Section 15

Using the GPIO Interface

Section Introduction

The 82940A interface allows your HP-85 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 overwelmed. 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 82940A 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 isolate 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, a technician or "hardware type" is practically a necessity during the installation of a parallel interface because there aren't any connectors wired to the 82940A when you receive it. Most of the hardware information is presented in the 82940A Interface Installation and Theory of Operation manual. Please refer to that manual for hardware details such as:

- How to set the interface select code
- How to set the default configuration switches
- How to connect the interface cables
- Recommended driver and receiver circuits

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

HP Computer Museum
www.hpmuseum.net

For research and education purposes only.

Essentials of a Parallel Interface

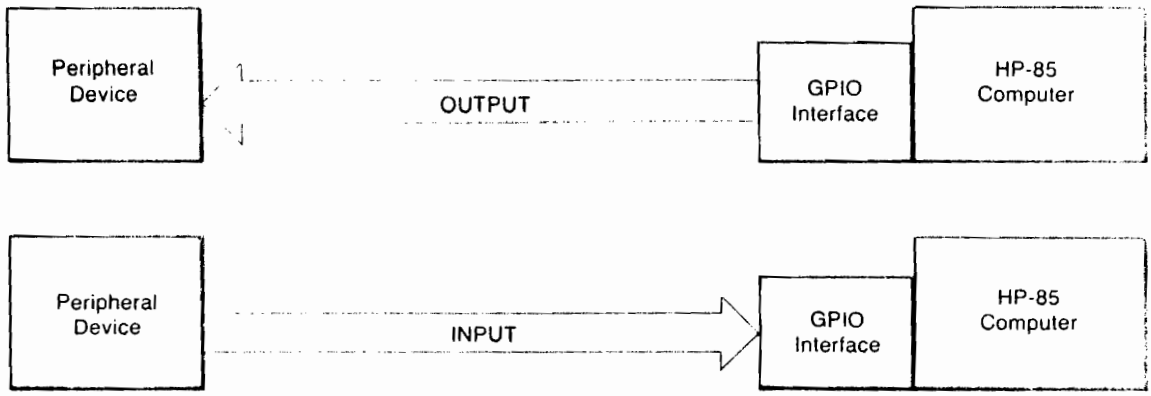
The Section Introduction recommended that you consider each interface characteristic individually whenever possible. What are these essential characteristics? In most cases, a parallel interface will be successful if each of the following characteristics has been properly determined:

- 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 2 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.

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 similiar 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. This is two bidirectional ports with small output drive and two output-only ports. Note that other configurations yield less 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 accessed 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 (refer to the Installation and Theory of Operation manual). 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 indicates which lines are used for handshake and direction indication with 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.

8-Bit Ports

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

16-Bit Ports

Data Direction	Primary Address	Port Description	Handshake Lines	Direction Indicator
Bidirectional; small output drive	08	LSB[1] on Port A MSB[1] on Port B	CTA/FLGA	$\overline{\text{OUTA}}$
	09	same	CTLB/FLGB	$\overline{\text{OUTB}}$
Input and output on separate lines; large output drive	10	LSB input on Port A MSB input on Port B LSB output on Port C MSB output on Port D	CTLA/FLGA	$\overline{\text{OUTA}}$
	11	same	CTLB/FLGB	$\overline{\text{OUTB}}$
Output only; large output drive	12	LSB on Port C 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) Port B and Port D wired together (MSB)	CTLA/FLGA	$\overline{\text{OUTA}}$
	15	same	CTLB/FLGB	$\overline{\text{OUTB}}$

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 Section 1. With the GPIO interface, you have four basic methods of handshake to choose from (with some variations, of course). These basic choices are:

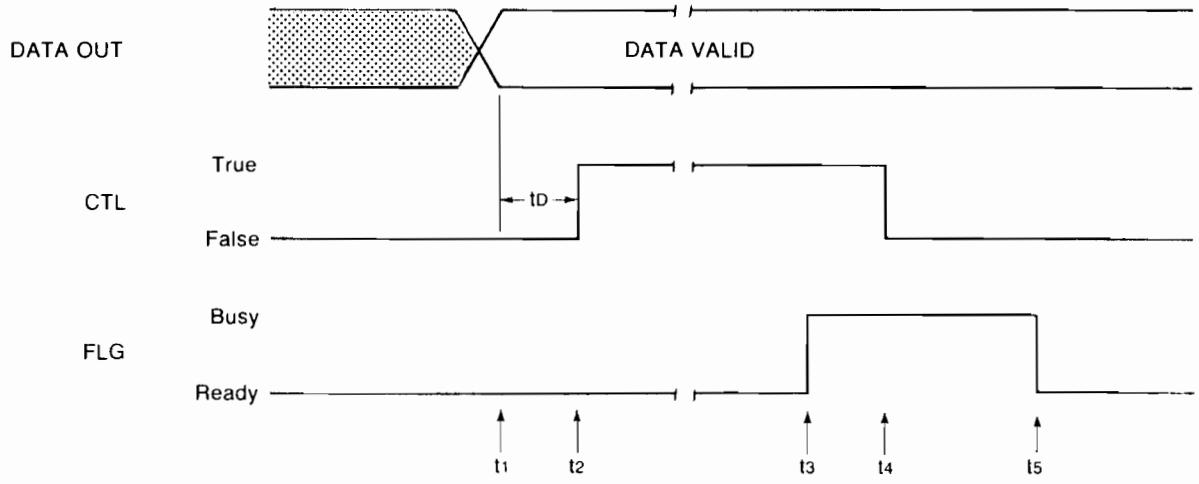
- Full handshake
- Partial handshake
- Strobe handshake
- No handshake

The handshake lines on the 82940A 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.

[1] As it is used here, the abbreviation "LSB" stands for "Least Significant Bits". These are bit 0 thru bit 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.

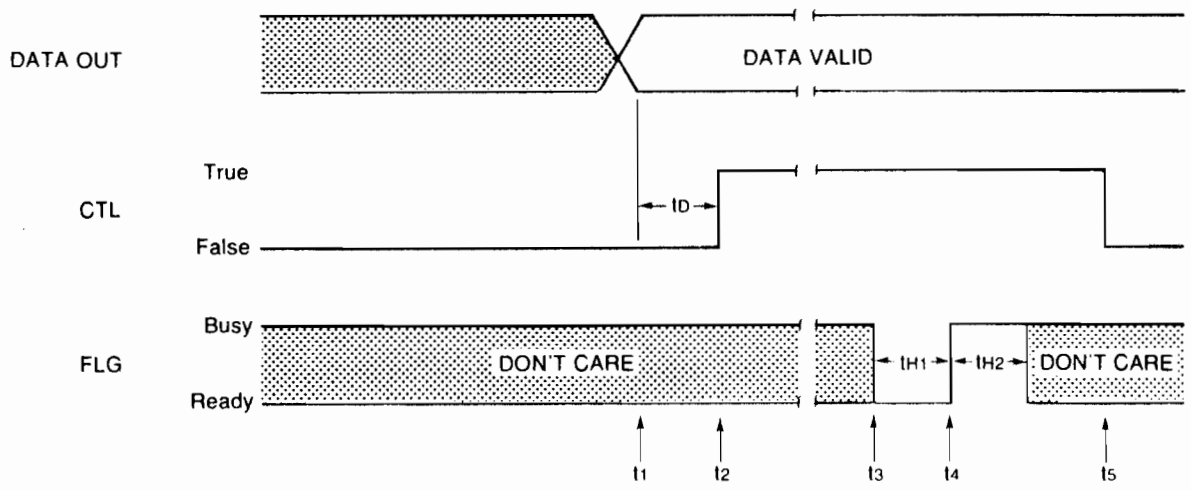
Output Handshakes

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 the Installation and Theory of Operation manual. 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.



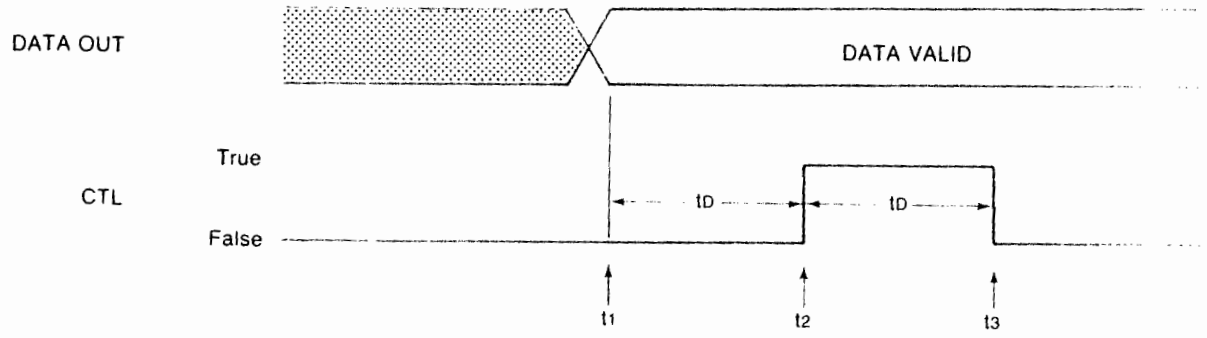
Output: Full Handshake

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 the 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).



Output: Partial Handshake

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).



Output: Strobe Handshake

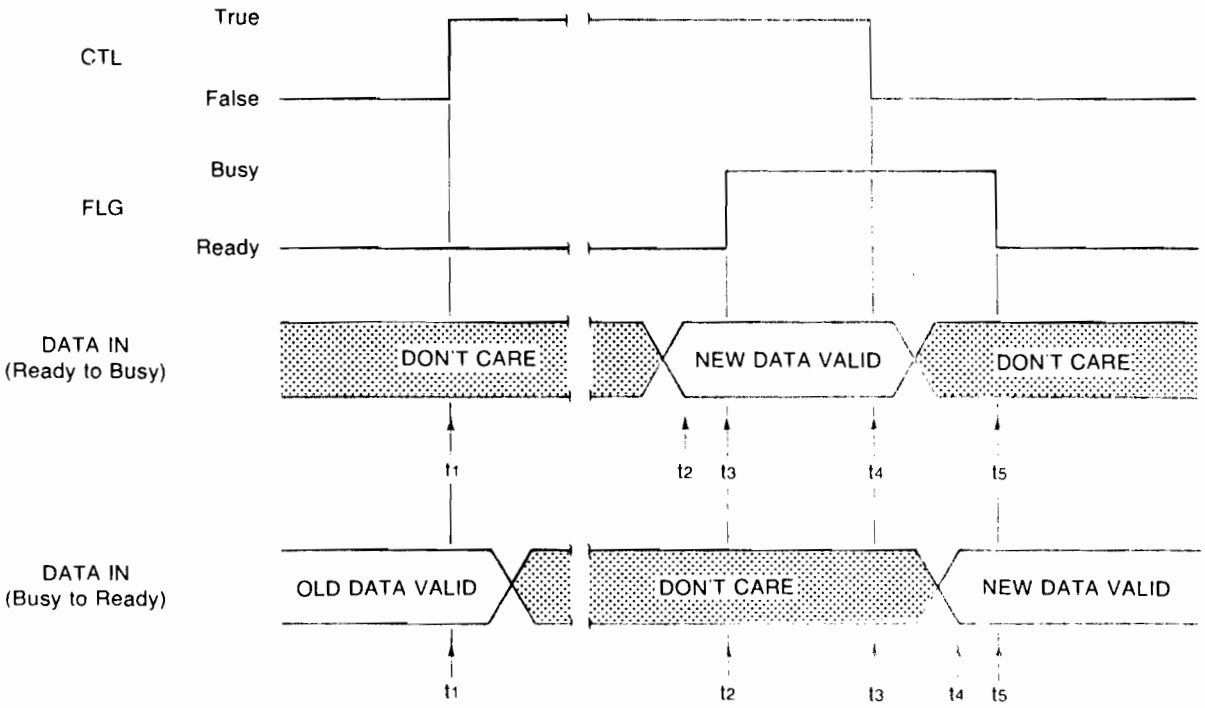
The strobe handshake for output is a very simple sequence. It is probably the most common handshake method used in devices that do not implement full handshake. This method assumes that the peripheral device is always ready and the FLG line is not used. (If your device is not always ready, then the "Output 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.

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".[1] 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.

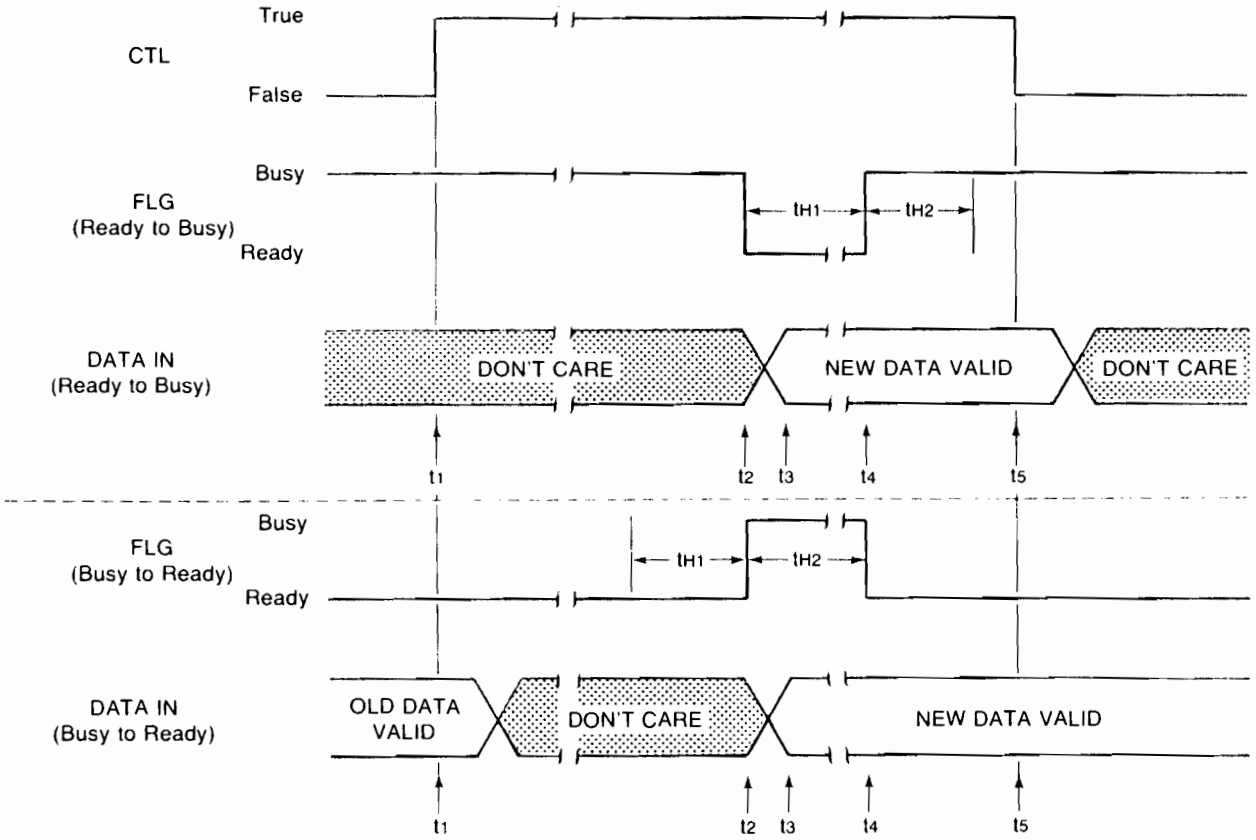
[1] These names were derived from the state change on the FLG line that triggers the read operation in full handshake mode. However, the terms are not meant to imply that FLG lines are edge-triggered. They are not. Also, these names are used to describe the timing choices for all input handshakes, even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



Input: Full Handshake

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).

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).

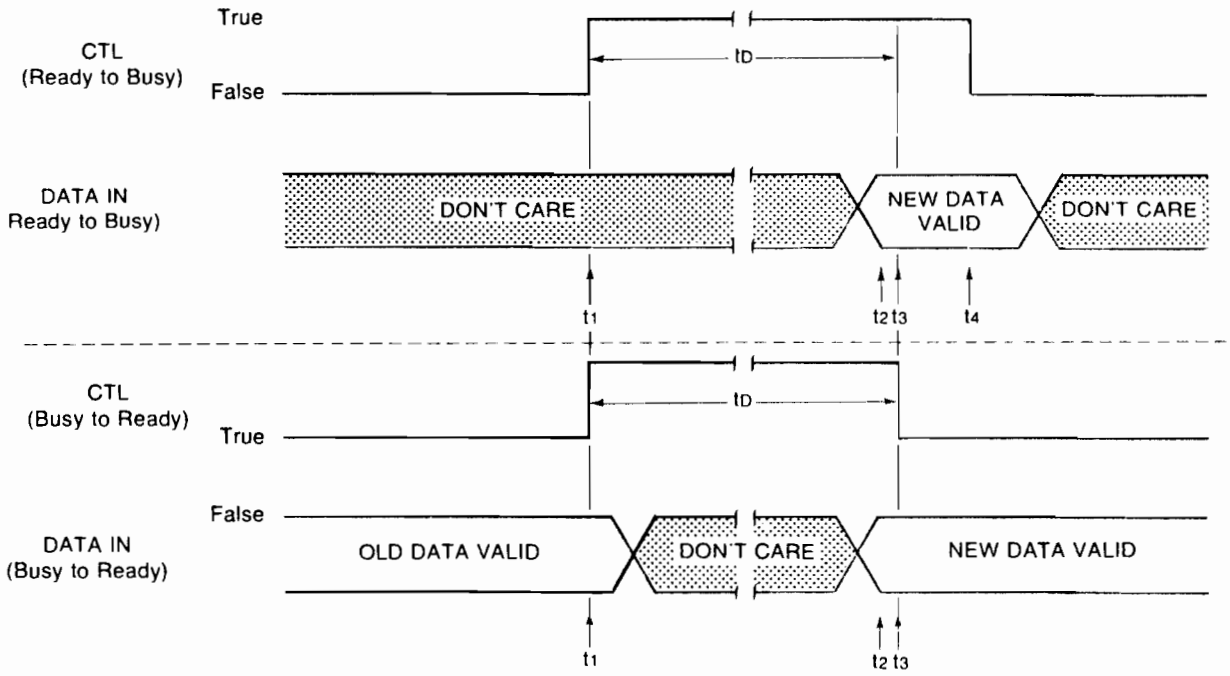


Input: Partial Handshake

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 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). The interface then indicates receipt of the data by returning CTL to the FALSE state (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").



Input: Strobe Handshake

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).

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 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 "Slecting 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:

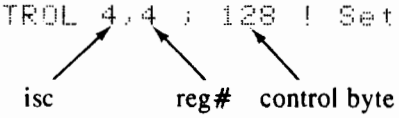
Bit 5	Data Input Timing
0	READY to BUSY
1	BUSY to READY

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
```




```
CONTROL 4,4 ; 96 ! Partial hndsk, 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 hndshk 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 establish an **additional** delay time which is added to a minimum time that is always present. The minimum times are generally around 60μs, but there are exceptions. Refer to the Installation and Theory of Operation manual 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 = 10μs 1 = 1ms	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 times 10 μs to determine the additional delay time. When bit 7 is set, the value of the other bits is multiplied times 1 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, so 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 use Output Inhibit
```

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" means 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 ST0	Invert FLGB	Invert FLGA	Invert CTL1	Invert CTLB	Invert CTL0	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 Logic LO = READY
	1	Logic HI = READY Logic LO = BUSY
CTL	0	Logic HI = TRUE Logic LO = FALSE
	1	Logic HI = FALSE 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 statement. 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 hndsk; 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, set 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 singe port can be changed without effecting 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
```

Why Won't This Thing 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 results that will only be resolved by the death of one set of drivers. 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.

Although it is impossible to completely prevent human error, 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 and Theory of Operation manual. Trying to write to register 8 without first setting this switch results in Error 115.