
S2
P/O R8
Y1 11MHZ
TP1
+5V
+12V
-5V
+6V
DIS

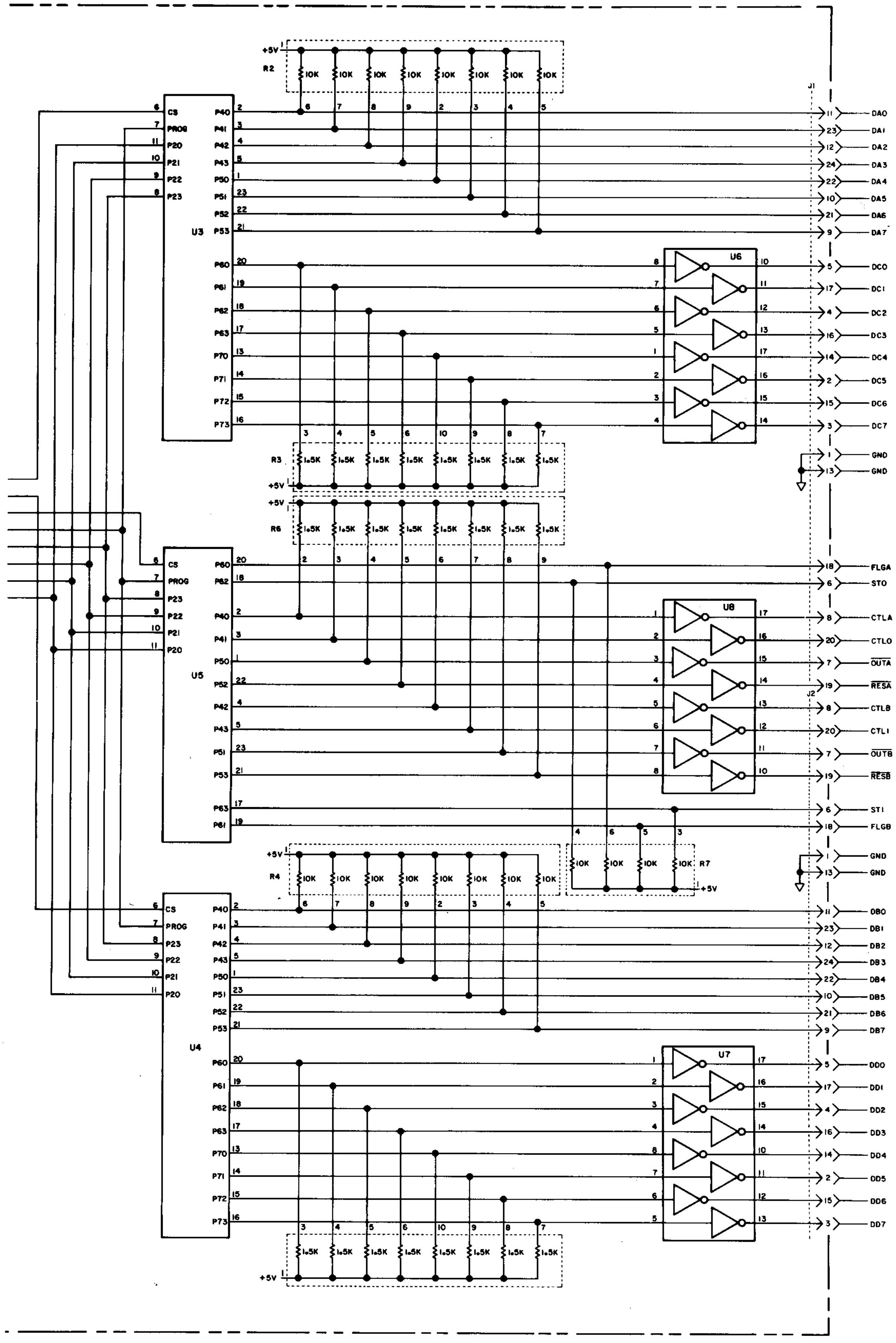


**Figure 3-10. Interface Schematic Diagram**

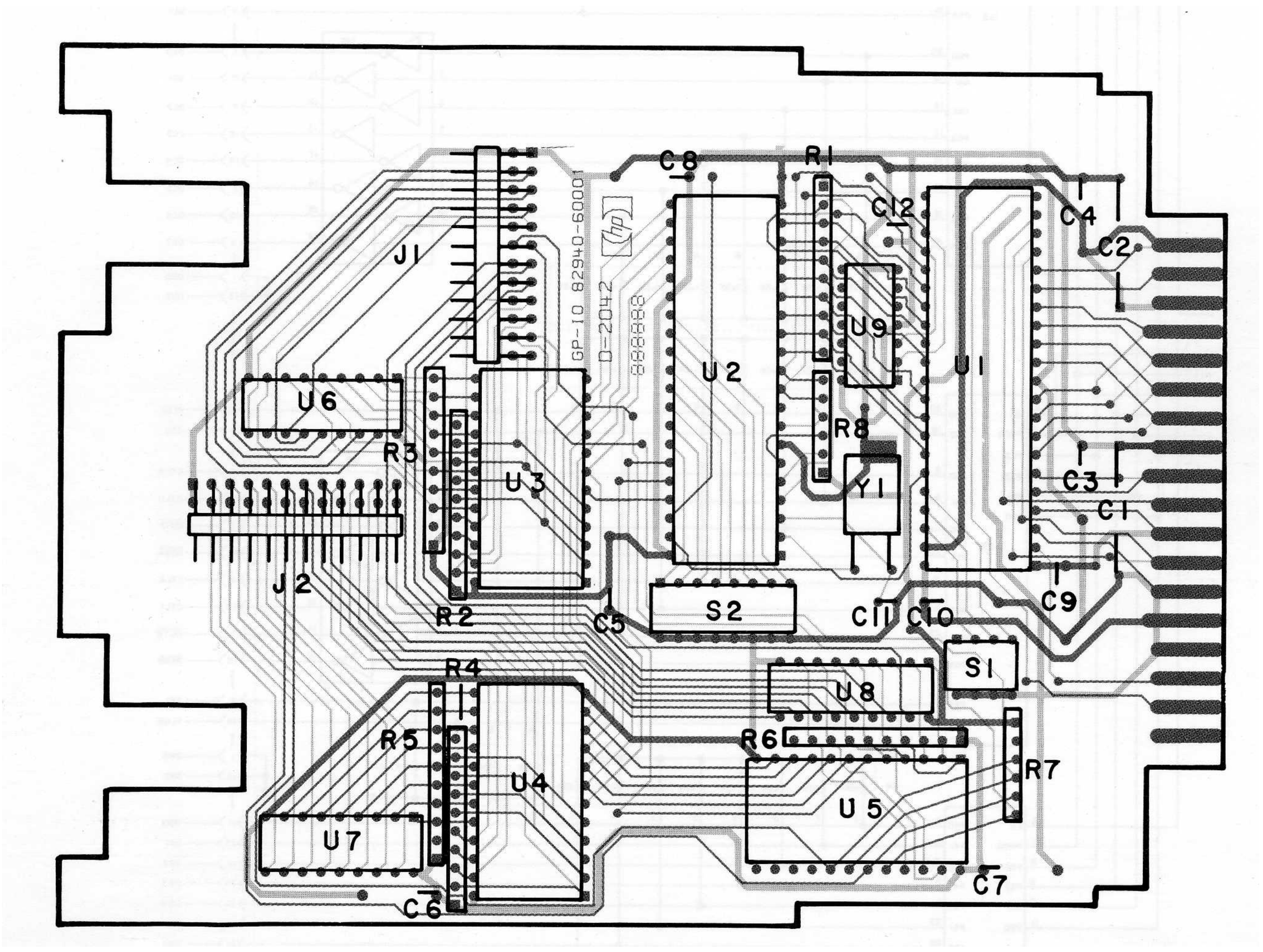


Figure A-8. Interface Component Location Diagram

Appendix B

# Register Assignments

The GPIO interface contains 11 registers which are accessible by means of (write) CONTROL. Nine registers are accessible by means of (read) STATUS. The bit assignments and their mnemonics are shown below.

The following is a description of the control registers and their bit assignments.

**Register 0 (CONTROL)**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| X | X | X | X | Odd Parity | Even Parity | One Parity | Zero Parity |

If bits 0 through 3 are zero, then no parity
X = Don't care

**Register 0 (STATUS) Interface I.D.**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |

Bit 2 always = 1 (I.D. code = 04)

**Register 1 (CONTROL)**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ST1 | ST0 | FLGB | FLGA | X | X | Received Data Parity Error | X |

FLGA = Flag A interrupt mask (1 = enabled)
FLGB = Flag B interrupt mask (1 = enabled)
ST0 = Status Bit 0 interrupt mask (1 = enabled)
ST1 = Status Bit 1 interrupt mask (1 = enabled)
X = Don't care
Received Data Parity Error Interrupt Mask (1 = enabled)

### Register 1 (STATUS)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ST1 | ST0 | FLGB | FLGA | 0 | 0 | Received Data Parity Error | 0 |

FLGA = Flag A interrupt
FLGB = Flag B interrupt
ST0 = Status bit 0 interrupt
ST1 = Status bit 1 interrupt
Received Data Parity Error Interrupt

### Register 2 (CONTROL)
### Register 28 (ASSERT)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| RESETB | RESETA | X | X | CTL1 | CTLB | CTL0 | CTLA |

CTLA = Control bit A before normalization (1 = true)
CTL0 = Control bit 0 before normalization (1 = true)
CTLB = Control bit B before normalization (1 = true)
CTL1 = Control bit 1 before normalization (1 = true)
A high to low transition on the RESETA or RESETB line will reset the peripheral device.
X = Don't care

### Register 2 (STATUS)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ST1 | ST0 | FLGB | FLGA | CTL1 | CTLB | CTL0 | CTLA |

CTLA = Control bit A before normalization (1 = true)
CTL0 = Control bit 0 before normalization (1 = true)
CTLB = Control bit B before normalization (1 = true)
CTL1 = Control bit 1 before normalization (1 = true)
FLGA = Flag A after normalization (1 = true)
FLGB = Flag B after normalization (1 = true)
ST0 = Status bit 0 after normalization (1 = true)
ST1 = Status bit 1 after normalization (1 = true)

### Register 3 (STATUS/CONTROL) Normalization

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| ST1 | ST0 | FLGB | FLGA | CTL1 | CTLB | CTL0 | CTLA |

Configuration Switch 7 Configuration Switch 8

CTLA = Control bit A (0 = positive-true)
CTL0 = Control bit 0 (0 = positive-true)
CTLB = Control bit B (0 = positive-true)
CTL1 = Control bit 1 (0 = positive-true)
FLGA = Flag A (0 = positive-true)
FLGB = Flag B (0 = positive-true)
ST0 = Status bit 0 (0 = positive-true)
ST1 = Status bit 1 (0 = positive-true)

**Register 4 (STATUS/CONTROL)**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| Handshake Mode | Handshake Mode | Flag Transition | 1 | Port D | Port C | Port B | Port A |

Configuration Switch 5 (bits 7–5) — Always 1 (bit 4) — Configuration Switch 6 (bits 3–0)

Port A data normalization (0 = positive-true)
Port B data normalization (0 = positive-true)
Port C data normalization (0 = positive-true)
Port D data normalization (0 = positive-true)
Flag transition 0 = Accept on Ready to Busy (default power on)
1 = Accept on Busy to Ready

| Handshake mode | Bit 7 | Bit 6 | | |
|---|---|---|---|---|
| | 0 | 0 | = | Full handshake |
| | 0 | 1 | = | Partial handshake |
| | 1 | 0 | = | Strobe handshake |
| | 1 | 1 | = | No handshake |

**Register 5 (STATUS/CONTROL)**

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| > DELIM | = DELIM | < DELIM | CTL Enable | Device Address | Device Address | Device Address | Device Address |

Device address = Bits 1, 2, 3 and 4 (configuration switches 1, 2 and 3 respectively)
< DELIM = Less than register 7 contents (delimiter)
= DELIM = Equal to register 7 contents (delimiter)
> DELIM = Greater than register 7 contents (delimiter)
CTL enable = Send CTL upon TRIGGER

**Register 6 (STATUS/CONTROL)**

| 7 | 6 5 4 3 2 1 0 |
|---|---|
| Resolution Increment | Strobe Pulse Duration |

Strobe pulse duration = Delay in number of resolution increments
Resolution increment = 1 = 1 millisecond, 0 = 10 microseconds

**Register 7 (STATUS/CONTROL)**

| 7 6 5 4 3 2 1 0 |
|---|
| Trigger Value |

Bits 0 through 7 are used in conjunction with register 5

## Register 8 (STATUS/CONTROL)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| X | X | X | X | X | X | Output Enable B | Output Enable A |

Output enable A = Enable output on port A
Output enable B = Enable output on port B

**Note:** CONTROL allowed only if switch S2 segment 4 is set to 1. If not set to 1, an error will result.

## Register 9 (STATUS/CONTROL) Enable Output Inhibit

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| X | X | X | X | X | X | X | Inhibit |

Inhibit — Inhibits output of data (1 = enabled)

## Register 16 (Write Only)

| 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|
| EOL | | | | | | | |

EOL = Number of End of Line characters

## Registers 17 Through 23 (Write Only)

EOL Character

# GPIO (I/O ROM) Statements

The following statements are available in the I/O ROM for use with the HP 82940A GPIO Interface:

| Statement | Description |
|---|---|
| ABORTIO | Aborts any interrupt transfer in progress, sets all CTL lines to the false state, places ports A and B in the high-impedance state, and sets ports C and D to the off state. |
| ASSERT | Immediately writes a value to Control Register 2, placing the CTL and RES lines in the specified states. |
| CLEAR | Pulses RES line(s). CLEAR 4 pulses both RESA and RESB. CLEAR 400 pulses RESA; CLEAR 401 pulses RESB (assuming that 4 is the interface select code). |
| CONTROL | Writes values to the interface control registers. |
| ENABLE INTR | Writes enable mask to control register 1. Used to select event interrupts. |
| ENTER | Enters data from a port to the BASIC program. |
| HALT | Stops any interrupt transfer in progress, leaving all handshake lines in their current state. Thus, STATUS can be used to troubleshoot a faulty handshake sequence. |
| OUTPUT | Outputs data from the BASIC program to a port. |
| RESET | Places interface in the power-on state. CTL lines are false, OUTA and OUTB indicate output, ports A and B are put in high-impedance state, ports C and D are off. |
| SEND | SEND...CMD can set primary address or pulse RES line(s) with Device Clear. SEND...DATA outputs data bytes and EOL if specified. SEND...LISTEN or SEND...TALK can set primary address. UNL, UNT, MLA, MTA are ignored; SCG generates error 111. |
| STATUS | Reads values from interface status registers. |
| TRANSFER | Moves data from a port to a buffer or from a buffer to a port. Interrupt or fast handshake may be used. |

Appendix D

# GPIO Interface Errors

The following errors are specific to GPIO Interface operations:

| Error No. | Meaning | Possible Cause |
|---|---|---|
| 110 | I/O card failed self test. | Cord not completely plugged in, or faulty. |
| 111 | Illegal operation. | Attempted operation is not used with this interface. |
| 113 | Word cut in half during a 16-bit operation. | Odd byte count specified in an image or COUNT parameter. Also beware of odd-length buffers and free-field format. |
| 114 | FHS transfer was aborted by STO. | True state on STO. See FHS and INTR Transfers. |
| 115 | Output to Port A or Port B is not allowed. | Using the wrong primary address in an output operation. Not enabling the proper bit in Register 8. Attempting to write to Register 8 when switch 4 is not properly set. Switch 4 is explained in the Installation section of this manual. |
| 116 | CTL line not in proper state to start handshake. | A previous ASSERT or CONTROL statement that left CTL in the true state. CTL must be in the false state at the start of a handshake operation. |

HEWLETT
PACKARD

Personal Computer Division
1010 N.E. Circle Blvd., Corvallis, OR 97330 U.S.A

Reorder Number
82940-90011

Printed in U.S.A. 9/82
82940-90014