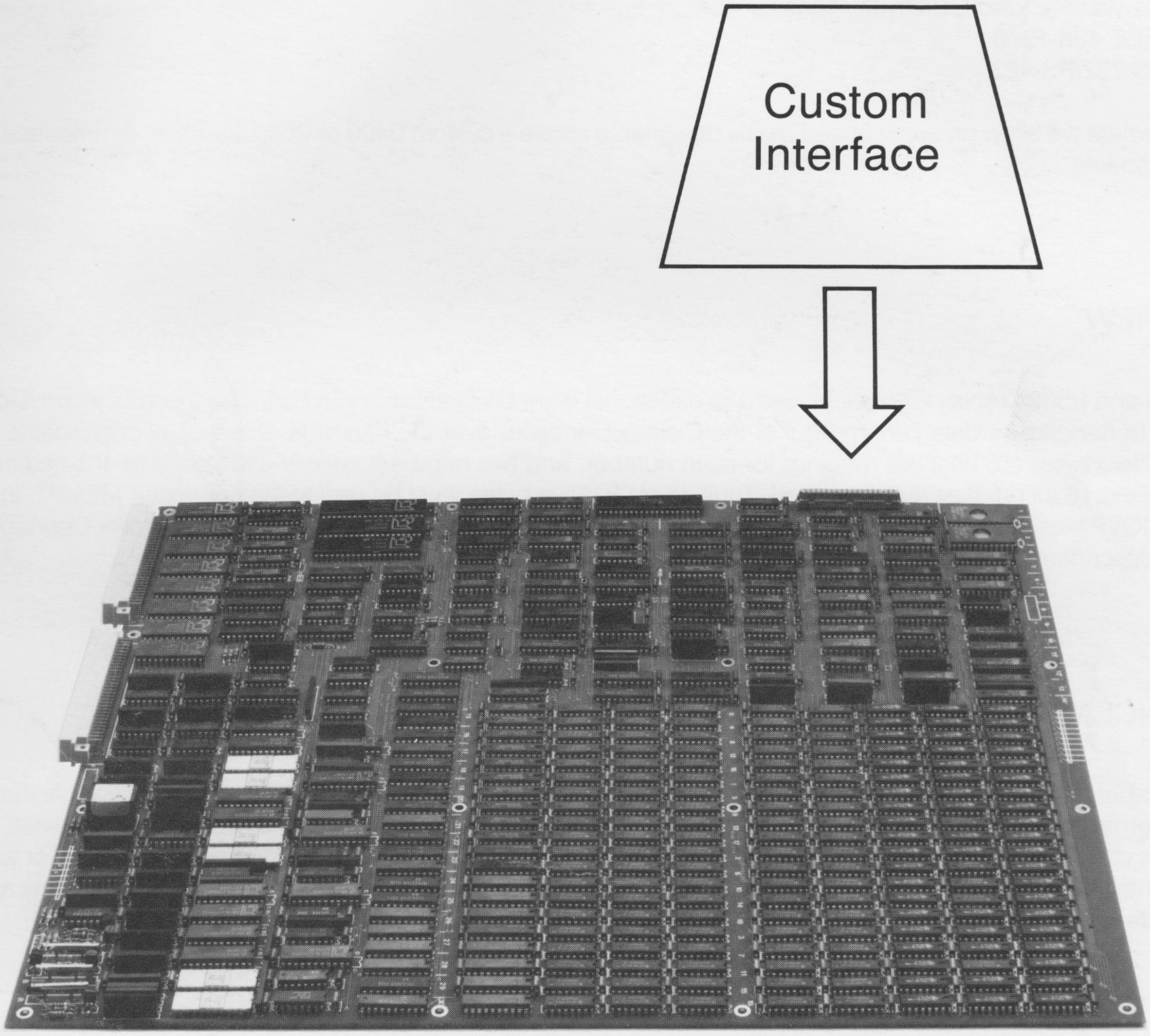
DESIGN
DOCUMENT

METHEUS



Ω400/Ω500 Display Controller
Custom Interfacing

ABSTRACT

This paper describes the METHEUS Ω400 and Ω500 I/O interface. By adding custom interface cards, it is possible to tach the Ω400 or Ω500 Display Controller to virtually any computer. A variety of interfaces are available from Metheus Corpo tion, including:

- Digital Equipment Corp. PDP*-11
- Digital Equipment Corp. VAX*
- Hewlett-Packard GPIO
- IEEE 488-1978
- RS-232/RS-423

This document will allow an experienced digital designer to create a custom Ω400 or Ω500 interface. An interface for a Zilog Z80** is shown.

OVERVIEW

The Ω400 and Ω500 Display Controllers use a bi-directional eight bit data bus for I/O with the external world. Control lines are used to handshake data into and out of the Display Controller over the data bus. Since pixel coordinates use 11 bit numbers, two bytes (16 bits) are required for each number, and two numbers specify each point on the screen. The top five bits of the 16 bit number are discarded. Thus, to set P1, five bytes must be sent to the controller: MOVP1, lox, hix, loy, hiy. The MOVP1 instruction is encoded as an ASCII 'R' (decimal 82). See the appropriate Display Controller Operator's Manual for a full description of the instruction set.

NOMENCLATURE

For this paper, **controller** is the METHEUS Display Controller. The **interface** is the I/O card that interfaces the controller to the external world. **Input** is defined as sending data to the Display Controller from the host device. Likewise, **output** is defined as data sent from the Display Controller to the host device. Signal names are ended with either an **H** or an **L**, telling whether they are active high (> 2 volts) or active low (< .8 volts). Thus, GATEH is active when high. The terms **assert** and **negate** are used to indicate that a signal is set to its active or inactive states, respectively.

INPUT

Six control signals and the data bus are used for data input. The control signals used to handshake data into the controller are:

IDATSTBH - Input Data Strobe, rising edge active.
IDATACCH - Input Data Accepted, active high.
IDATRDYH - Input Data Ready, active high.
CLIDRDYH - Clear Input Data Ready, active high.
IDATLRDENL - Input Data (Low Byte) Read Enable, active low.
PROCCLKH - Processor Clock, rising edge active.

---

* PDP and VAX are trademarks of Digital Equipment Corporation.

** Z80 is a registered trademark of Zilog Inc.

The interface drives the IDATSTBH and CLIDRDYH handshake lines, and the controller drives the rest. The data bus is active high. Figure 1 is a timing diagram for a single byte data input sequence. The rising edge of IDATSTBH begins the data in sequence. This rising edge causes the controller to:

1) Negate IDATACCH, showing that the input data has not yet been accepted by the Display Controller, and
2) After synchronizing the request with the bit-slice, assert IDATRDYH, showing that input data is ready for the controller.

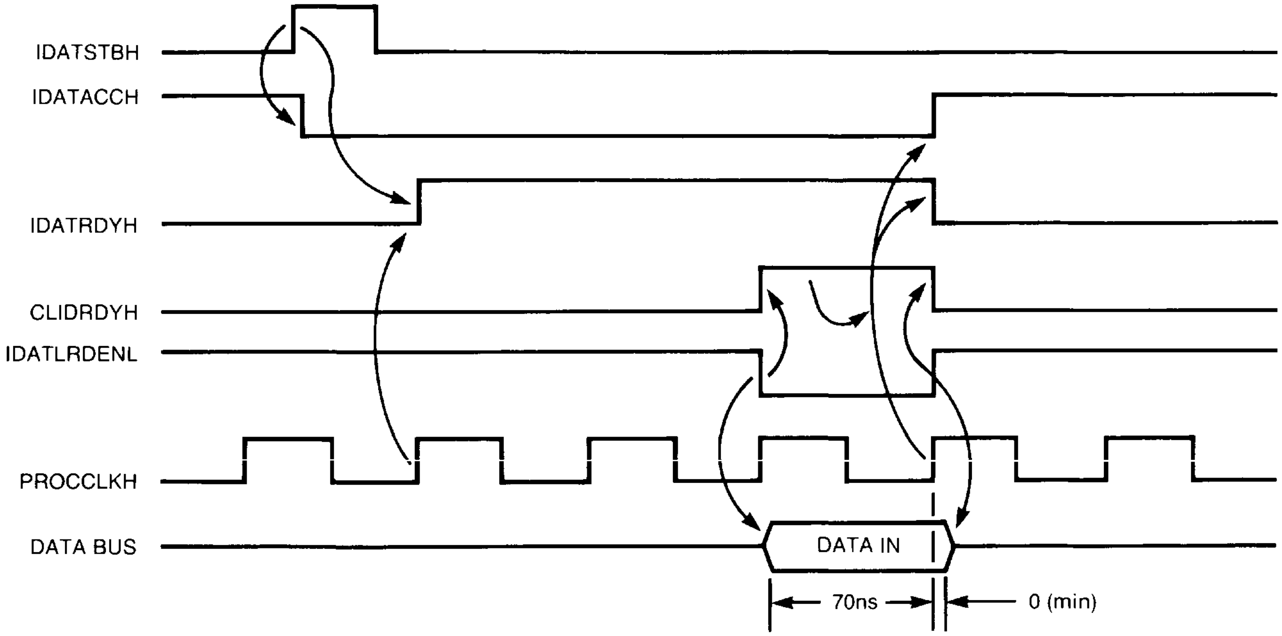


Figure 1. Single Byte Input

When the controller is ready to read the input data, it asserts IDATLRDENL. Only while this signal is asserted may the interface place its data onto the data bus by enabling a tri-state buffer. The controller latches the data internally on the rising edge of PROCCLKH. This happens slightly before the rising edge of IDATLRDENL. When the controller negates IDATLRDENL, the interface must release the data bus by tri-stating its bus drivers. If no more data is immediately available for input, CLIDRDYH is used to negate IDATRDYH to the Display Controller. This signal should be asserted synchronously with IDATLRDENL. If more data is available to be transferred, the interface need not assert CLIDRDYH. This keeps IDATRDYH asserted telling the Display Controller that more input data is available. Figure 2 illustrates a two byte data input sequence. When the last byte of data has been read by the controller, the interface should assert CLIDRDYH as soon as IDATLRDENL is asserted by the controller. This stops data input cycles until the interface again asserts IDATSTBH.

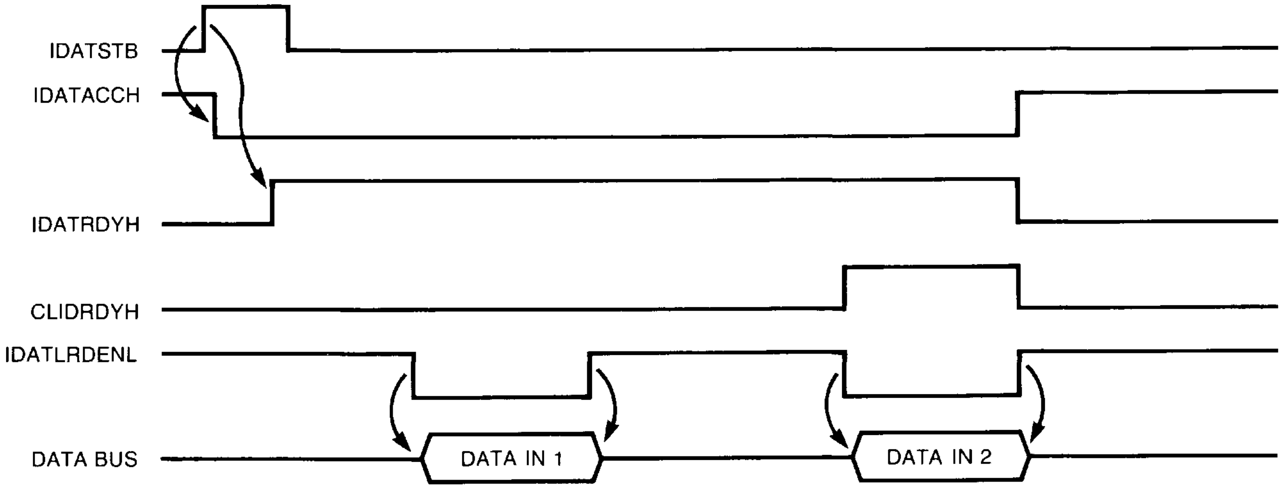


Figure 2. Two Byte Input

OUTPUT

Five control signals and the data bus are used for data output cycles. The control signals used for handshaking are:

ODATSTBH - Output Data Strobe, rising edge active.
SODRDYH - Set Output Data Ready, active high.
ODATLLDENL - Output Data (Low Byte) Load Enable, active low.
ODATRDYH - Output Data Ready, active high.
PROCCLKH - Processor Clock, rising edge active.

The interface drives the SODRDYH and ODATSTBH handshake lines, and the controller drives the remainder. Figure 3 is the timing diagram for the data output sequence. To start an output cycle, the controller asserts ODATLLDENL and places the output data on the data bus. The interface must latch the data on the next rising edge of PROCCLKH. A 74ls377 is a part well suited for this. After the rising edge of PROCCLKH, the controller negates ODATLLDENL. If the interface can only store this data byte, it should assert SODRDYH as soon as ODATLLDENL is asserted by the controller. If the interface does not assert SODRDYH, the controller will start next output cycle at its leisure. See figure 4 for a two byte data output cycle. Asserting SODRDYH causes ODATRDYH to be asserted by the controller. When the interface has consumed the data byte and is ready for the next one, it should assert ODATSTBH. The rising edge of this signal negates ODATRDYH and sets the controller up for the next ouput cycle.

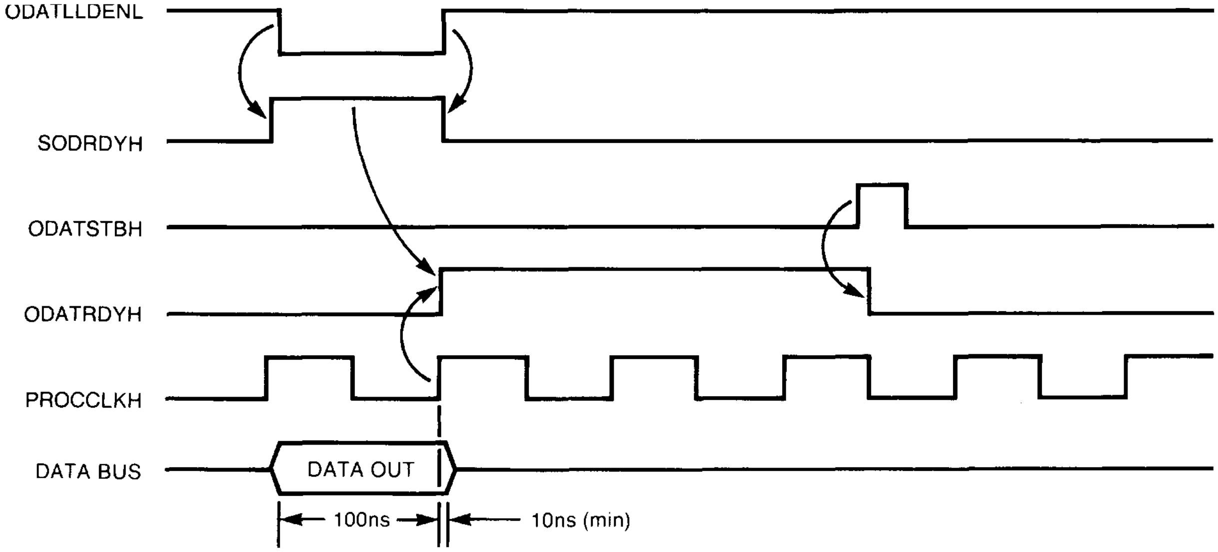


Figure 3. Single Byte Output

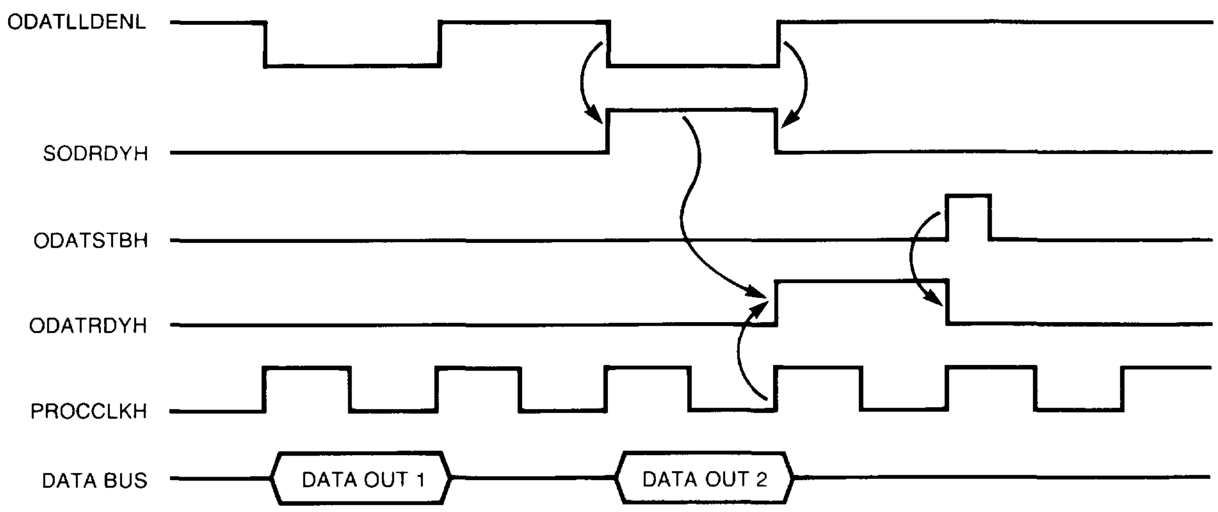


Figure 4. Two Byte Output

INITIALIZATION

The interface generates the controller reset signal, INIT1L. This signal must be asserted for at least 10 microseconds to reset the controller. The controller conditions INIT1L through a Schmitt trigger and returns the clean signal to the interface as INITL. This allows the interface to use either an RC network or a TTL reset signal from a host computer to reset the controller. After the interface negates INIT1L, the controller negates INITL and begins microcode diagnostics. To test and reset the interface, ODATHLDENL is asserted for a minimum of 256 clock cycles. While ODATHLDENL is asserted, the state of all other interface control lines is undefined. After negating ODATHLDENL, the controller tests the interface by outputting data bytes until the interface asserts SODRDYH (see **OUTPUT** section above). The controller then waits until the interface begins an input cycle by asserting IDATSTBH. IDATSTBH may be asserted any time after ODATHLDENL is negated (but is not tested by the controller until after the test data has been output). The controller then performs input cycles until the interface asserts CLIDRDYH (see **INPUT** section above). The controller assumes that the interface is defective if:

1) The data bytes read back do not match those written, or
2) The number of bytes read does not equal the number written.

The controller will not operate with a defective interface, and will light the appropriate status LEDs to indicate a bad interface card.

HINTS

Since the control signals IDATLRDENL and ODATLLDENL are generated by decoders, they may glitch following the rising edge of PROCCLKH. Because of this, all control signals are only guaranteed valid 50nsec after the rising edge of PROCCLKH. Any logic that may be sensitive to these glitches should qualify these signals with PROCCLKH. For example, don't run ODATHLDENL directly into the reset pin of a flip-flop.

PROCCKL has a maximum frequency of *about* 5 to 7MHz. This frequency is *not* stable and changes (radically) during normal operation. The 5MHz is given as a ballpark figure to help select logic families for the interface.

Since IDATSTBH and ODATSTBH are rising edge triggered, they may be held either high or low at the design's convenience.

Z80 INTERFACE

Figure 5 shows a simple Z80 interface. This interface implements the single byte data transfers illustrated in figures 1 and 3. In this application, the AMD 2950 I/O port performs virtually all the interfacing between the Z80 and the Display Controller. Familiarity with the 2950 is assumed. An I/O decoder interprets the Z80 address and control lines to produce three I/O control signals:

CRDL - Controller Read, active low.
CWRL - Controller Write, active low.
SRDL - Status Read, active low.

For simplicity, RAM and ROM are omitted from the diagram.

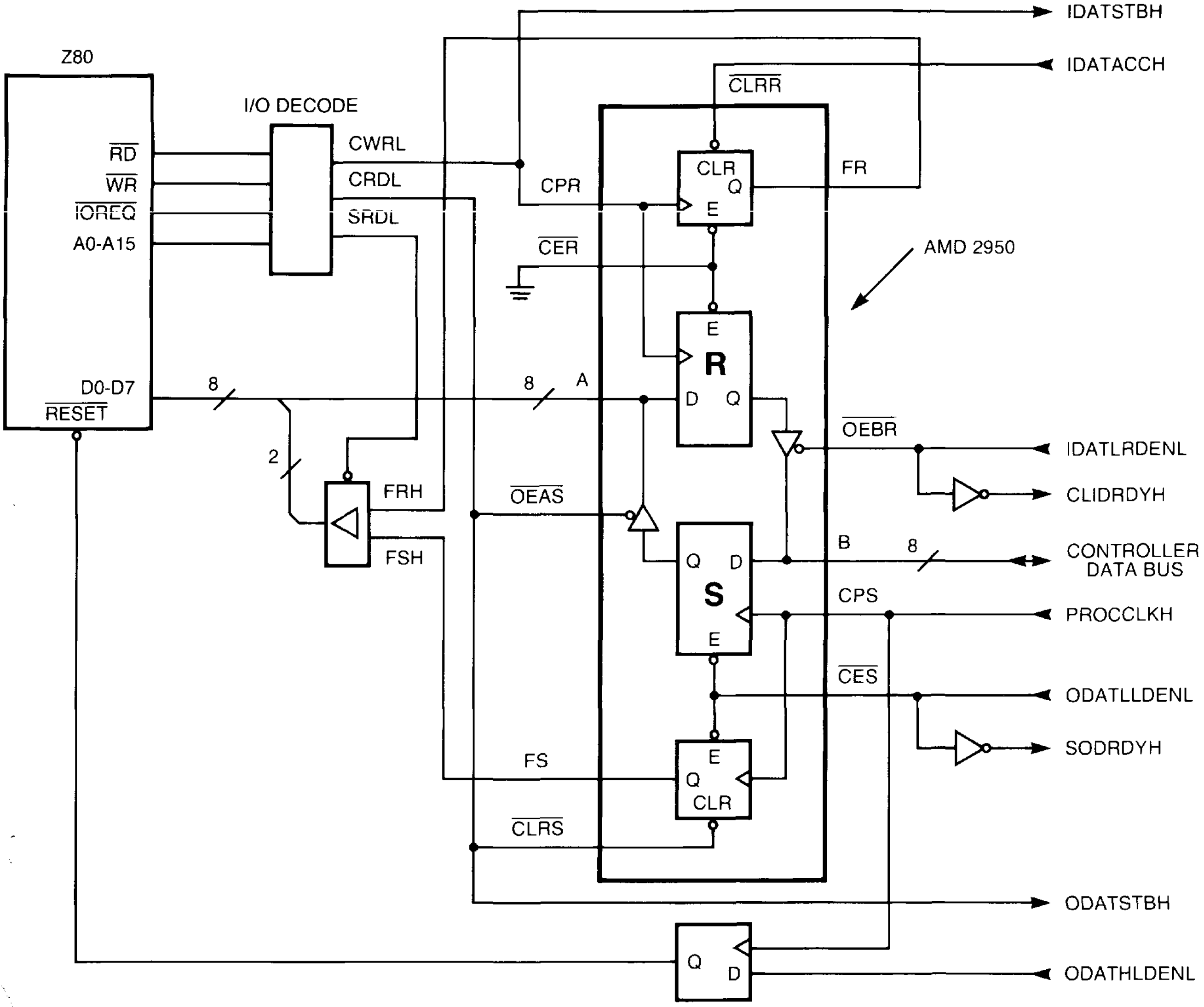


Figure 5. Z80 Interface

Initialization. When the controller asserts ODATHLDENL, a flip-flop deglitches it and asserts the Z80 reset line. When ODATHLDENL is negated, the Z80 is released from reset on the next rising edge of PROCCLKH. At this time, the Z80 would initialize its stack and do any needed house cleaning. To pass the controller I/O diagnostics, the Z80 must read a byte from the controller and echo it back. Because the controller is much faster than the Z80, we may assume that by the time the Z80 finishes house cleaning, a test byte is already present in the 2950. The Z80 reads the byte by performing an IN instruction that asserts CRDL. The Z80 may immediately write the test byte back out with an OUT instruction which asserts CWRL. After this sequence, the Z80 may start normal I/O operations to the controller. Since the 2950 is not reset by ODATHLDENL, the buffer full signals, FRH and FSH, may be invalid before this sequence and should not be used until diagnostics are finished.

Controller Input. When the Z80 performs as an *OUT* instruction that asserts the CWRL signal, a byte is sent to the controller from the Z80. The rising edge of CWRL:

1) Latches the Z80 data bus into the 2950 R register,
2) Asserts FRH flag from the 2950 (R buffer full), and
3) Asserts IDATSTBH, which starts a data input cycle.

The rising edge of IDATSTBH negates IDATACCH. When the controller is ready to read the data, it asserts IDATLRDENL which gates the stored data onto the controller data bus and asserts CLIDRDY (via an inverter). When IDATLRDENL is negated, IDATACCH is asserted, causing FRH from the 2950 to be negated (R buffer empty). This tells the Z80 that another byte may be sent to the interface.

Controller Output. When the controller outputs a byte to the interface, it asserts ODATLLDENL and places the data on its data bus. While ODATLLDENL is asserted, the 2950 S register is enabled and SODRDYH is asserted by an inverter. At the next rising edge of PROCCLKH:

1) The controller data is latched into the S register,
2) ODATRDYH is asserted by the controller, and
3) FSH is asserted (S buffer full).

By reading the status register, the Z80 can test the FSH signal and see that a byte of data is ready from the controller in the S register. When the Z80 executes an *IN* instruction that asserts CRDL,

1) The S register is gated onto the Z80 data bus,
2) FSH is negated (S buffer empty), and
3) ODATSTB is negated, then asserted.

At this time the controller negates ODATRDYH and the interface is ready for the next data transfer.

Programming. The interface uses three Z80 I/O ports:

1) a status port (read only),
2) a data output port (write only), and
3) a data input port (read only),

The status port has two active bits, FR and FS. FR is '1' when data is waiting to be read by the controller. FS is '1' when controller data is ready for the Z80 to read. To output a byte, the Z80 tests the FR bit in the status port and waits till it is a '0'. The Z80 can then send the next byte to the Display Controller by using an *OUT* instruction to the data output port. To read the next data byte from the controller, the Z80 tests the FS bit in the status register until it is '1'. The Z80 may then read the controller data byte with an *IN* instruction from the input data port.