CD1283/’1284 Evaluation Kit

Application Note

May 2001

As of May 2001, this document replaces the Basis Communications Corp. document *AN-CD6*.

Application Note

Contents

Figures

Tables

1.0 Evaluation Kit Contents

1.1 Hardware

- CD1284 Evaluation Board
- IEEE Std. 1284 type C to IEEE Std. type A cable

1.2 Software

- Floppy disk contents:
  - — CD1284 programming examples
  - — Genoa test suite programming example
  - — CD1284 schematics

1.2.1 References

- CD1283 Datasheet
- CD1284 Datasheet

2.0 Introduction

This document describes the CD1283/’1284 evaluation board.This board is an IBM PC, ISA bus-compatible plug-in board.

This board can be used to evaluate the CD1283 or CD1284 devices using the provided software and documentation, or customer-based software can be used. The board can also function as a development tool in an easy-to-use environment (IBM PC-compatible). Therefore, code implementation can begin while hardware is in the design/debug phase.

If the serial ports of the CD1283/’1284 are used, custom cables must be built by the end user. Section 4.0 on page 7 describes hook-up procedures, jumper settings, and board address assignments. Section 5.0 on page 9 describes the installation and operation of the demonstration software.

3.0 Supplemental Information

3.1 Interrupts

The board provides optional interface methods of the CD1283/’1284. These options are in interrupt selection and optional DMA access for the device. One interrupt is generated by the board for the CD1283/’1284. This interrupt is selected by the jumper wires on jumper block JP8. The left column of pins on the jumper block indicate the ISA bus interrupt number that can be selected; the right column of pins has the selection of pins with the interrupt source. This interrupt can be wired to any ISA bus interrupts appropriate to the system in use.

3.2 CD1283/’1284 DMAACK Select

The DMAACK input of the CD1283/’1284 can be driven from either of two sources:

- The DMAACK input from the ISA bus or an address decode from the on-board address decoder (see the users manual for the address assignment).This option allows evaluation of the CD1283/’1284 without programming the DMA controller on the PC.
- DMA cycles can be emulated by performing a 16-bit I/O read or write from the defined address decode. The selection is made through jumper J1, which should be set to 2–3. The programmed I/O method of emulation should be used. Jumper locations 1–2 are reserved for future development.

3.3 Ordering Information

Kit Number: CDK1284-E-AT02A

Software examples on floppy disk and evaluation kit documentation can be ordered separately from Intel at no additional charge.

4.0 Hardware Installation

4.1 System Requirements

The system requirements are:

- A PC/AT or compatible with minimum 640K RAM
- One available 16-bit expansion slot
- DOS version 3.3 or higher

4.2 Evaluation Board

The evaluation board is an add-on ISA card with a type C IEEE, 36-pin parallel port connector, two RJ-45 serial connectors, a MACH120, and the CD1283/’1284.

Note: Verify that the CD1283/’1284 is the latest version. The device should be printed with CD1284-10QC-E. If it is not, please return the board to Intel for replacement.

4.2.1 Switches and Jumpers

The SW includes eight switches; each can be set to ON or OFF to convey certain information to the PC. A jumper consists of two gold pins that may or may not be connected by a plastic connector plug. The I/O port address SW1 8-bit pattern is 01001000xxx (binary) or 240h. The card allocates 32 PC I/O base addresses from 0x240 to 0x25F. The description of these valid addresses is provided in Table 1. Table 2 presents jumper definitions and Table 3 provides switch settings.

Table 1. Switch Setup

PC I/O Base Address	Address Description
0x240	RESERVED
0x242	RESERVED
0x244	CD1283/'1284 register address
0x246	CD1283/'1284 reset
0x248	CD1283/'1284 CS
0x24A	SVCACKR
0x24C	SVCACKT
0x24D	SVCACKP
0X250	SVCACKM
0X252	Generate DMAACK for CD1283/'1284.

Table 2. Jumpers

Jumper	Description
JP1-	DMA Request (DREQ 0, 5, 6, and 7)
JP2-	DMA Acknowledge (DACK 0, 5, 6, and 7)
JP3-	Motorola ® /Intel ® format selector for the CD1283/'1284
JP4-	Reset switch
JP5-	Quickturn connector
JP6-	Quickturn connector
JP7-	Quickturn connector
JP8-	IRQ select for 3 through 15
J1-	CD1283/'1284 DMA acknowledge selector (JP2 or I/O command from the PC)

Table 3. Eligible Switch and Jumper Settings

W1	SW2	SW3	SW4	SW5	SW6	SW7	SW8	Base Address
ON	OFF	ON	ON	ON	ON	X	X	0x200
ON	OFF	ON	ON	ON	OFF	X	X	0x220
ON	OFF	ON	ON	OFF	ON	X	X	0x240
ON	OFF	ON	ON	OFF	OFF	X	X	0x260
ON	OFF	ON	OFF	ON	ON	X	X	0x280
ON	OFF	ON	OFF	ON	OFF	X	X	0x2A0
ON	OFF	ON	OFF	OFF	ON	X	X	0x2C0
ON	OFF	ON	OFF	OFF	OFF	X	X	0x2E0
ON	OFF	OFF	ON	ON	ON	X	X	0x300
ON	OFF	OFF	ON	ON	OFF	X	X	0x320
ON	OFF	OFF	ON	OFF	ON	X	X	0x340
ON	OFF	OFF	ON	OFF	OFF	X	X	0x360
ON	OFF	OFF	OFF	ON	ON	X	X	0x380
ON	OFF	OFF	OFF	ON	OFF	X	X	0x3A0
ON	OFF	OFF	OFF	OFF	ON	X	X	0x3C0
ON	OFF	OFF	OFF	OFF	OFF	X	X	0x3E0

5.0 Software Installation

The CD1283/’1284 demonstration software is provided on one floppy disk with six directories:

- 1284: genrn.c and genrn.exe programs to perform the Genoa test suite.
- BOARD: The CD1283/’1284 evaluation board schematics, MACH120 schematics, and 22V10 PAL schematics in OrCad®.
- DOC: The CD1283/’1284 technical documentation in PDF (portable document format) for online viewing.
- EVAL1284:Two programs that demonstrate CD1283/’1284 programming, and the operation of the device.
- MISC1284: Miscellaneous programming examples.
- INTERRUPT: An interrupt-driven code program demonstration.

5.1 Running the Demonstration Software

There should be two PCs when running these two programs; one is the host and the other the target.

1. Copy host1284.exe from the EVAL1284 directory to the host PC
2. Copy eval1284.exe from the EVAL1284 directory to the target PC.
3. Run host1284.exe on the host PC; simultaneously run eval1284.exe on the target PC.
4. Type ‘i’ on the target PC to initialize the CD1283/’1284 registers.
5. Type ‘n10’ on the host PC so the interface negotiates to ECP Forward.

Note: For other values of ‘n’ refer to IEEE Std. 1284-1994, page 23, Table 4.

6. Type ‘v’ on the target PC to view the CD1283/’1284 register contents after the negotiations.
7. Type ‘t’ on the host PC to terminate from the current protocol.
8. Type ‘v’ on the target PC to list the CD1283/’1284 register contents after protocol termination.

Note: The eval1284.exe runs on the CD1284 evaluation board; host1284.exe runs on Warp Nine Engineering® (previously called Far Point Communications®) parallel port on the host. The Warp Nine parallel card can be obtained from Warp Nine Engineering directly (phone: (805) 726-4420; fax: (805) 726-4438; www.fapo.com). The name of the parallel card used by Intel is the F/PortPlus parallel card.

6.0 CD1283/’1284 Parallel Channel Programming Guide

The CD1283/’1284 parallel channel provides unique features for straightforward IEEE 1284-compatible interface programming. Many functions are built into the device, such as automatic negotiation among the various protocols (reverse nibble, reverse byte, ECP, and EPP), built-in on-the-fly data compression using run length encoding/decoding, simple DMA handshake, and programmable FIFO threshold. These functions free the CPU from parallel interface interactions and helps eliminate some of the mundane tasks associated with higher-level protocol implementation.

The parallel channel of the CD1283/’1284 is comprised of two major functional blocks: the local CPU data interface and the parallel port control state machine. The CPU interface is further subdivided into three blocks: a 64-byte FIFO, a data pipeline, and a DMA buffer register.These major blocks share a com mon interrupt logic block that informs the CPU of conditions that require direct intervention.

6.1 Functional Blocks

Although separate entities, the CPU interface and the parallel port state machine are tightly coupled through a data interface that includes a data transfer buffer/latch and the status/control signals. Depending upon the direction of data movement within the parallel channel, control and status indicate the current state of the FIFO and the state of data in the parallel port. In the receive direction, FIFO control logic moves data into the FIFO (if space is available). In response to a parallel port request (if space is not available) the ‘move request’ is not honored. The parallel port remains in the BUSY state, stopping further transfers from the remote. In the transmit direction, status signals from the FIFO indicate if data is available to transfer to the remote. If data is available and the remote is not in the BUSY state, the parallel port transmits a byte out of the FIFO.

If DMA is enabled and the quantity of data in the FIFO is equal to or exceeds the programmed threshold (receive direction) or is equal to or less than the programmed threshold (transmit direction), a DMA transfer cycle initiates. The data pipeline logic attempts to fill the FIFO in the transmit direction; it attempts to empty it when the threshold is reached in the receive direction.Two registers are used for this determination, the PFTR (Parallel FIFO Threshold register) and the PFQR (Parallel FIFO Quantity register). Data is always moved into or out of the FIFO in 16-bit words; odd bytes must be handled directly by the CPU. For an odd byte, once a FIFO time-out occurs in the receive direction, the CPU removes the remaining byte from PFHR2 (Parallel FIFO Holding register 2). In the transmit direction, an odd byte at the end of a block transfer is placed in the FIFO through PFHR1 (Parallel FIFO Holding register 1). See Figures 11 and 12 in the CD1284 datasheet for a graphic presentation of the FIFO/data pipeline structure.

The CD1283/’1284 DMA port works in conjunction with the system DMA controller. Two handshake signals, DMAREQ* and DMAACK*, are provided for communication with the system DMA controller. When the device is ready for a DMA transfer, it activates DMAREQ*, which remains active until one of three events occur:

1. The FIFO is full (transmit)
2. The FIFO is empty (receive)
3. DMA transfers are disabled by the CPU

The duration of a DMA transfer can be controlled by the threshold setting. For example, if a burst of 20 words maximum is required in the transmit direction, the PFTR value would equal a full FIFO (the FIFO is byte-organized but is filled in word increments, where 64 bytes = 32 words) minus the desired burst size.(in this case, 64 – 40 = 20). When the FIFO quantity falls below this point, the CD1283/’1284 activates DMAREQ* and remains in effect until 20 words (40 bytes) are moved into the FIFO. For each word transferred the system activates DMAACK* once. This procedure is reversed for the receive direction.

For the receive direction, if the same 20-word burst is desired, the threshold should be set to 40. When the FIFO quantity reaches this point, the DMA logic attempts to empty the FIFO. Since the two holding registers are included in the burst count, the threshold should actually be set to two less than the desired burst length, or in this case, 38.

After device initialization, the CD1283/’1284 parallel channel enters Compatibility mode where the parallel port is an input-only Centronics® -compatible port. The port receives data; all handshake signals operate as defined for Compatibility mode in the IEEE specification. For any other mode and to move data bidirectionally, the CD1283/’1284 must negotiate with a remote master 1284 device. Also, the CD1283/’1284 must have the negotiations enabled for desired mode(s). When the remote master attempts negotiations into one of the other modes, any enabled mode is accepted by the device.

Note: The CD1283/’1284 cannot move into one of the bidirectional modes on its own; a remote master is required. Modes are enabled by the NER (Negotiation Enable register). Typically, all modes would be enabled so that the peripheral communicates in any mode requested by the remote. However, if only some modes are supported, only the bits in the NER corresponding to those modes are set.

When a successful negotiation completes, the CD1283/’1284 posts a CPU interrupt notifying the change. The NSR (Negotiation Status register) indicates the negotiation results and mode entered. The status bits indicate if the negotiation completed correctly, NegOK, or with error, NegFI. If an invalid negotiation was attempted (the extensibility request bits on the bus during negotiations were not a recognized value), then the NegFI status bit is set along with the Invalid status bit. If an invalid state was requested or entered, the port returns to Compatibility mode (default); any port failures force a return to Compatibility mode. Refer to the IEEE Std. 1284-1994 specification for a detailed discussion of the negotiation sequence.

6.2 Using the Parallel Channel

Several steps must be performed before the parallel channel can be used. For example, basic channel initialization must be performed. Examples include: setting the correct count for the short pulse duration based on the frequency of the master device clock, setting the FIFO threshold, enabling supported modes, and so on. Once the initialization operations complete, the CPU should set up the system DMA controller for receiving and transmitting data. Finally, the interrupts, DMA, and parallel port transfers should be enabled in the CD1283/’1284. Each of these initialization operations are detailed in Section 6.2.1.

For the purposes of this document, the operating frequency of the clock (CLK) is 25 MHz. Also, the two serial channels in the CD1283/’1284 are ignored. Manual control of the channel through Manual mode and the Manual Data register are not discussed; this mode is intended primarily for testing and special cases and is not considered part of normal operation. Flowcharts of the various operations are provided in Section 7.3.

6.2.1 Channel Initialization

The initialization of the channel can be performed in two sequences: First the parallel port and then the data pipeline.

6.2.1.1 Parallel Port

During initialization of the parallel port, the supported modes must first be selected by setting the appropriate enable bits in the NER. The E1284 bit in the PCR (Parallel Configuration register) must be set for any higher-level parallel port operations. The ETxfr bit in the PCR must also be set for transfers on the port. However, this bit can be set after the DMA controller and interface have been initialized. If transfers are enabled and begin before any DMA transfers can occur, the FIFO fills and the port is the BUSY state.

Note: This is not an error but may be inconsistent with the rest of the system programming.

One step that must be performed for proper operation of the parallel interface is setting the minimum pulse width to 500 ns, as defined in the IEEE specification. This pulse width duration allows maximum data transfer performance; however, longer pulse widths are allowed.

The pulse width is set by the value programmed in the SPR (Short Pulse register). The SPR is a simple counter/divider driven by the CLK input therefore, the value loaded depends upon the operating clock frequency of the CD1283/’1284. For a 25-MHz clock, the proper value is 13; this value produces a minimum pulse width of 520 ns (as close to 500 ns as can be generated with a 25-MHz clock).

$$13 \times \frac{1}{25MHz} = 520ns$$

For EPP mode support, load the EAR (EPP Address register) with the value driven on the parallel port during an EPP address read cycle.

The pin control registers are not used during normal, non-compatible mode 1284 transfers with the parallel port. One exception is during EPP mode. In this case, three signals in the OVR (Output Value register) are user-defined and are labeled as USER1–USER3. If these three signals are used, then set their value in the OVR before EPP mode is entered.

In Compatibility mode, the OVR delivers port status to the host. This status might consist of paper empty, fault, and so on. To set the status pins, the host must place the port into Manual mode, set the desired status in the OVR, then resume normal operation.

The final step in initializing the parallel port is setting the interrupt enable bits as required in the PCIER (Parallel Channel Interrupt Enable register). Under normal operating conditions, all interrupts should be enabled. However, if some modes are supported, not all bits are set. For example, if EPP mode is not enabled, the EPPAW (EPP Address Write) interrupt enable need not be set.

This completes parallel port configuration. The next step is configuring the data path registers.

6.2.2 Data Pipeline

The data pipeline has registers for FIFO control, DMA circuitry, and the timer. There are also several registers that display the current status of the entire pipeline.

The FIFO data threshold used for triggering DMA transfers is set in the PFTR. During operation, the CD1283/’1284 compares this value to that in the PFQR (Parallel FIFO Quantity register) as a DMA burst requirement. In the receive direction, if PFQR is greater than PFTR, the device requests a DMA transfer to empty the FIFO. DMA remains active until the last full word (16-bits) is transferred; a remaining odd byte is left in PFHR2 for the CPU to remove manually. In the transmit direction, if PFQR is less than PFTR, DMA transfers occur and remain active until the FIFO is full.

The CD1283/’1284 ensures that data is not trapped in the FIFO (‘stale’) if the amount of data received does not reach the threshold programmed in the PFTR. This feature is implemented with a stale data timer. The timer duration is set by the value placed in the SDTPR (Stale Data Timer Period register). The timer is driven by a clock generated by dividing the master device clock (CLK) by 250. This ‘intermediate’ clock is then prescaled by a fixed divide-by-ten algorithm to produce the basic stale timer period. Thus, with a 25-MHz clock, the resolution of the timer is 0.1 ms.

Each time a character is placed in the FIFO, the SDTCR (Stale Data Timer Count register) is loaded from the value in the SDTPR. If new data does not arrive to restart the timer, it expires after the programmed duration set in SDTPR. This forces the ‘stale’ condition to become true and triggers a DMA transfer to empty the FIFO.

The purpose of this timer is to determine when a block transmission is complete. Data arriving can be considered to be a block or ‘frame’ of data, but the normal protocol of moving parallel data between a host and a peripheral does not include the concept of a message therefore, does not have a defined way of delimiting one message from another. However, once a text block transmits (or commands transmit from a high-level page-description language such as PostScript), there is a pause while a new block is prepared. This pause enables the timer to expire, allowing the local CPU to assume the transmission is complete.

There are several miscellaneous control bits for special pipeline operations. These bits are in the PACR (Parallel Auxiliary Control register). During the initialization process, only two bits in this register are used, Unfair and Async DMA. The Unfair bit is associated with the serial channels. The Async DMA bit provides alternate timing for the DMA control circuitry. These bits are discussed in the *CD1284 Datasheet*, but are not relevant to this document.

The vector driven on the data bus during a parallel channel service acknowledge cycle (SVCACKP* input) is set by the LIVR (Local Interrupt Vector register). The most-significant five bits are set to the appropriate value for the system interrupt service routine. The least-significant three bits are supplied by the CD1283/’1284 and indicate the interrupt source within the parallel channel (pipeline, parallel port, or both).

The PFCR (Parallel FIFO Control register) must be set to enable the channel and any supported special operations. For beginning any activity within the parallel channel, a FIFO reset command and the desired DMAdir must be issued. This initializes the channel and sets the direction of the pipeline and parallel port. This procedure must be performed whenever the direction of the parallel channel is changed. The RLEen bit must also be set if Run-Length data compression is supported in ECP mode.

6.3 Channel Operation

The operation descriptions that follow assume that the interface is in Compatibility mode (default state after device reset) and that the preceding initialization procedures were performed. Final initialization includes setting the IntEn bit so that interrupts can be generated by the parallel channel. The ErrEn bit must also be set if interrupt sources in the Data Error register are to be included in interrupts from the channel. Finally, the DMAen bit is set to enable DMA transfers.

The parallel channel operates in one of many modes and moves freely between modes at the command of the remote master. The CD1283/’1284 can never change modes on its own; it must always wait for the master to complete the negotiation sequence. Available modes are enabled individually by the control bits in the NER.

EPP mode operation differs significantly from other modes. It does not incorporate ‘receive’ and ‘transmit’ but behaves more like a processor address/data bus. As such, it is not included with the other modes that follow. See Section 6.4 on page 17 for more details on EPP mode.

6.3.1 Receiving Data — Compatibility Mode

As data arrives on the parallel port, it begins filling the FIFO. The first two bytes will fall through to the bottom of the FIFO and then into the holding registers, PFHR1 and PFHR2. Subsequent data continues filling the FIFO. DMA transfers when the programmed threshold is reached and continue until the FIFO and holding registers are empty. Incoming data continue filling the holding registers and FIFO until the threshold is once again reached, then DMA transfers commence. The DMA request signal (DMAREQ*) remains active until the FIFO is empty, in this way the programmed threshold sets the burst duration of the DMA transfer.

If incoming data ceases for a period longer than the programmed stale data time-out period, one of two situations occur. If there is no data in the pipeline because an even number of bytes was received, nothing occurs, except the Stale bit in PFSR is set.

Odd transfers leave a single character in the holding register because the DMA interface only performs 16-bit transfers. As such, if an odd number of bytes is received, a time-out occurs. In addition to a time-out indication, the OneChar status bit is set indicating that the CPU must manually remove the last byte of the transfer from the holding register.

The CPU must then toggle the ClrTO bit in the PACR to remove the Time-out interrupt status and rearm the time-out mechanism for the next transfer from the parallel port.

6.3.2 Receiving Data and ECP Mode

If the remote master opts to use ECP mode, it enter negotiations with the CD1283/’1284 to set this mode. The local CPU must have previously enabled ECP mode in the NER. Upon completion of the negotiation sequence, the CD1283/’1284 posts an interrupt with status in the NSR indicating that a negotiation change occurred. The new mode is indicated in the four-bit result code. For the purpose of this document, it is assumed that the transfer direction remains receive.

Unless RLE data compression is enabled and RLE compressed data is being received, basic operation of the parallel channel in ECP mode receive is the same as Compatibility mode. DMA transfers commence when the level in the FIFO reaches the programmed threshold and continues until the FIFO is empty or a time-out occurs due to a single character remaining in the holding registers or no more data remains.

If RLE compression is enabled and RLE-compressed data is being received, the behavior of the pipeline (the holding registers plus the DMA buffer) changes slightly. RLE-compressed data is sent over the parallel interface as a two-byte sequence; a command and data byte (refer to the IEEE 1284 Std. specification for a complete description of ECP mode RLE compression.). The command byte indicates that this pair is RLE compressed and includes the byte count; the second byte is the data to be replicated. The pipeline stores the count, then repeats the data while decrementing the counter. If the resultant count of repeated characters is odd, then the last character in the compressed sequence is joined by the next character in the FIFO to form the full 16-bit word for DMA transfer. If the odd character is the last character in the transfer and a time-out occurs, then the OneChar interrupt is generated.

One other function is available in ECP mode. In addition to RLE compression using RLE command/data sequences, an address can be received using an address-command/address byte sequence. When the CD1283/’1284 receives an address command, it stops DMA movement and posts an interrupt indicating that a ‘tagged’ byte has been received. The CPU must manually remove the tagged character from the holding register. The PFSR and HRSR (Holding Register Status register) determine that a tagged character has been received and which holding register it is in. After the tagged character is removed from the holding register, normal data movement continues.

Note that if RLE compression is not enabled in the PFCR, the receipt of an RLE compressed data sequence causes data movement to stop and a tagged-data interrupt to occur.

6.3.3 Changing Directions

To send data to the remote master, the link must be reversed. Reversal occurs when the master negotiates into one of the directions capable of reverse transfers (Reverse Nibble, Reverse Byte, ECP, or EPP) and the local CPU has data to send. Status indicating the availability of data is provided on the parallel interface during the negotiation sequence and differs in implementation for each mode; however, the procedure used by the local CPU is the same. EPP is a special case of reverse-data transfer and is quite different from the others. The EPP method of moving data in the reverse direction is discussed in Section 6.4.

It is important to remember that the CD1283/’1284 cannot initiate a reversal; it must wait for the remote master to initiate the reversal with a negotiation sequence. The CD1283/’1284 can request a direction reversal through the RevRq bit in the Special Command register.

The CD1283/’1284 requests the reversal by using a command bit in the SCR, RevRq (bit 0). When the local CPU is ready to send data to the remote master, it sets RevRq and waits for the remote to perform a negotiation sequence. Then, the CD1283/’1284 activates the appropriate signal, nDataAvail, to indicate that reverse data is available. When negotiations are complete, the CD1283/’1284 posts an interrupt to the local CPU indicating a direction change (the DirChg bit is set in PCISR). Upon receipt of this direction change interrupt, the CPU should proceed to change the direction of the parallel channel through a command to the PFCR. This is done by setting the FIFOres and DMAdir bits. Program the PFTR next if the threshold is to be changed for the transmit direction.

Finally, after programming the system DMA controller, the DMAen bit is set in the PFCR to initiate the actual reverse data transfer. As specified in the IEEE Std. 1284 specification, this data pipeline direction change is allowed to take up to an infinite amount of time to occur. When the pipeline and FIFO have been reversed, DMA transfers begin filling the FIFO. When the first data byte is available to the parallel port, it begins moving the data out with the defined Strobe/Acknowledge sequence.

With the exception of EPP mode, transmitting data to the remote in Reverse Nibble, Reverse Byte and ECP modes occurs in similar ways. As long as data is available, the nDataAvail status bit remains asserted and the remote continues to be fed data. When there is no longer any data in the FIFO, nDataAvail deasserts, at which time the remote master might negotiate back into a forward data direction and return to forward data transfers. However, this is unnecessary; the link may be left in the reverse idle state. If this is the case or if more data arrives in the FIFO, data movement can start again.

6.3.4 Transmit Data — Reverse Nibble Mode

When data (status or ID request responses) must transfer from local memory to the remote master, the local CPU initiates a reversal as previously stated. Data is transmitted in four-bit nibbles using the Centronics-defined status lines as data lines rather than using the actual data signals.

This method is used if reversed data transmission is necessary, but the data lines are not bidirectional, as could be the case on older host systems. Each byte of data is transmitted using two nibbles with the least-significant four bits being sent first. The transmission speed is necessarily slow due to the requirement to.break the data up into the four-bit nibbles (transmission speed is half that if full bytes are being transmitted).

Handshake signals, Strobe, Acknowledge, and Busy are implemented using the reverse sense. The handshake is in the forward direction. The strobe from the local to the remote uses the ACK* signal. The acknowledge from the remote to the local uses the STROBE signal. BUSY is implemented using nAutoFeed as BUSY.

6.3.5 Transmit Data — Reverse Byte Mode

Reverse Byte mode is similar to Reverse Nibble mode except that data is moved using the parallel data signal lines as the data lines, rather than the status lines. Generally, new host systems provide a bidirectional data interface so a full, byte-wide data path exists in both directions. The Strobe, Acknowledge and Busy functions are implemented using the same signals as used in Reverse Nibble mode.

6.3.6 Transmit Data — ECP Mode

ECP mode transmit is similar to Reverse Nibble and Byte modes, but adds the capability to send RLE-compressed data. It also includes the ability to send the channel address sequence, as is the case during receive data.

If RLE compression is enabled, the data pipeline scans incoming data for identical sequences of characters. If the number of duplicate sequential characters is greater than two, the pipeline begins to count the characters while maintaining a copy of the character. When a non-matching character arrives or the count of characters reaches 128, the pipeline logic generates a tagged, two-byte sequence including the command and data. The resultant

two-bytes are placed in the FIFO along with the tag status. When this tagged reaches the parallel port, it transmits while activating the appropriate control signals to indicate that it is an RLE-compressed command.

Sending a channel address requires that the local CPU manually place the tagged-byte address in the PFHR1 after setting the SetTag bit in the PFCR. When the tagged byte reaches the parallel port, it is transmitted with the control signals set to indicate an address byte.

6.4 EPP Mode

EPP mode differs significantly from other modes, primarily because its behavior is more like a microprocessor interface bus than a typical peripheral parallel interface. It defines both address and data cycles using the handshake signals to indicate address reads or writes, and data reads or writes. It is also the case that all transactions on the ‘bus’ are initiated by the master, so there is no equivalent to a receive or transmit operation. It is not clear how this mode should be used in a peripheral interface (printer, scanner, and son on), but it is included in the CD1283/’1284 so as to be completely compatible with the IEEE 1284 Std. specification.

EPP mode has the capability of high-speed data movement as long as it does not use an address cycle with each data transfer cycle. If this ‘burst’ mode is used, minimal local CPU interrupt overhead is required. Each time an address write is performed on the parallel interface, the value written is placed in the EAR (EPP Address register) and, if the EPPAW bit is set in the PCIER, an interrupt is generated. However, if cycles are primarily data writes, then the FIFO and pipeline perform in the same manner as the other receive modes: Data flows into the FIFO then into the holding registers, and when the programmed threshold is reached, DMA transfers commence and continue until the FIFO is empty.

Reading from the parallel port while in EPP mode requires that the pipeline be reversed before the first read transaction occurs. This is because the CD1283/’1284 is not in control of the data movement handshake; the remote controls movement through the data read cycle. Therefore, the pipeline must be reversed and the FIFO filled before the remote can read the data. For this switch to occur, some higher level protocol must be implemented in software so that the message contents indicate a reversal is requested.

7.0 Programming Examples

This section provides specific programming examples for various operations required by the CD1283/’1284. These examples expand upon examples shown in the *CD1284 Datasheet* and include parallel channel initialization and code to handle a direction reversal, the single remaining character left in the pipeline due to an odd byte count, and so on. These examples are in Borland® C++.

7.1 Initialization Code

Initialization of the parallel channel consists of setting the SPR, selecting modes supported during negotiation, setting the stale data time-out value, initializing the FIFO, setting the source for acceptable inter-rupts, and other operational functions.

```
par_init()
{
/* First, issue chip reset command */

outportb(GFRCR, 0x00);       /*Clear the GFRCR*/

outportb(CAR, 0x02);         /* Set channel 2 (could also use 3) in CAR */
while (inportb(CCR) != 0x00)
            ;                /*Wait for CCR to clear */

outportb(CCR, 0x81);
while inportb(GFRCR) == 0x00)
            ;                /* Wait for GFRCR to be set */

/* Start by initializing the parallel channel */

outportb(CAR, 0x00);         /* Set channel 0 in the access register */
outportb(SPR, 0x0D);         /* Assume 25MHz clock, set short pulse value */
outportb(NER, 0x4F);         /* Support all modes except EPP */
outportb(PCR, 0x80);         /* Set manual mode */
outportb(OVR, 0x58);         /* Start in Compatible Mode, set status signals: */
                             /* PError = 0 */
                             /* SELECT = 1 */
                             /* nFault = 1*/
                             /* nACK = 1 */

outportb(PClER, 0x37);       /* Enable all interrupts except EPP address write */
outportb(PCR, 0x60);         /* Enable 1284 negotiations and transfers */
/* Next, set up the pipeline control registers */

/* Clear the GFRCR */

outportb(LIVR, 0x00);        /* Clear the LIVR, set device ID to 0 */
outportb(PFCR, 0xD8);        /* Enable pipeline DMA, set the direction to input, */
                             /* enable interrupts (but not error ints) and reset*/
            /* the FIFO. At reset, it is assumed that the starting */
            /* direction will be input. */
outportb(PFCR, 0x58);        /* Clear FIFOres to complete reset operation */
outportb(PFTR, 0x20);        /* Set the DMA threshold for receive (burst = 32) */
outportb(SDTPR, 0x64);       /* Set the stale data time-out period to 10ms */
outportb(PACR, 0x02);        /* Set asynchronous DMA mode */
}
```

7.2 Service Requests

When the CD1283/’1284 parallel channel requires local CPU intervention, it posts an interrupt with the status in the PIVR indicating which section of the channel, the parallel port or the pipeline, requires service. Each section has its own status register to specify which of several possible sources is the cause of the interrupt. The following sections provide detailed descriptions of the interrupt sources and software examples of service routines.

7.2.1 Parallel Port

The parallel channel can post requests for service from either the pipeline or the port. This section discusses service requests from the port. Section 7.2.2 on page 21 discusses the pipeline.

Service requests from the port can occur when one or more of several events happen. These requests are individually enabled by the following bits in the PCIER:

- **nlNIT:** The remote master pulsed the nlnit signal on the parallel interface (Compatibility mode only).
- **IDReq:** During negotiations, the remote master requested a device ID.
- **DirCh:** The remote master reversed the direction of the channel. This generally occurs in response to a data available state set by the CD1283/’1284.
- **EPPAW:** An EPP address write cycle occurred on the parallel port (EPP mode only).
- **SigCh:** One of the programmed signal transitions (ZDR or ODR) occurred on the parallel port (Manual mode only).
- **NegCh:** The remote master performed a negotiation and the state of the port has changed.

Refer to the *CD1284 Datasheet* for the service request handling code. The code following segment example shows how service of the parallel port request might occur. This code segment example might be called in response to the ‘service_par’ routine shown in the *CD1284 Datasheet* and is the routine called ‘service_par_chan’.

Many of the reactions to these interrupts are system-specific to allow useful examples to be shown in documentation. However, some requests have general responses that can be shown in example code. These responses must make a change to device operation, for example, direction reversal of the parallel channel.

The first step in responding to the parallel port service request is to read and parse the contents of the interrupt status register, PCISR. The following is an example of this routine. Note the final command issued at the completion of the routine; it is required that the PCISR be cleared by the local host CPU to remove any pending requests.

```
/* This routine is called by the main parallel channel interrupt handler,
service_par*l/
/*(shown in the data book). */

service_par_chan()
{

char status;

status = inportb(PCISR);     /* Get the interrupt sources from status register */
while (status){              /* Do all interrupt sources*/
   switch (status) {      /* The case statement is used to prioritize the response
```

```
*/
   case 0x01:                /* This example simply steps through each source */
             nlnit();
             status &= 0xFE;  /* Clear this status, recycle */
             break;
   case 0x02:
             IDReq();         /* IDReq could also include a Mode change */
             status &= 0xFD;  /* Routine should also check for NegCh */
             break;
   case 0x04:
             DirCh();
             status &= 0xFB;
             break;
   case 0x08:
             EPPAW();
             status &= 0xF7;
             break;
   case 0x10:
             SigCh();
             status &= 0xEF;
             break;
   case 0x20:
             NegCh();
             status &= 0xDF;
             break;
   default:
             break;
      }
}
outportb(PCISR, 0x00);                 /* Clear pending requests */
outportb(PFCR, inportb(PFCR) & 0xEF);  /* Toggle IntEn to clear pending request */
outportb(PFCR, inportb(PFCR) | 0x10);
}
```

The nlnit handling is system-dependent, as is the response to a SigCh interrupt. The nlnit interrupt only occurs in Compatibility mode; the SigCh interrupt only occurs in Manual mode. nlnit might be used as way to force the printer into a reset condition. Signal changes can only generate an interrupt if Manual mode is being used and the signals are used as status/ control. In other modes, the output signals are under automatic control of the parallel port and the input signals are used by the remote host as part of the data transfer protocol.

Two responses require more clarification because specific changes must be made to the device: the DirCh and IDReq interrupts. An IDReq interrupt request response could follow somewhat the same.procedure as the DirCh interrupt response, since it normally involves a direction change and transmission of the ID string. The following routine shows one way of responding to the DirCh interrupt.

The following routine is an example of servicing interrupts from the data pipeline and directing processing to the appropriate routine. It is beyond the scope of this document to attempt to show in detail how the system might respond to these interrupt conditions as that is dependent upon a number of system architecture parameters. In general, the response to a 'Time-out with OneChar' is to remove the remaining character and place it in the receive buffer; 'Time-out without OneChar' the buffer might be tagged as complete. An interrupt with HRtag indicates that either an ECP tagged address was received and a change in the virtual device address made. If this is the purpose of the ECP, device addresses in the application, or an RLE command were received, but automatic RLE decompression was not enabled.

```
/* This routine responds to the direction change interrupt by issuing the required
commands to flush the FIFO and reverse the direction of the data path. It is
assumed that all data has been removed from the FIFO before the DirCh interrupt was
generated. Optionally, the routine could wait for the FFmpty bit to become active
```

```
in the PFSR before the reversal is initiated. The general procedure is to set the
DMAdir bit to 1 for transmit, 0 for receive and then set the FIFOres bit. Setting
FIFOres is required as this is a command to initialize the direction circuits and
flush the FIFO. FIFOres will clear automatically after the action is complete. */

extern char direction; /* Global direction value */
DirCh()
{
char pfcr_val;                    /* Holding location for current status */

pfcr_val = inportb(PFCR);
switch (direction) {
   case 0:
             direction = 1;       /* Change direction to transmit */
             pfcr_val |= 0xA0;    /* Set FIFOres bit and set DMAdir = 1 */
             outportb(PFCR, pfcr_val);/* Issue command */
             break;
   case 1:
             direction = 0;       /* Change direction to receive */
             pfcr_val &= 0xDF;    /* Set DMAdir = 0 */
             pfcr_val |= 0x80;    /* Set FIFOres bit */
             outportb(PFCR, pfcr_val); /* Issue command */
             break;
   default:
             break;
   }
}
```

7.2.2 Pipeline

There are three sources of interrupts from the pipeline

1. Those that occur when the stale data timer expires.
2. Those generated at the arrival of tagged data when the port is operating in ECP mode.
3. Those due to data handling errors on the part of the local system.

The stale data timer causes the Time-out condition to be set in the PFSR when it decrements to zero and there is either a single or no characters remaining in the data pipeline. As explained in the *CD1284 Datasheet*, this timer restarts whenever a new character is placed in the FIFO by the parallel port. Once the regular arrival of data stops, the timer expires and, if the FIFO is empty, the Time-out interrupt is posted. If there were an odd number of bytes in the transferred block, then the OneChar status is also set because there would be one character remaining in the PFHR2. If there are two or more characters remaining in the FIFO, the expiration of the timer causes a DMA cycle to be initiated to empty it, then the Time-out interrupt or the Time-out with OneChar interrupt is posted.

The 'tag' interrupt is generated if ECP-tagged data is received by the port. ECP tagged data can be either an ECP address or RLE-compressed data. If RLEen is set in the PFCR, RLE data is automatically decompressed and no interrupt is generated. If RLEen is not set, the receipt of RLE-compressed data causes an interrupt, which local host software must decode manually.

Data error interrupts are only generated if the DataErr bit is set in the PFCR. If DataErr is set, the occurrence of any of the conditions described in the Data Error register cause an interrupt to be posted. These errors are induced by erroneous read/write operations by the local system, such as reading an empty holding register or writing to the DMABUF register when it already contains data. The data error interrupt is primarily intended for debug purposes and would not be used during normal system operation.

```
/* This routine is called by the main parallel channel interrupt handler,
service_par. */

/* It checks for either a Time-out or Tag interrupt and directs the service
accordingly. */

/* If a Time-out was posted, the ClearTO bit in PACR must be toggled in order to */

/* clear the Time-out status in PFSR. */

service_pipeline()
{

char status;

status = inportb(PFSR);      /* Read the status register */
switch(status & 0x30) {/* Just check status of bits that cause ints */


   case 0x10:
             HR_tag(status); /* Holding register has tagged data */
             break;
   case 0x20:
             Time_out(status); /* Time-out, check OneChar, etc. */
             outportb(PACR, inportb(PACR) | 0x08);/* Toggle ClearTO bit */
             outportb(PACR, inportb(PACR) & 0xF7);
             break;
   default:                  /* Must be both */
             Time_out(status);
             outportb(PACR, inportb(PACR) | 0x08);/* Toggle ClearTO bit */
             outportb(PACR, inportb(PACR) & 0xF7);.HR_tag(status);
             break;
}
outportb(PFCR, inportb(PFCR) & 0xEF); /* Toggle IntEn to clear
pending request */
outportb(PFCR, inportb(PFCR) | 0x10);
}
```

7.2.3 Miscellaneous Pipeline Routines

For diagnostic purposes, it is possible to perform a 'loopback' test of the pipeline to verify correct movement of data in and out of the FIFO, as well as testing RLE compression/ decompression. Doing this involves placing data into the FIFO through the pipeline logic in a transmit operation then reversing the direction and removing the data after updating the quantity value in the PFQR (Parallel FIFO Quantity register). All of this is done without enabling transfers in the parallel port so that the device will not attempt to actually move the data over the parallel interface. To facilitate this, the FIFOlock bit is set in the PACR (Parallel Auxiliary Control register). The following is an example of this operation:

```
/* This routine tests proper operation of the parallel FIFO using a 'pseudo'
loopback */

/* operation. The routine first sets the FIFOlock bit in PACR to prevent the
parallel port */

/* from trying to move the data out of the FIFO. Data is then moved into the FIFO
via the */

/* PFHR2 (could also use PFHR1 using single bytes instead of words, followed by */

/* reversing the direction and updating the quantity value in PFQR to 'fake' a full
FIFO */
```

```
/* Data read out is compared with the data put in to verify correct operation. */


#define fail 1;

#define pass 0;

loopback() /* Loopback without RLE compression test */

{

int pattern[] = {0x11, 0x22, 0x44, 0x88, 0x55,0xAA};/* Test pattern, walking
one's */
int i, temp;

/* Set up channel for output and fill the FIFO */

outportb(PACR, 0x10);        /* Lock FIFO */
outportb(PFTR, 0x20);        /* Set threshold value (only needed in DMA, really) */
outportb(PFCR, 0xA0);        /* Set direction and FIFO reset */
outportb(PFCR, 0x20);        /* Clear FIFO reset */

for (i = 0; i < sizeof(pattern); i++) { /* Fill FIFO with first pattern */
   while (!(inportb(HRSR) & 0x04))

        ;                    /*Wait for DMAbuffer to be empty */
   outport(PFHR2, pattern[i]); /* Stuff word into buffer */

}

if ((temp = inportb(PFQR)) != 58){ /* Read the quantity register, should be (64
- 6) */
   outportb(PACR, 0x00);     /* Unlock FIFO */
   return(fail);             /* Quantity should have been 6 */
}

outportb(PFCR, 0x00);        /* Reverse direction */
outportb(PFQR, (64 - temp));/* Load new quantity for receive direction */
outDortb(PFTR, 0x01); /* Set a low threshold (only needed in DMA, really) */
for (i = (sizeof(pattern)*2); i > -1; i--) {
   while(linportb(HRSR) & 0x20)
   ;
   temp = inportb(PFHR2); /* Read data out when PFHR2 is full */
   if (temp != pattern[i]){ /* Read data */
            outportb(PACR, 0x00); /* Unlock FIFO */
            return(fail);
   }
}

outportb(PFCR, 0x80);        /* Reset FIFO and Reverse Direction */
outportb(PFCR, 0x00);        /* Reverse Direction */

if (inportb(HRSR) & 0x20){
   outportb(PACR, 0x00); /* Unlock FIFO */
   return(fail);             /* Shouldn't have been any chars left */
}
outportb(PACR, 0x00); /* Unlock FIFO */
return(pass);
}
```

This same general loopback method can be used to test RLE compression/decompression. To perform this test, the pattern placed in the FIFO should have a number of repeating values (greater than 2) and RLEen must be set in the PFCR. Also, to have RLE compression/ decompression function, the data must be placed into, and removed from the pipeline through the DMA buffer.

After loading the FIFO, the PFQR should contain a value that shows compression occurred. For example, if a repeating pattern of characters that is six in length is placed in the FIFO along with one additional, non-matching character, the PFQR should have a count of three, two used by the compressed data and count/tag plus the single character that did not match the pattern.

7.3 Flowcharts

This section provides flowchart examples of interrupt-driven and polling method code for the CD1284.

Figure 1. Polling Method

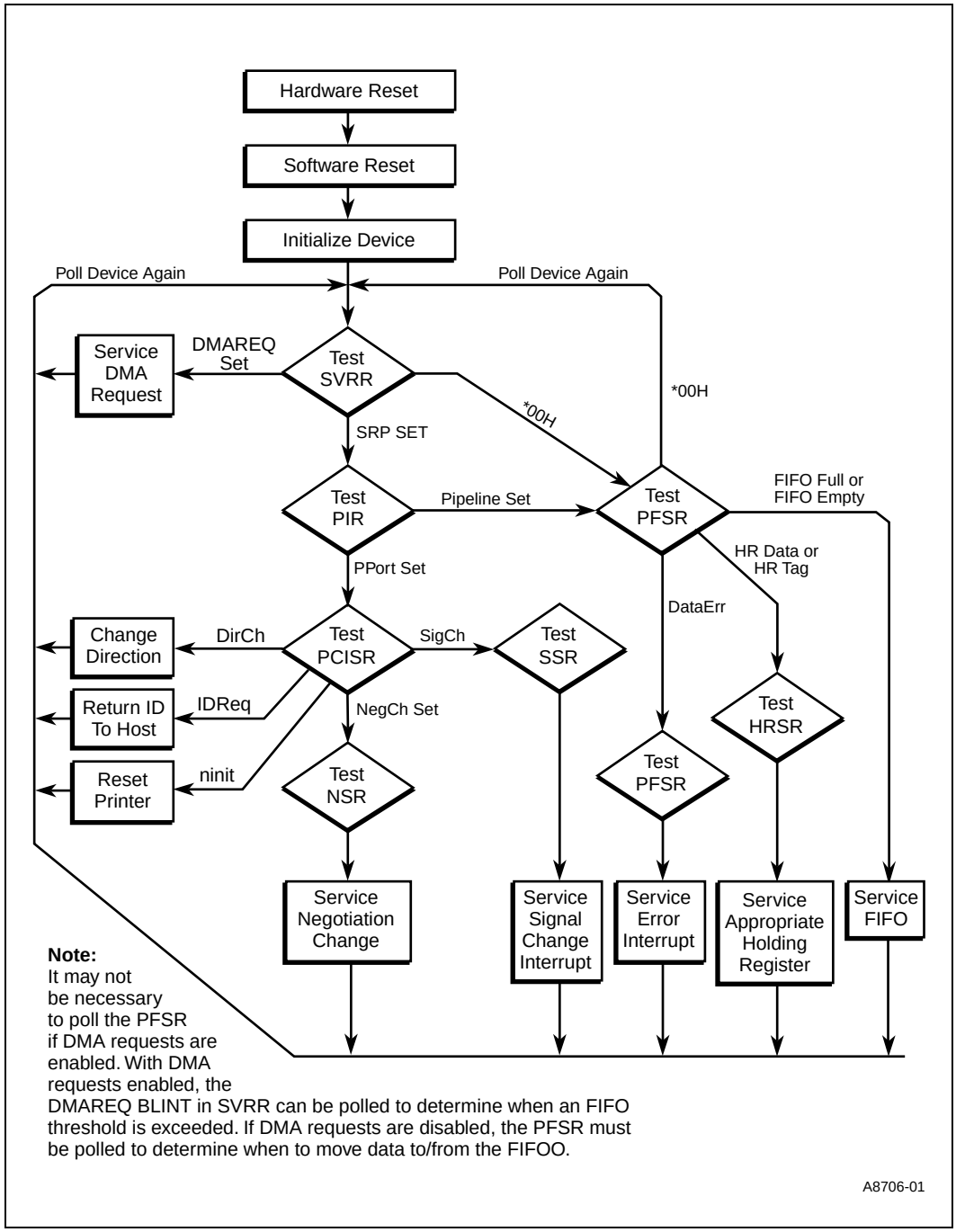
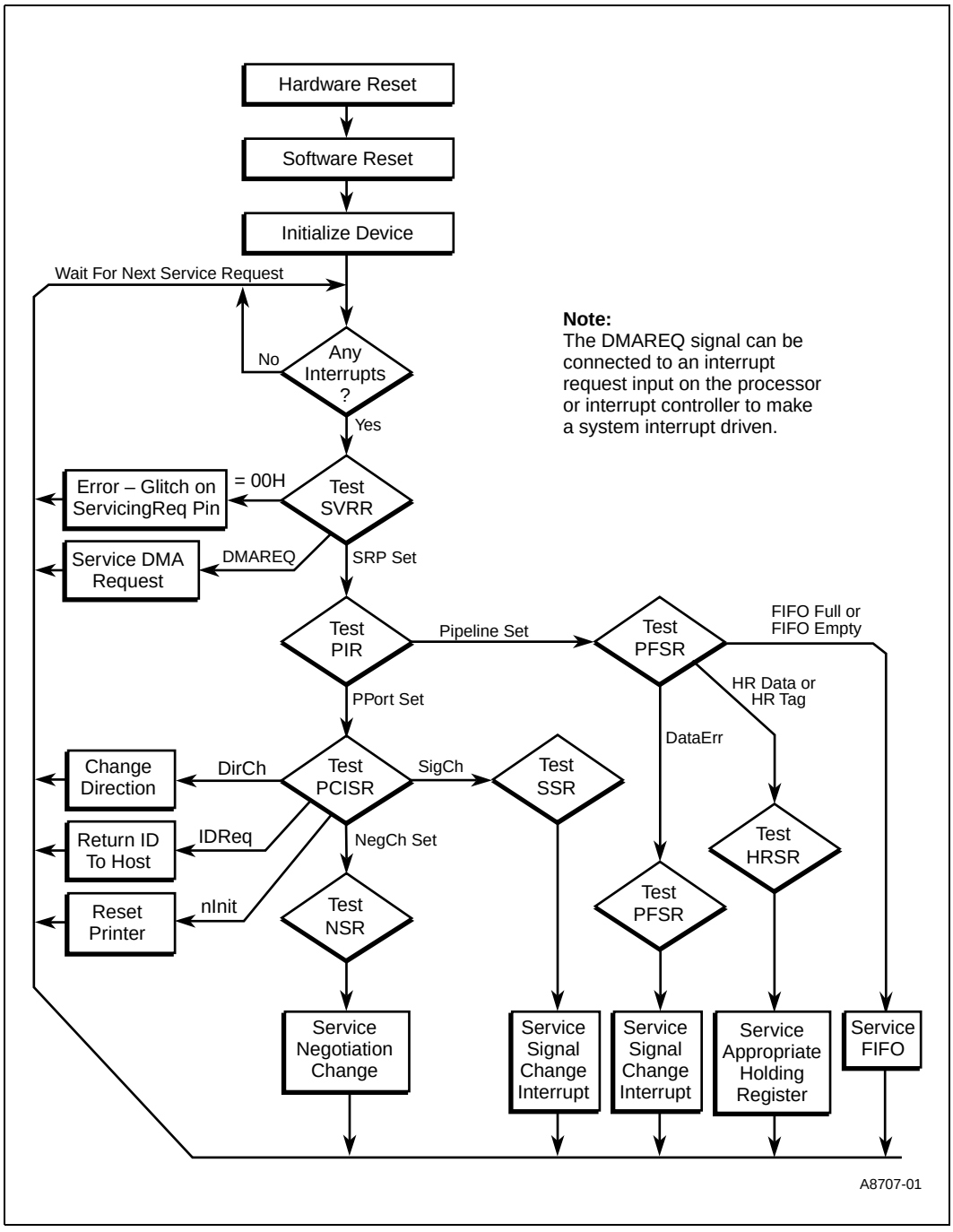


Figure 2. Interrupt-Driven Method



8.0 PAL Equations and Schematic

This section includes PAL equations for the MACH120, and the 22v10 PAL device, and a schematic for the evaluation board.

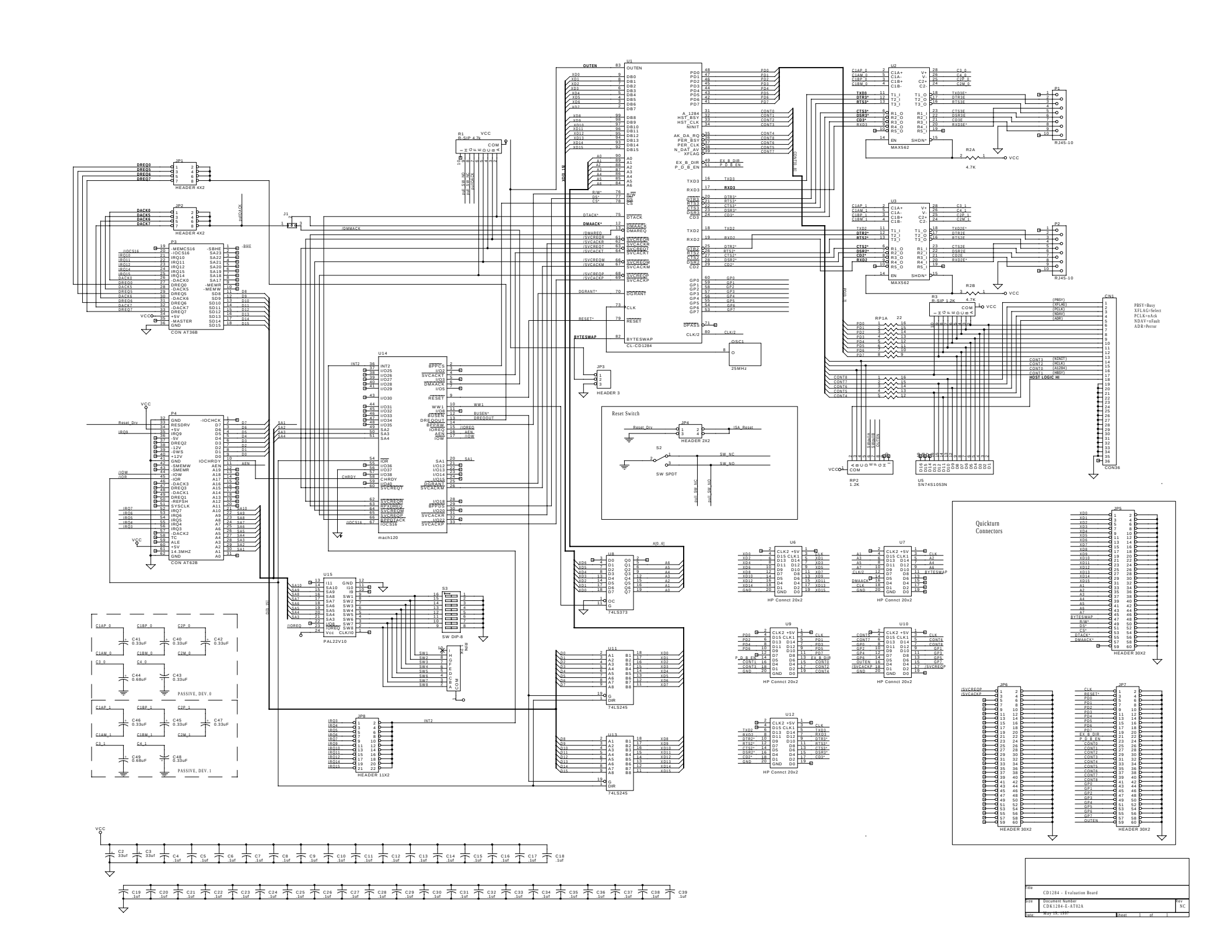
PAL Equations

```
;PALASM Design Description

;---------------------------------- Declaration Segment ------------
TITLE    ADDRESS DECODER FOR 22V10
PATTERN
REVISION 2.0
AUTHOR   NIMA TAIE-NOBARIE
DATE     06/09/97

CHIP  _1284pal2  PAL22V10

;---------------------------------- PIN Declarations ---------------
PIN  14      SA10                    COMBINATORIAL ; INPUT
PIN  15      SA9                     COMBINATORIAL ; INPUT
PIN  16      SA8                     COMBINATORIAL ; INPUT
PIN  17      SA7                     COMBINATORIAL ; INPUT
PIN  18      SA6                     COMBINATORIAL ; INPUT
PIN  19      SA5                     COMBINATORIAL ; INPUT
PIN  20      SA4                     COMBINATORIAL ; INPUT
PIN  21      SA3                     COMBINATORIAL ; INPUT
PIN  9       SW1                     COMBINATORIAL ; INPUT
PIN  8       SW2                     COMBINATORIAL ; INPUT
PIN  7       SW3                     COMBINATORIAL ; INPUT
PIN  6       SW4                     COMBINATORIAL ; INPUT
PIN  5       SW5                     COMBINATORIAL ; INPUT
PIN  4       SW6                     COMBINATORIAL ; INPUT
PIN  3       SW7                     COMBINATORIAL ; INPUT
PIN  2       SW8                     COMBINATORIAL ; INPUT
PIN  22      COMP1                   COMBINATORIAL ;
PIN  23      /IOREQ                  COMBINATORIAL ; OUTPUT

;---------------------------------- Boolean Equation Segment ------
EQUATIONS

COMP1 = (/(SA10 :+: SW1) * /(SA9 :+: SW2))*(/(SA8 :+: SW3) * /(SA7 :+: SW4))
IOREQ = COMP1 * /(SA6 :+: SW5) * /(SA5 :+: SW6)
;IOREQ = COMP1 * COMP2 * COMP3

;---------------------------------- Simulation Segment ------------
SIMULATION

;TRACE_ON IOREQ
;SETF SW1 /SW2 SW3 SW4 /SW5 SW6 SW7 SW8
;TRACE_OFF
;-----------------------------------------------------------------
```

```
;PALASM Design Description

;---------------------------------- Declaration Segment ------------
TITLE    MACH120 PLD
PATTERN
REVISION 2.0
AUTHOR   Nima Taie-Nobarie
DATE     06/06/97

CHIP  _1284PLD  MACH120

;---------------------------------- PIN Declarations ---------------
PIN  2       /BPPCS          COMBINATORIAL ; OUTPUT
PIN  4       /SVCACKT        COMBINATORIAL ; OUTPUT
PIN  6       /DMAACK         COMBINATORIAL ; OUTPUT
PIN  9       /RESET          COMBINATORIAL ; OUTPUT
PIN  10      /WW1            COMBINATORIAL ; OUTPUT
PIN  12      /BUSEN          COMBINATORIAL ; OUTPUT
PIN  13      DREQOUT         COMBINATORIAL ; OUTPUT
PIN  14      /BPPRW          COMBINATORIAL ; OUTPUT
PIN  15      /IOREQ          COMBINATORIAL ; INPUT
PIN  16      AEN             COMBINATORIAL ; INPUT
PIN  17      /IOW            COMBINATORIAL ; INPUT
PIN  20      SA1             COMBINATORIAL ; INPUT
PIN  49      SA2             COMBINATORIAL ; INPUT
PIN  50      SA3             COMBINATORIAL ; INPUT
PIN  51      SA4             COMBINATORIAL ; INPUT
PIN  60      SVCREQT         COMBINATORIAL ; INPUT
PIN  62      SVCREQR         COMBINATORIAL ; INPUT
PIN  64      SVCREQM         COMBINATORIAL ; INPUT
PIN  65      SVCREQP         COMBINATORIAL ; INPUT
PIN  66      /BPPDTACK       COMBINATORIAL ; INPUT
PIN  63      RPXDREQ         COMBINATORIAL ; INPUT
PIN  54      /IOR            COMBINATORIAL ; INPUT
PIN  36      INT2            COMBINATORIAL ; OUTPUT
PIN  56      /LOW            COMBINATORIAL ; INPUT
PIN  67      /IOCS16         COMBINATORIAL ; OUTPUT
PIN  29      /BPPDS          COMBINATORIAL ; OUTPUT
PIN  25      /DGRANT         COMBINATORIAL ; OUTPUT
PIN  26      /SVCACKM        COMBINATORIAL ; OUTPUT
PIN  33      /SVCACKP        COMBINATORIAL ; OUTPUT
PIN  31      /SVCACKR        COMBINATORIAL ; OUTPUT
PIN  58      /CHRDY          COMBINATORIAL ; OUTPUT
PIN  57      /CHRDYOE        COMBINATORIAL ; OUTPUT

;---------------------------------- Boolean Equation Segment ------
EQUATIONS

/WW1 = /AEN * IOREQ * IOW * /SA4 * /SA3 * SA2 * /SA1
BUSEN = /AEN * IOREQ * IOW + /AEN * IOREQ * IOR

IOCS16.TRST = /AEN * IOREQ
IOCS16 = LOW
```

```
RESET = /AEN * IOREQ * IOW * /SA4 * /SA3 * SA2 * SA1

BPPCS = /AEN * IOREQ * /SA4 * SA3 * /SA2 * /SA1 * IOW * /BPPDTACK +
      /AEN * IOREQ * /SA4 * SA3 * /SA2 * /SA1 * IOR * /BPPDTACK

BPPRW = IOW

BPPDS = /AEN * IOREQ * /SA4 * SA3 * /SA2 * /SA1 * IOR +
      /AEN * IOREQ * /SA4 * SA3 * /SA2 * /SA1 * IOW +
      SVCACKR + SVCACKP + SVCACKT + SVCACKM

INT2 = /SVCREQR + /SVCREQT + /SVCREQP + /SVCREQM

DGRANT = SVCACKR + SVCACKT + SVCACKP + SVCACKM

SVCACKR = /AEN * IOREQ * IOR * /SA4 * SA3 * /SA2 * SA1
SVCACKT = /AEN * IOREQ * IOR * /SA4 * SA3 * SA2 * /SA1
SVCACKP = /AEN * IOREQ * IOR * /SA4 * SA3 * SA2 * SA1
SVCACKM = /AEN * IOREQ * IOR * SA4 * /SA3 * /SA2 * /SA1

DREQOUT = /RPXDREQ

DMAACK = /AEN * IOREQ * IOW * SA4 * /SA3 * /SA2 * SA1 +
       /AEN * IOREQ * IOR * SA4 * /SA3 * /SA2 * SA1

CHRDYOE = /AEN * IOREQ * /BPPDTACK * /SA4 * SA3 * /SA2 * /SA1 * IOW +
        /AEN * IOREQ * /BPPDTACK * /SA4 * SA3 * /SA2 * /SA1 * IOR

CHRDY = LOW
CHRDY.TRST = CHRDYOE

;---------------------------------- Simulation Segment ------------
SIMULATION

;TRACE_ON INT2
;SETF SVCREQR SVCREQT SVCREQP /SVCREQM
;SETF /SVCREQR SVCREQT SVCREQP /SVCREQM
;SETF SVCREQR SVCREQT SVCREQP SVCREQM
;SETF SVCREQR SVCREQT SVCREQP SVCREQM
;SETF SVCREQR SVCREQT /SVCREQP SVCREQM
;SETF /SVCREQR /SVCREQT SVCREQP /SVCREQM
;SETF SVCREQR SVCREQT SVCREQP SVCREQM
;SETF SVCREQR SVCREQT SVCREQP SVCREQM
;SETF SVCREQR /SVCREQT SVCREQP SVCREQM
;SETF /SVCREQR /SVCREQT /SVCREQP /SVCREQM
;SETF SVCREQR SVCREQT SVCREQP SVCREQM
;SETF SVCREQR SVCREQT SVCREQP SVCREQM
;SETF /SVCREQR SVCREQT SVCREQP SVCREQM
;SETF SVCREQR SVCREQT SVCREQP SVCREQM
;SETF SVCREQR SVCREQT SVCREQP SVCREQM
;TRACE_OFF
```

```
;TRACE_ON BPPCS
;SETF SA4 SA3 SA2 SA1    /AEN IOREQ /BPPDTACK IOW
;SETF /SA4 SA3 SA2 SA1   /AEN IOREQ /BPPDTACK IOW
;SETF SA4 /SA3 SA2 SA1   /AEN IOREQ /BPPDTACK IOW
;SETF SA4 SA3 /SA2 SA1   /AEN IOREQ /BPPDTACK IOW
;SETF SA4 SA3 SA2 /SA1   /AEN IOREQ /BPPDTACK IOW
;SETF SA4 SA3 /SA2 /SA1  /AEN IOREQ /BPPDTACK IOW
;SETF SA4 /SA3 SA2 /SA1  /AEN IOREQ /BPPDTACK IOW
;SETF /SA4 SA3 SA2 /SA1  /AEN IOREQ /BPPDTACK IOW
;SETF /SA4 SA3 /SA2 SA1  /AEN IOREQ /BPPDTACK IOW
;SETF /SA4 /SA3 SA2 SA1  /AEN IOREQ /BPPDTACK IOW
;SETF /SA4 /SA3 /SA2 SA1 /AEN IOREQ /BPPDTACK IOW
;SETF /SA4 /SA3 SA2 /SA1 /AEN IOREQ /BPPDTACK IOW
;SETF /SA4 SA3 /SA2 /SA1 /AEN IOREQ /BPPDTACK IOW
;SETF SA4 /SA3 /SA2 /SA1 /AEN IOREQ /BPPDTACK IOW
;SETF /SA4 /SA3 /SA2 /SA1 /AEN IOREQ /BPPDTACK IOW
;SETF /SA4 SA3 /SA2 /SA1 /AEN IOREQ /BPPDTACK IOR
;SETF /SA4 SA3 /SA2 /SA1 /AEN IOREQ /BPPDTACK IOW
;SETF /SA4 SA3 /SA2 /SA1 AEN IOREQ /BPPDTACK IOW
;SETF /SA4 SA3 /SA2 /SA1 /AEN /IOREQ /BPPDTACK IOW
;TRACE_OFF

;TRACE_ON IOCS16
;SETF /AEN /IOREQ /LOW
;SETF /AEN /IOREQ LOW
;SETF /AEN IOREQ /LOW
;SETF /AEN IOREQ LOW
;SETF AEN /IOREQ /LOW
;SETF AEN /IOREQ LOW
;SETF AEN IOREQ /LOW
;SETF AEN IOREQ LOW
;TRACE_OFF

;TRACE_ON CHRDY BPPCS BPPDS
;SETF    /AEN /IOREQ /IOW /BPPDTACK /SA4 SA3 /SA2 /SA1
;SETF    /AEN /IOREQ /IOW /BPPDTACK /SA4 SA3 /SA2 /SA1
;SETF    /AEN IOREQ IOW /BPPDTACK   /SA4 SA3 /SA2 /SA1
;SETF    /AEN IOREQ IOR /BPPDTACK   /SA4 SA3 /SA2 /SA1
;SETF    /AEN IOREQ IOR /BPPDTACK   /SA4 SA3 /SA2 /SA1
;SETF    /AEN IOREQ IOW BPPDTACK    /SA4 SA3 /SA2 /SA1
;TRACE_OFF
;---------------------------------------------------------------------
```


Quickturn Connectors
Reset Switch
SW SPDT
HEADER 3
HEADER 2X2
HEADER 4X2
CON AT36B
CON AT62B
CON36
mach120
PAL22V10
SW DIP-8
HEADER 11X2
HEADER 30X2
HP Connct 20x2
MAX562
RJ45-10
RP2 1.2K
SN74S1053N
74LS373
74LS245
25MHz
PASSIVE, DEV.0
PASSIVE, DEV.1
CD1284 - Evaluation Board
CDK1284-E-AT02A
May 19, 1997