

The TMS320C54x DSP HPI and PC Parallel Port Interface

Application Report



The TMS320C54x DSP HPI and PC Parallel Port Interface

***Oh Hong Lye
Customer Applications Center
TI Singapore***

SPRA151
March 1997



IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Contents

Abstract	1
1 Introduction	2
2 The PC Parallel Port	3
3 Hardware Design	5
4 Software Design	9
4.1 The Bootloading Phase	9
4.2 The Kernel Phase	9
5 Summary	11
References	12
Appendix A Kernel Assembler Code	A-1

List of Figures

1 HPI Design	6
2 HPI Timing Diagram	7
3 State Diagram for HCNTL1 HCNTL0	8

List of Tables

1 PC Parallel Port Pins	3
2 PC Parallel Port R/W Register	4
3 Parallel Port Pins to HPI	5

The TMS320C54x DSP HPI and PC Parallel Port Interface

ABSTRACT

The TMS320C5x and TMS320C54x family of DSPs contains a host port interface (HPI) functional block. This block interfaces to the microcontroller unit (MCU) without external logic hardware. The interface relieves the DSP of the handshake process necessary for communication with the MCU and provides the user with the additional MIPS for actual computations. With minimal interface logic, the HPI interfaces to the host (PC parallel port) in a bi-directional mode. This application report describes the hardware considerations for the interface design. A state machine resolves the problem of insufficient signals on the parallel port side. The report describes the software bootloading sequence and kernel software for parallel communications.

1 Introduction

The host port interface (HPI) unit is a peripheral to the TI TMS320 fixed point DSP family, specifically the TMS320C57, TMS320C542, TMS320C545, and TMS320C548. The unit allows the DSP to interface with an MCU containing a single or dual data strobe, a separate or multiplex address bus, or a separate or multiplex data bus. The HPI communicates with the host independently of the DSP. The HPI features allow the host to interrupt the DSP, or vice versa when required. The interfaces contain minimal external logic, so a system with a host and a DSP is designed without increasing the hardware on the board. The HPI interfaces to the PC parallel ports directly, with simple and minimal hardware.

2 The PC Parallel Port

The HPI interfaces to the PC parallel port (bi-directional mode) in a straightforward manner. The data pins are used for data transfer and the control pins are used as input/output (I/O) for handshaking and read/write (R/W) protocols. Table 1 provides a description of the parallel port pins and the port R/W registers.

Table 1. PC Parallel Port Pins

I/O	DB25 PIN	CENTRONICS PIN	SIGNAL NAME	REGISTER BIT
O	1	1	–STROBE	C0–
O	2	2	DATA0	D0
O	3	3	DATA1	D1
O	4	4	DATA2	D2
O	5	5	DATA3	D3
O	6	6	DATA4	D4
O	7	7	DATA5	D5
O	8	8	DATA6	D6
O	9	9	DATA7	D7
I	10	10	–ACK	S6+IRQ
I	11	11	+BUSY	S7–
I	12	12	+PAPEREND	S5+
I	13	13	+SELECTIN	S4+
O	14	14	–AUTOFD	C1–
I	15	32	–ERROR	S3+
O	16	31	–INIT	C2+
O	17	36	–SELECT	C3–
N/A	18-25	19-30	GND	
N/A		33	GND	
N/A		16, 17	GND	

NOTE: The I/O column is defined from the PC viewpoint.

Table 2 shows the PC parallel port R/W register.

Table 2. PC Parallel Port R/W Register

PORT	R/W	I/O ADDRESS	BITS	FUNCTION
Data out	W	Base + 0	D0 – D7	Eight TTL outputs
Status in	R	Base + 1	S3 – S7	Five TTL inputs
Control out	W	Base + 2	C0 – C3	Four TTL open collector outputs
Control out	W	Base + 2	C4	Internal, IRQ enable
Control out	W	Base + 2	C5	Internal, Tristate data (PS/2)
Data feedback	R	Base + 0		Matches data out
Control feedback	R	Base + 2	C0 – C3	Matches control out
Control feedback	R	Base + 2	C4	Internal, IRQ enable readback

3 Hardware Design

The HPI connects to the host (PC parallel port in the bi-directional mode) through the data bus of the HPI. The control and status pins are used as bit I/O. These I/O bits provide the handshaking and control for communications with the HPI. The pin names, such as ACK or BUSY, are not significant. Table 3 shows the use of the parallel port pins for communication with the HPI.

Table 3. Parallel Port Pins to HPI

PARALLEL PORT	HPI	FUNCTION
D0 – D7	HD0 – HD7	Data bus
ACK	HINT	Host interrupt
BUSY/PAPEREND†		No connection
SELECT	HRDY	Host ready pin
AUTOFD†	HCNTL0	Access mode control
ERROR	HBIL	HBIL status feedback
INIT	HRW	R/W control strobe
SELECTIN†	HCNTL1	Access mode control

† The parallel port generates a signal on these pins to the HPI through external hardware. HCS is set to ground. HAS and HDS are connected to V_{dd} .

The HPI pins and their functions are described as follows:

- The HPI data bus uses HD0 - HD7.
- The host interrupt, HINT, is controlled by a HINT bit in the HPIC register. HINT goes low when the DSP writes a 1 to this bit. The bit is read as a 1 by the DSP and host. If the host writes a 1 to this bit, HINT goes high and the DSP and host read this bit as a 0. When the DSP requires the attention of the host, the DSP signals the host using this bit.
- HRDY is a DSP output indicating the DSP is ready for data transfer.
- HCNTL0 and HCNTL1 are the control signals. These signals indicate which transfer to complete. The transfer types are data, address, etc.
- HBIL low indicates the current byte is the first byte; HBIL high indicates the second byte.
- HRW high indicates the host is doing a read; HRW low indicates a write.

Figure 1 shows the HPI design. The host accesses data by an R/W control pin, HRW, and a data strobe, HDS1. HDS2 is tied to V_{dd} . The strobe, HDS1, inputs the data at the rising edge of the signal.

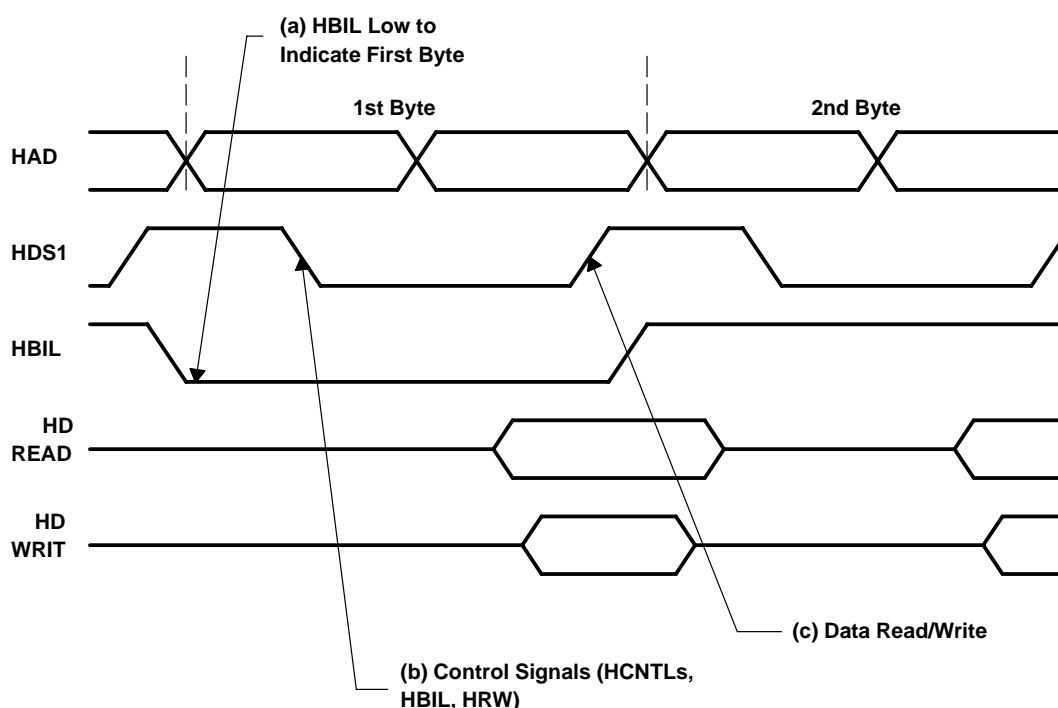
The DSP is reset by generating clock edges from AUTOFD.



Data is strobed into the HPI data/address register on the rising edge of HDS1. The HBIL input indicates whether the byte received is the first (low) or second (high) byte. HBIL transitions during the rising edge of HDS1, at a point later in time than HDS1. This delay is the propagation delay of the flip-flop (see Figure 2). The flip-flop ensures that the correct value of HBIL is sampled for every byte transferred.

To ensure HBIL is initialized high at power up, a resistor and capacitor are connected to the flip-flop. The resistor and capacitor are not necessary if HBIL is fed back to the host for software initialization. The software monitors the HBIL input to ensure the validity of a data/address transfer.

The timing diagram of Figure 2 shows the control signals and data strobed in and out of the HPI. HAD, HD Read and HD Write are the read and write data signals.[1] The signals show the data transferred across the HPI data bus (HD0–HD7).



Note: HAD stands for HCNTL0, HCNTL1, HBIL and HRW.

Figure 2. HPI Timing Diagram

At time (a), the control signals, HCNTL1, HCNTL0, HRW, and HBIL are set to the required logic level. This level indicates the type of access necessary, whether read or write, and the appropriate registers.

At time (b), the falling edge of HDS1 causes the control signals to latch into the HPI and set the HPI to the required mode.

At time (c), the rising edge of HDS1 indicates data is being written to the HPI or read from the HPI.

The two control signals, HCNTL1 and HCNTL0 combine to form the state diagram of Figure 3.

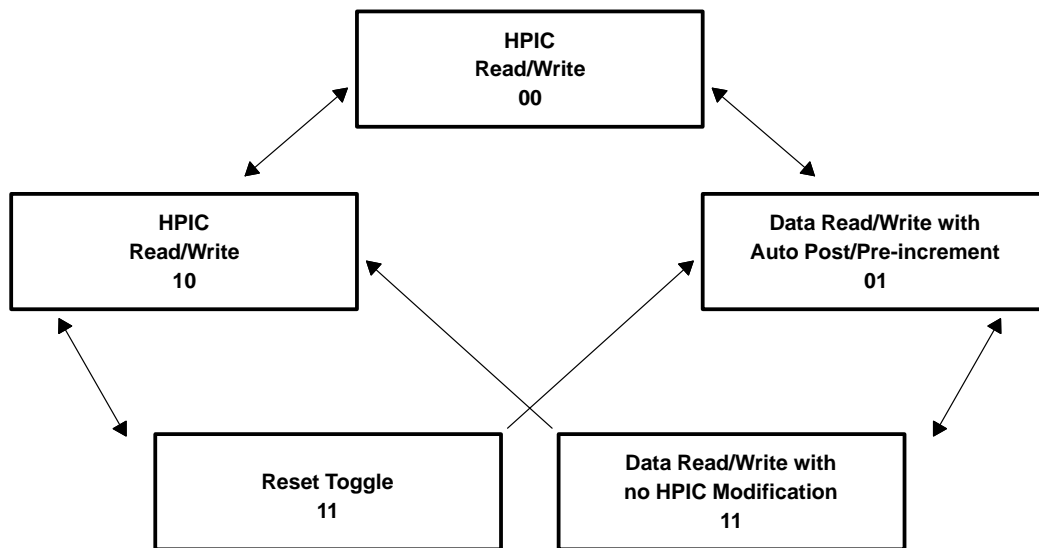


Figure 3. State Diagram for HCNTL1 HCNTL0

In Figure 3, state 11 corresponds to either of the possible instances, reset (RS) toggle or data read/write. Special care is necessary when entering state 11 so the user can determine which of the two functions the system is executing.

A reset is performed upon system initialization or reinitialization. Reinitialization occurs when the system encounters a fatal error. The reset function is not necessary during normal operation.

The reset toggle state occurs when HCNTL0 transitions from 0 to 1 and HCNTL1 is 1. This state occurs when HCNTL1 is the input to the J-K flip-flop and the input HCNTL0 is the inverted clock of the flip-flop.

If the system access to these states is changed so state 11 is always entered through state 01, the reset toggle state is never entered. This feature is easily implemented on the host side. The program access to the various states is a fixed sequence such as, 00 to 01 to 11 to 10 to 00 and so forth. This program sequence makes the system go through state 01 when transitioning from state 00 to 10.

4 Software Design

The software design for the HPI-to-PC parallel port interface is divided into two parts: the bootloading phase and the kernel phase.

4.1 The Bootloading Phase

To select the HPI bootload function, the HINT output is connected to the INT2. The boot routine first pulls the HINT low and then checks to see if the INT2 interrupt is active. If INT2 is active, the program branches to the HPI boot routine sequence. This program branch is a jump to location 1000H.

Since the HPI boot routine branches to location 1000H, the user code (communication kernel) must be loaded into the HPI RAM during the DSP reset. During the DSP reset, the HPI is operating in the host-only mode (HOM). The HOM allows the host to access the HPI RAM even if the DSP clock input is stopped.

After the kernel loads, the DSP is taken out of reset and the kernel (loaded code) executes.

4.2 The Kernel Phase

The kernel is a collection of data transfer routines. The basic set of routines that comprise the kernel are:

- Communication protocol
- Data transfer from program memory to PC
- Data transfer from data memory to PC
- Data transfer from PC to program memory
- Data transfer from PC to data memory
- Program execution

Not all the routines from the basic set are necessary for an application. For example, a user may require more than one routine for data transfer between the host and program memory while not requiring a routine for the transfer of data from program memory to the host. One essential routine is the communication protocol. This routine is necessary so the host and the DSP are able to exchange information such as the kernel command to execute, starting address, amount of data to be transferred, etc.

The TMS320C54x DSK debugger kernel uses four pointers. These pointers contain the locations of the command word, start address, data length, and data buffer. The kernel polls the DSPINT bit in the HPIC register of the TMS320C54x (see Appendix A). This polling checks to see if the host software is trying to establish a communication link. If the DSPINT bit is set, the kernel software fetches a command word. The command word from the host is decoded and the DSP performs the required operations.

Before the host interrupts the DSP by setting the DSPINT, the PC loads the required information, such as command word, start address, and data, by means of the HPI. After the load is completed, the PC sets the DSPINT bit in the HPIC. The set bit causes the DSP to fetch the command word and execute the defined operations.

See Appendix A for the kernel source code.

5 Summary

This application report describes the hardware interface design between the host (PC parallel port in the bi-directional mode) and the HPI using the TI TMS320C54x DSP. The report gives a brief description of the software bootload sequence and the debugger kernel. Appendix A provides the source kernel code.

References

1. *TMS320C54x, TMS320LC54x, TMS320VC54x Fixed-Point Digital Signal Processors*, Texas Instruments, 1996, Literature Number SPRS039.
2. *TMS320C5x DSP Starter kit User's Guide*, Texas Instruments, 1994, Literature Number SPRU101.

Appendix A Kernel Assembler Code

```

;File:      Host.asm -> basic monitor kernel for TMS320C542
;
Host Port Interface
ESC        .set      01bh
           .width    100
           .length   55
           .title    "TMS320C542 Comms Kernel loaded via Host
                    Port Interface"

           .mmregs

VERSION    .set      0001h          ; 01 version
           .def      tmp, buffptr, scratch, command, startadd,
                    length, dump
           .def      main, start, hack, lddm, ldld, ldldp, ldpm, exec
           .def      special, trapx
           .bss      buffptr,1      ; add=60h    ;not used
           .bss      tmp,1          ; add=61h    ;not used
           .bss      scratch,1      ; add=62h    ;not used
           .bss      usp,1          ; add=63h
           .bss      blank, 3       ; add=64h to 66h not used
           .bss      STACK,12       ; add=67h to 72h
           .bss      TMPSTK,12      ; add=73h to 7eh
           .bss      PC, 1          ; add=7fh
command    .usect    "COMMS", 512,1 ; add=1200h
startadd   .set      command+1
length     .set      command+2
dump       .set      command+3

;-----
; HOST ACKNOWLEDGE: Ensure that Host has acknowledged end of task
;-----
HOSTACK    .macro
hack        ldm      hpic, a          ; load accumulator with
                    ; HPI control word
            and      #08h, a          ; mask out all bits except
                    ; hint
            bc       hack, aneq       ; wait for hint to go
                    ; high/bit
                    ; goes low
            ret
        .endm

;-----
; DPM: DUMP PROG MEM TO PC
;-----
DPM        .macro
            ld        startadd,0, a    ; store starting address in
                    ; ACCA
            mvdm      length, ar7      ; store length in ar7
            stm       #dump, ar5       ; store dump address ar5
            nop
            nop
            nop

```

```

        loop?   reada  *ar5+      ; copy a word from prog to HPI
                                ; buffer
        add     #1,0,a            ; startadd++
        banz    loop?,*ar7-      ; repeat for whole blk of data
        stm     #0ah, hpic       ; interrupt host(shared) to
                                ; indicate end of task
        call    hack             ; ensure host has acknowledged
        ret
                                .endm
;-----
; DDM: DUMP DATA MEM TO PC
;-----
DDM      .macro
        mvdm    startadd, ar6    ; store starting address in
                                ; ar6
        mvdm    length, ar7      ; store length in ar7
        stm     #dump, ar5       ; store dump address ar5
        nop
        nop
loop?    ld      *ar6+,0,a        ; ACCA=content of loc pointed
                                ; by ar6
        stl     a, 0, *ar5+      ; store ACCA in HPI buffer
        banz    loop?,*ar7-      ; tx whole block
        stm     #0ah, hpic       ; interrupt host(shared) to
                                ; indicate end of task
        call    hack             ; ensure host has acknowledged
        ret
                                .endm
;-----
; DLD: DOWNLOAD DATA FROM PC TO DSP
;-----
DLD      .macro
        mvdm    startadd, ar6    ; store starting address in
ar6      ; ar6
        mvdm    length, ar7      ; store length in ar7
        stm     #dump, ar5       ; store dump address ar5
        nop
        nop
loop?    ld      *ar5+, 0, a      ; received wrd is in ACCA
        stl     a,*ar6+          ; store to loc pointed to by
                                ; ar6
        banz    loop?,*ar7-      ; dwnld whole blk
        stm     #0ah, hpic       ; interrupt host(shared) to
                                ; indicate end of task
        call    hack             ; ensure host has acknowledged
        ret
                                .endm
;-----
; DLP: DOWNLOAD PROG FROM PC TO DSP
;-----
DLP      .macro
        ld      startadd,0,a     ; store starting address in
                                ; ACCA

```

```

        mvdw    length, ar7    ; store length in ar7
        stm     #dump, ar5     ; store dump address ar5
        nop
        nop
loop?:   writa   *ar5+          ; copy wrd from data at ar5 to
                                ; the prog add
        add     #1,0,a         ; acca++
        banz    loop?,*ar7-    ; transfer whole blk
        stm     #0ah, hpic     ; interrupt host(shared) to
                                ; indicate end of task
        call    hack           ; ensure host has acknowledged
        ret
                                .endm
;-----
; DMPREG: dump register: context save to system stack in
; scratchpad RAM
;-----
DMPREG .macro
trapx   ssbx    intm           ; disable all interrupts
        pshm    bl            ; old value of bl in user stack
        ldm     sp,b          ; save user sp to BL
        stm     #STACK+12,sp  ; restore system stack
        pshm    st0
        pshm    st1
        pshm    tim
        pshm    ar5
        pshm    ar6
        pshm    ar7
        pshm    ag
        pshm    ah
        pshm    al
        stlm    b,sp          ; restore user sp to SP
        nop
        nop
        popm    bl            ; retrieve old value of bl
        ld      #0, dp        ; set dp to page 0
        popm    al
        stl     A, PC
        ldm     ifr, a         ; clearing DSPINT
        or      #0204h,0,a     ; & int 2
        stlm    a,ifr         ;
        ld      #command, dp   ; setting dp to command area
        stm     #0ah, hpic     ; interrupt host(shared) to
                                ; indicate end of task
        call    hack           ; ensure host has acknowledged
                                .endm
;-----
; EXECUTE: execute a program from given address
;-----
execute .macro
        popm    al            ; return address to be overwritten
        ld      startadd,0,a   ; store starting address in ACCA
        nop                                ; latency problem

```

```

                pshm    al                ; addr for restart of user prog
                ld      #0, dp            ; is in user stack
                ; setting dp to 0 to access vars

                ldm     sp,a              ; save user stack pointer to AC-
CAL
                stl     a,usp              ; store user sp to usp
                stm     #STACK+3,sp       ; ld SP with top of system stack
                stm     #0ffffh, ifr
                popm    al
                popm    ah
                popm    ag
                popm    ar7
                popm    ar6
                popm    ar5
                popm    tim
                popm    stl
                popm    st0
                mvdm    usp,sp            ; restore user sp to SP
                nop
                rete

                .endm

;-----
; BEGIN OF MAIN PROGRAM
;-----

                .text
; the following is the int vector table which will be placed in
; 1000h by setting the IPTR of the PMST register
                b       start            ; 00;reset
                .space  2*16
                .space  4*16              ; 02;nmi
                .space  56*16             ; 04;software int
int0            rete                    ; 40;int0
                .space  3*16
int1            rete                    ; 44;int1
                .space  3*16
int2            rete                    ; 48;int2
                .space  3*16
tint            rete                    ; 4c;timer int
                .space  3*16
                .space  16*16             ; 50;BSP & HPI RX and TX ready
                ; int
int3            rete                    ; 60;int3
                .space  3*16
HPINT          b       trap              ; 64;HPI int
                .space  2*16              ; 67;END
start          ssbx    intm              ; disable all interrupts
                ld      #0,dp
                stm     #0ffffh, ifr      ; clear interrupt flag register
                stm     #0200h, imr        ; enable only DSPint
                st      #0000h, 72h        ; initialise ST0=0
                st      #2A00h, 71h        ; initialise ST1=#2a00
                rsbx    sxm                ; default is reset sign

```

```

stm      #TMPSTK+12,sp      ; initialise SP to temp
                        ; stack of 79h
ld        #1020h,0,a        ; to set IPTR of PMST to
                        ; point to 1000h

stlm     a,pmst
stm      #0,swwsr          ; no wait states
call     hack              ; check for host
                        ; acknowledgement to begin
stm      #0ah, hpic        ; interrupt host(shared)
                        ; indicate ready to start
ld        #command,dp      ; setting dp to command
                        ; area
redo     ssbx      intm      ; disable all interrupts
rsbx     xf
ldm      ifr, a            ; load accumulator with
                        ; interrupt
                        ; flag register
and      #0200h, a        ; mask out all bits except
                        ; DSPINT
cc        main, aneq       ; wait for DSPINT to go
                        ; high implies that an
                        ; interrupt occurred

main     ssbx      xf
b        redo             ; loop back
ldm      ifr, a           ; clearing DSPINT
or       #0204h,0,a       ; & int 2
stlm     a,ifr
mvd      command, ar5     ; store command word in
                        ; ar5
nop
nop
nop
;execute commands
lddm     banz      ldpm,*ar5- ; dump data mem      ;0
DDM
ldpm     banz      ldld,*ar5- ; dump prog mem      ;1
DPM
ldld     banz      ldlp,*ar5- ; load data to DSP   ;2
DLD
ldlp     banz      exec,*ar5- ; load prog to DSP   ;3
DLP
exec     banz      special,*ar5- ; execute to single step
                        ; or breakpt      ;4
EXECUTE
HOSTACK
trap     DMPREG
b        redo             ; wait for commands
special  ret              ; branch back if invalid
                        ; code
len      .set      $-1000h  ; length of monitor kernal
                        ; THE END OF KERNAL

```

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.