

DSP563xxEVM
Host Interface Adapter



Reference Manual

A Product of Domain Technologies, Inc.

DSP Development Systems

© 1997-2003, Domain Technologies, Inc.
811 East Plano Pkwy, Ste 115, Plano, TX 75074
Tel.: (972) 578-1121 • Fax: (972) 578-1086
Email: info@domaintec.com
<http://www.domaintec.com>
September, 2003

TABLE OF CONTENTS

CHAPTER 1: DSP563xxEVM ISA Host Interface	5
PC ISA Interface Card.	5
Base I/O Address Selection	6
Active Termination Adaptors	6
ISA Interface Pal Equations	7
ISA Controller Circuit Diagram	10
DSP5630xEVM Active Termination Circuit Diagram.	11
ISA Interface I/O Port Equates.	12
CHAPTER 2: DSP563xxEVM Printer Port Interface - Type A . . .	13
Printer Port I/F Pal Equations	14
Printer Port Interface Circuit Diagram	15
Printer Port Interface I/O Port Equates	16
CHAPTER 3: DSP563xxEVM Printer Port Interface - Type B . . .	19
Printer Port I/F Pal Equations	21
Printer Port Interface Circuit Diagram	24
Printer Port Interface I/O port Equates.	25

CHAPTER 1: DSP563xxEVM ISA Host Interface

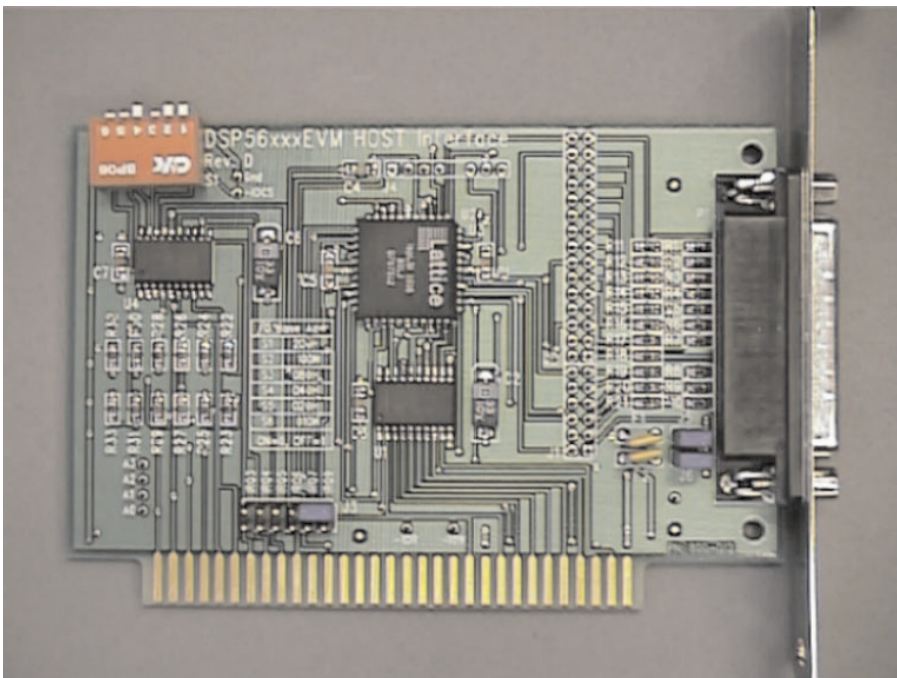
ISA host interface comprises of three parts:

- PC ISA interface card
- Ribbon cable
- DSP5630xEVM active termination

1

PC ISA Interface Card

An 8-bit ISA controller provides high-speed access to the EVM Host Port. Controller card supplies all power required for EVM operation. To prevent equipment damage both +5VDC and +12VDC are protected with resettable fuses. All control signals are driven through 150 ohm resistors allowing hot connection of the interface to the PC without powering down the host computer.



Base I/O Address Selection

Base I/O address of the ISA interface card is set with six position dip-switch on top of the card. Switch in the ON position sets bit value to 0.

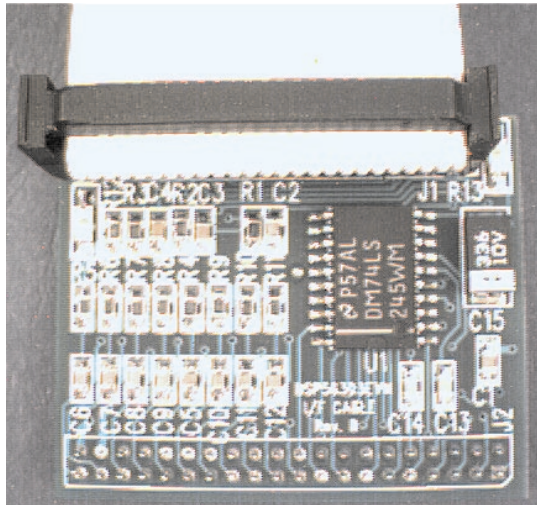
1

Position	1	2	3	4	5	6
Value	0x200	0x100	0x080	0x040	0x020	0x010

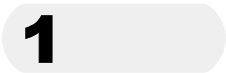
For example, to set base I/O address to 0x240:
 switch 1 and 4 are OFF (up)
 switch 2, 3, 5 and 6 are ON (down)

Active Termination Adaptors

Active termination adaptors connect directly to the Host Port signals of the DSP5630x, so extreme care should be used when connecting or disconnecting the interface. Attaching or disconnecting of the active termination board should be executed with the ribbon cable disconnected from the host PC. Also, external power supply should be disconnected from the EVM, as it will not be needed after connecting the EVM to the PC.



ISA Interface Pal Equations



```

=====
"| HIEVM56k - Host Interface for DSP56xxxEVM |
"| (c) 1999, Domain Technologies, Inc. |
"| http://www.domaintec.com |
"| support@domaintec.com |
"| |
"| Equations for the isp1016LT44 |
"| Revision History |
"| 01-07-99 SLH Rev 1.0000 |
=====

```

```

MODULE HiEvmD
TITLE 'HiEvmD'
"Inputs
  HD1,HD2,HD3,HD4      pin 14,15,16,13;          "data bits from ISA
  AA0,AA1,AA2,AA3     pin 19,20,21,33;          "address lines from ISA
  !IOCS,!IOW, !IOR    pin 34,25,26;          "decoded CS, R/W from ISA
  BCLK,RESDRV         pin 5,11;              "bus clock, reset from ISA
  !HREQ               pin 4;                 "host request from DSP
"Outputs
  DIR,!ENA            pin 24,23;              "PC buffer control
  !RSTOUT             pin 10;                 "Power on reset for isp1016
  !ZEROWS,!INTA      pin 22,9;              "ISA ctrl
  HA0,HA1,HA2        pin 1,2,3;              "buffered PC addr
  !HEN, !HRDWR       pin 44,43;             "DSP Host access
  HMODA,HMODB,HMODC  pin 35,36,37  istype 'reg,buffer';
  HRESET             pin 12  istype 'reg,buffer';
  !HACK              pin 42  istype 'reg,buffer';
  HCTRL              pin 38  istype 'reg,buffer';
  HCS                pin 41  istype 'reg,buffer';
  //BALE,AEN,TC      pin 40,31,32;          "Unused pins
"Outputs that don't leave this logic block (buried nodes)
  ENAOWS,IRQEN,IRQINV node istype 'reg';
  STRTIOW, ENDIOW    node istype 'reg,buffer';
"Set declarations
  DSP_ACCESS = IOCS & !AA3;                "RW DSP at address 0..7
  ADDR_8     = AA3 & !AA2 & !AA1 & !AA0;
  ADDR_9     = AA3 & !AA2 & !AA1 & AA0;
  ADDR_A     = AA3 & !AA2 & AA1 & !AA0;
  LATCH_MODE = IOCS & IOW & ADDR_8;        "Latch reset and mode bits
  LATCH_OPT  = IOCS & IOW & ADDR_9;        "Latch PC options
  LATCH_OPTB = IOCS & IOW & ADDR_A;        "Latch DSP options
Equations
  RSTOUT      = RESDRV;                    "Pin definitions will invert
  DIR         = IOW;                        " Direction of the 245
  ENA         = LATCH_MODE                  " OEENABLE of the 245

```

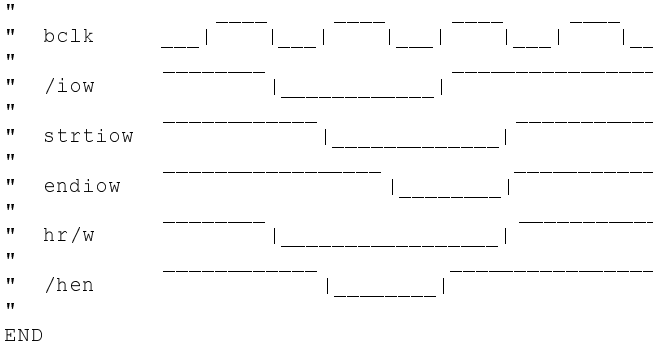
1

```

        # LATCH_OPT           "   when writing to pld latch
        # LATCH_OPTB          "   when writing to pld latch B
        # HEN;                 "   or when read/writing to dsp
    HA0      = AA0;            "Drive buffered address line
    HA1      = AA1;            "Drive buffered address line
    HA2      = AA2;            "Drive buffered address line
"Zero WS is enabled on all IO accesses as a default
    ZEROWS   = 1;
    ZEROWS.OE = ENA0WS & IOCS & IOR
                # ENA0WS & IOCS & IOW;
"Optional Interrupt to the PC
    INTA     = 1;                "Int will drive pin active (low)
    INTA.OE  = IRQEN & HREQ & !IRQINV "Tristate during non-interrupt
                # IRQEN & !HREQ & IRQINV;
"Define a 4 bit mode latch
    HRESET   := HD4;
    HMODC    := HD3;
    HMODB    := HD2;
    HMODA    := HD1;
    HRESET.C = !LATCH_MODE;     "Note LATCH_MODE has a !, so rising
    HMODC.C  = !LATCH_MODE;     "   clock edge will occur when IOW is
    HMODB.C  = !LATCH_MODE;     "   negated in LATCH_MODE.
    HMODA.C  = !LATCH_MODE;
"Define a 3 bit PC options latch
    ENA0WS   := HD1;            " 0=zero wait state disabled, 1=enabled
    IRQEN    := HD2;            " 0=interrupts to PC are disabled
    IRQINV   := HD3;            " 0=INT with HREQ, 1=INT with !HREQ
    ENA0WS.C = !LATCH_OPT;
    IRQEN.C  = !LATCH_OPT;
    IRQINV.C = !LATCH_OPT;
"Define a 4 bit DSP options latch
    HACK     := HD1;
    HCS      := HD2;
    HCTRL    := HD3;
    HACK.C   = !LATCH_OPTB;
    HCS.C    = !LATCH_OPTB;
    HCTRL.C  = !LATCH_OPTB;
"Logic for !HEN and HR/W timing
    STRTIOW  := IOW;            " Delay start of IOW by 1 positive clock edge
    STRTIOW.C = BCLK;           " Use POSITIVE bclk clock edge
    ENDIOW   := IOW;
    ENDIOW.C = !BCLK;           " Use negative clock edge to end iow cycle
    HEN      = DSP_ACCESS & IOR
                # DSP_ACCESS & IOW & STRTIOW; "delay start of HEN by 1/2 bclk
                " to allow HRDWR to propogate.
    HRDWR    = IOW
                # ENDIOW;       " begin WR signal when HIOW goes low
                " and hold until delayed HIOW is complete.
    " Use NEGATIVE bclk edge

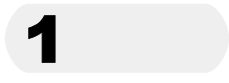
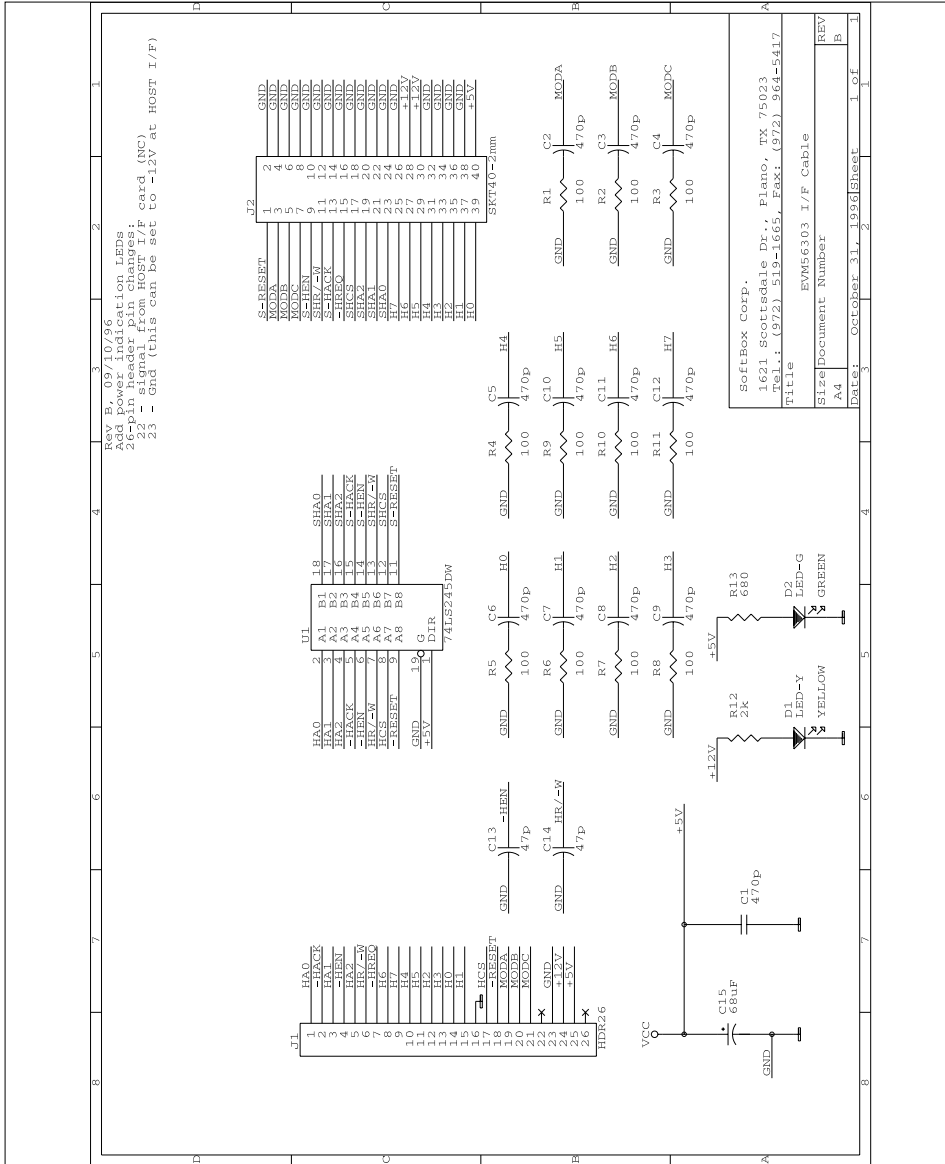
```

" On IO writes, this will produce a /HEN of 1 BCLK (125ns) when configured as
 " ZEROWS. When ZEROWS isn't enabled, /HEN will be 4 BCLKS. On IO reads, /HEN
 " will always be 1/2 BCLK longer, as the first half of the first BCLK will be
 " enabled. Here is the diagram for a IO cycle with ZEROWS enabled.



1

DSP5630xEVM Active Termination Circuit Diagram



ISA Interface I/O Port Equates

1

```

/* Bits definition for the DSP mode latch */
#define MODA      0x02
#define MODB      0x04
#define MODC      0x08
#define RESET     0x10

/* Bits definition for the PC interface latch */
#define ENA0WS    0x02
#define IRQEN     0x04
#define IRQINV    0x08

/* Bits definition for the DSP control latch */
#define HACK      0x02
#define HCS       0x04

/* I/O address definitions */
#define HOST_ICR   (IoBase+0)
#define HOST_CVR   (IoBase+1)
#define HOST_ISR   (IoBase+2)
#define HOST_IVR   (IoBase+3)
#define HOST_HIGH  (IoBase+5)
#define HOST_MID   (IoBase+6)
#define HOST_LOW   (IoBase+7)
#define HOST_MODE  (IoBase+8)
#define HOST_PC    (IoBase+9)
#define HOST_DSP   (IoBase+10)
int resetHost (int IoBase)
{
    IoWrite (HOST_PC, 0);           /* Clear PC ctrl */
    IoWrite (HOST_DSP, 0);         /* Clear DSP ctrl */
    IoWrite (HOST_MODE, MODA+MODB_MODC); /* Assert reset */
    Delay (50);                   /* Wait 50 ms */
    IoWrite (HOST_MODE, RESET+MODA+MODB_MODC); /* Deassert reset */
    Delay (50);                   /* Wait 50 ms */
    if (IoRead (HOST_IVR) != 0x0f) /* Check IVR value */
        return FALSE;
    return TRUE;
}

```

CHAPTER 2: DSP563xxEVM Printer Port Interface - Type A

Printer port interface allows for direct connection to the bi-directional parallel port of IBM PC or compatible computers. This interface allows for moderate speed data transfers between DSP and host PC. Supported are two modes of operation:

- Bi-directional
- EPP Enhanced Printer Port

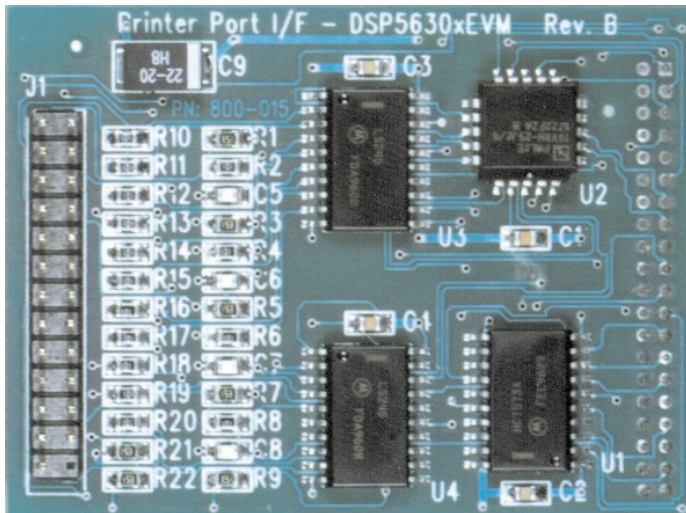
The “nibble” mode is not supported.

Computers equipped with the ECP interface have to be configured as EPP port.

Every access to the DSP’s host port consists of two steps:

- Write address/control bits
- Read/write data

2



The printer port interface does not allow +5V power supply to the EVM. The EVM needs to be powered from the standard power supply when the printer port interface is in use.

Printer Port I/F Pal Equations

```

name      HiPP_a;
partno    460-002;
date      10/29/96;
revision  A;
designer   Peter Koenig;
company   SoftBox Corp.;
assembly  PP30xEVM;
location  U2;
device    g16v8;
format    -ja;
    
```

2

```

PIN [1..3]   =                               !WRITE, !DATASTB, !RESET;
PIN [4..8]   = !ADDRSTB, HREQ,  HACK,  INTENA,  INTMODE;
PIN [9..13]  = NC9,      Gnd,  NC11,  !HEN,    HRW;
PIN [14..18] = LATCH,    BHREQ, BHACK,  DIR,    ENA;
PIN [19..20] = WAIT,     Vcc;

DIR  = !WRITE;                               /* Direction of the 245 */
ENA  = DATASTB # ADDRSTB;                   /* Enable of the 245   */
LATCH = WRITE & ADDRSTB;                      /* Write addr/ctrl bits */
HEN   = DATASTB;                             /* access host port    */
HRW   = !WRITE;                               /* Read or write host port */
WAIT  = DATASTB # ADDRSTB;                   /* handshake with the EPP */
BHREQ = INTENA & INTMODE & HREQ              /* Interrupt to the PP  */
      # INTENA & !INTMODE & !HREQ;
BHACK = HACK;                                /* status signal to the PP */
    
```


Printer Port Interface I/O Port Equates

2

```

#define DATA_PORT      (unsigned short)(IoBase + 0)
#define STATUS_PORT     (unsigned short)(IoBase + 1)
#define CTRL_PORT       (unsigned short)(IoBase + 2)
#define EPP_ADDR        (unsigned short)(IoBase + 3)
#define EPP_DATA        (unsigned short)(IoBase + 4)
/* bit definition for the PP control register (CTRL_PORT) */
#define HOST_WRITE      0x01      /* Controls HR/-W and direction of the buffer */
#define DATA_STROBE   0x02      /* Controls -HEN */
#define ADDR_STROBE    0x08      /* Controls address LATCH */
#define RESET_ON       0x00      /* Reset asserted */
#define RESET_OFF      0x04      /* Reset deasserted */
#define IRQ_ENA        0x10      /* IRQ to the PC from the PP */
#define SPP_READ_ENA   0x20      /* Direction of the PP data buffer (1 - read) */
#define SPP_WRITE      (HOST_WRITE+RESET_OFF)
#define SPP_READ       (SPP_READ_ENA+RESET_OFF)
/* signals on the ADDR latch (PP interface */
#define HOST_A0        0x01
#define HOST_A1        0x02
#define HOST_A2        0x04
#define HOST_MODA      0x08
#define HOST_MODB      0x10
#define HOST_MODC      0x20
#define INT_ENA        0x40
#define INT_MODE       0x80
#define HOST_BOOT      (HOST_MODA+HOST_MODB+HOST_MODC)
#define HOST_ICR       (0)
#define HOST_CVR       (HOST_A0)
#define HOST_ISR       (HOST_A1)
#define HOST_IVR       (HOST_A0+HOST_A1)
#define HOST_HIGH      (HOST_A0+HOST_A2)
#define HOST_MID       (HOST_A1+HOST_A2)
#define HOST_LOW       (HOST_A0+HOST_A1+HOST_A2)

int resetEpp (int IoBase){
    IoWrite (CTRL_PORT, RESET_ON);          /* Assert reset */
    IoWrite (EPP_DATA, HOST_IVR); Delay (50); /* Set Addr Latch to IVR */
    IoWrite (HOST_CTRL, RESET_OFF); Delay (50); /* Deassert reset */
    return (IoRead (EPP_DATA) == 0x0f)      /* Check IVR value */
}

int resetSpp (int IoBase){
    int value;

```

```
IoWrite (CTRL_PORT, RESET_ON);           /* Assert reset          */
IoWrite (DATA_PORT, HOST_IVR); Delay (50); /* Set Addr Latch to IVR */
IoWrite (CTRL_PORT, SPP_WRITE); Delay (50); /* Prepare for ADDR write*/
IoWrite (CTRL_PORT, SPP_WRITE+ADDR_STROBE); /* Write ADDR latch      */
IoWrite (CTRL_PORT, SPP_WRITE);           /* End strobe ADDR latch */
IoWrite (CTRL_PORT, SPP_READ);            /* Prepare for read      */
IoWrite (CTRL_PORT, SPP_READ+ADDR_STROBE); /* Strobe DATA port    */
value = IoRead (DATAPORT);                /* Read IVR              */
IoWrite (CTRL_PORT, RESET_OFF);           /* End read cycle       */
return (value == 0x0f)                    /* Check IVR value      */
}
```

2

2

CHAPTER 3: DSP563xxEVM Printer Port Interface - Type B

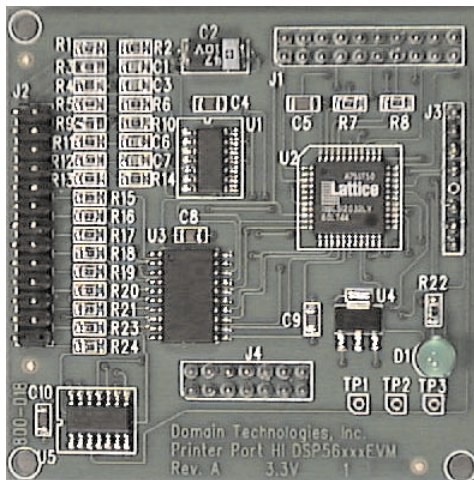
Printer port interface allows for direct connection to the bi-directional parallel port of IBM PC or compatible computers. This interface allows for fast speed real-time data transfers between DSP and host PC. Supported are two modes of operation:

- SPP (nibble mode)
- Bi-directional
- EPP Enhanced Printer Port

Computers equipped with the ECP interface have to be configured into EPPmode.

Every access to the DSP's host port consists of two steps:

- Write address/control bits
- Read/write data



3

Type B adaptor features “autoincrement” mode. Enabling this mode increases the transfer speed twice. After setting “autoincrement” bit, adaptor cycles HOST port accesses through addresses: 2 (ISR), 5 (High Data), 6 (Mid Data) and 7 (Low Data).

The printer port interface operates on 3.3 V power provided by the EVM.

Conection to the EVM is through the 20-pin Host Port heade (HI08(r. This is J3 on the DSP56307/9EVM or JP11 for the DSP56362EVM.

For proper operation MODE/INT jumpers need to be removed. This will enable Host mode bootstrap mode. Mode jumpers are located on J1 header (DSP56307/9EVM) or on the JP8 header on the DSP56362EVM.

3

Printer Port I/F Pal Equations

```

MODULE HIP307
TITLE 'HIP307'
"Inputs
  [D0..D7]      pin 9..16;
  OPTIONS_SEL   pin 32;
  !HACK, !HREQ  pin 43,42;
  WRITE         pin 31;
  DATASTB     pin 26;
  ADDRSTB      pin 25;
"Outputs
"Test Point outputs
  TP1          pin 36 istype 'reg,buffer';
  TP2          pin 35 istype 'reg,buffer';
  !LED         pin 19 istype 'reg,invert';
  HA0,HA1,HA2  pin 4,3,2;
  !HEN, HRW    pin 1,44;
  DIR,ENA      pin 22,21;
  !HRESET      pin 41;
  !HDS         pin 33;
  !WAIT        pin 20;
  BHREQ        pin 24;
  BHACK        pin 23;
  SLCTOUT      pin 38;
"Outputs that don't leave this logic block (buried nodes)
  NIBBLESEL    node istype 'reg' ; "Set to enable lpt nibble mode
  OEENABLE     node istype 'reg' ; "OUTPUTS ARE DISABLED UNTIL SET
  AUTOINC      node istype 'reg' ; "Set to enable AutoInc mode
  LPTRST       node istype 'reg' ;
  [HIA2..HIA0] node istype 'reg' ; "Address bits to dsp host interface
  [CQ2..CQ0]   node istype 'reg' ; "Counter for autoinc address generator
"Set declarations
  LATCH_ADDR = !ADDRSTB #  OPTIONS_SEL; "Write addr on rising edge of ADDRSTB
  LATCH_OPT  = !ADDRSTB #  !OPTIONS_SEL; "Write cntl on rising edge of ADDRSTB
  LATCH_ADDR = !ADDRSTB;           "Write address or control bits on rising edge
  LATCH_OPT  = !OPTIONS_SEL;       "Select address or control bits
  COUNT      = [CQ2..CQ0];        "Up counter for autoincrement feature
  HADDR      = [HA2..HA0];        "Address bits to dsp host interface
  HIADDR     = [HIA2..HIA0];      "Address bits generated by writing to latch
  HENCLK     = HEN.PIN;           "Use .pin to maximize dsp address hold time
  NIBBLESEL  = OPTIONS_SEL;       "selects high/low nibble for nibble transfers
Equations
  DIR        = WRITE;             " Direction of the 245
  ENA        = !DATASTB & !ADDRSTB; "OEENABLE of the 245
  HEN        = DATASTB;          " access host port
  HDS        = DATASTB;          " access host port
  HRW        = !WRITE;            " Read or write host port
"All bits are inverted since they go through a 7414 on '307 board.

```

3

```

WAIT          = NIBBLEENA & !NIBBLESEL & !D3
              # NIBBLEENA & NIBBLESEL & !D7
              # !NIBBLEENA & (DATASTB # ADDRSTB);
BHREQ         = NIBBLEENA & !NIBBLESEL & !D2
              # NIBBLEENA & NIBBLESEL & !D6
              # !NIBBLEENA;          "11-24-99 version A of the EPP logic
BHACK         = NIBBLEENA & !NIBBLESEL & !D1
              # NIBBLEENA & NIBBLESEL & !D5;
SLCTOUT      = NIBBLEENA & !NIBBLESEL & !D0
              # NIBBLEENA & NIBBLESEL & !D4;
HRESET       = LPTRST;
"Define a latch that controls address and control bits.
LED          := D7;          "1=LED on
TP2          := D5;          "1=Test Point signal high
TP1          := D4;          "1=Test Point signal high
AUTOINC      := D3;          "0=direct address generation, 1=autoinc
NIBBLEENA    := D2;          "0=SPP or EPP modes, 1=Enable Nibble mode
LPTRST       := D1;          "reset to the DSP
OEENABLE     := D0;          "0=all pld signals are tri-stated
HIA2         := D2;
HIA1         := D1;
HIA0         := D0;
LED.C        = LATCH_OPT;
TP2.C        = LATCH_OPT;
TP1.C        = LATCH_OPT;
AUTOINC.C    = LATCH_OPT;
NIBBLEENA.C  = LATCH_OPT;
LPTRST.C     = LATCH_OPT;
OEENABLE.C   = LATCH_OPT;
[HIA2..HIA0].C = LATCH_ADDR;
"Signals to the dsp are tri-stated until a master OEENABLE bit is set
"by the lpt port. The LED & TP1 bits are never tri-stated.
[HA2..HA0].OE =OEENABLE;
HEN.OE       =OEENABLE;
HRW.OE       =OEENABLE;
HRESET.OE    =OEENABLE;

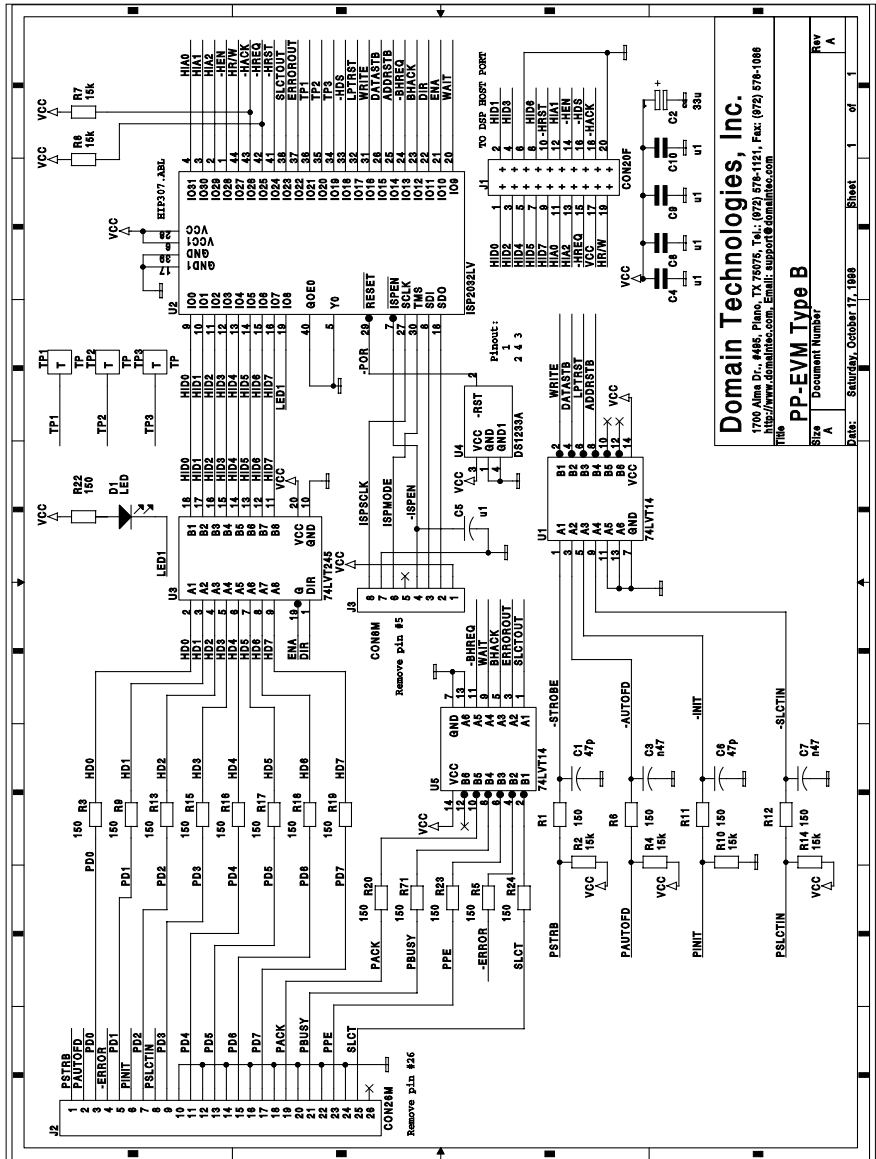
"Address bits to dsp are generated by taking the 3 address bits on the output
"of the latch, or by taking the output of the counter while in Autoinc mode.
"We are interested in generating autoinc sequence for access to registers
"2,5,6,7...2,5,6,7. The method to use autoinc is to set the AUTOINC bit
"and also write a 2 to the address bits on the latch at the same time.
" ie: write address 2 + AUTOINC to the latch then read dsp,read dsp,read dsp,
" read dsp. Same method will work for a write to the dsp.
"
"If the sequence needs to be aborted in the middle, just reset
"the AUTOINC bit, then start the sequence as above.
"
"Use latch outputs for address when not in autoinc mode, also use latch
"address for the 1st access when in autoinc mode since the counter needs
"at least 1 clock to get started.
"When in autoinc, 1st access must be from latch since counter has not been

```

```
"clocked yet.
HADDR = (AUTOINC==0) & HIADDR
# (AUTOINC==1) & (COUNT==0) & HIADDR
# (AUTOINC==1) & (COUNT!=0) & COUNT;
"Define a counter to implement an autoincrement address generator.
"Counter is clocked on each rising edge of the dsp's HEN signal.
"The counter is reset by !Autoinc, then the count is 5,6,7,2,5,6,7,2...

COUNT := (COUNT==0) & (COUNT+5) "After .ar is released start count at 5
# (COUNT==5) & (COUNT+1) "count to 6
# (COUNT==6) & (COUNT+1) "7
# (COUNT==7) & (COUNT-5) "2
# (COUNT==2) & (COUNT+3); "5
COUNT.AR = !AUTOINC; "counter stays reset until AUTOINC is enabled
COUNT.C = !HENCLK; "Clock on rising edge of dsp HEN signal
" Revision History
" 11/24/99 pek rev 1.2 1) Change ADDR and OPTIONS latch strobes signals
" 2) NIBBLESEL controlled by OPTIONS_SEL
" 5/09/98 SLH Rev 1.0001
" 1) Change WAIT to just be opposite polarity of ENA for EPP access.
" 2) Add output signal !HDS.
" 3) Delete interrupt logic, use registers to implement nibble mode.
" 4) Add autoinc mode
" 4/13/98 SLH Rev 1.0000
" Initial Development
END
```

Printer Port Interface Circuit Diagram



Printer Port Interface I/O port Equates

```

#define LATCH_OPTIONS      0x00 /* enable OPTIONS access */
#define LATCH_ADDR        0x04 /* enable ADDR access */
#define HIPPENA           0x01 /* PP enable */
#define LPTRESET          0x02 /* Target RESET */
#define NIBBLEENA         0x04 /* Enable nibble mode */
#define AUTOINC           0x08 /* Enable autoinc */
#define LED_ON            0x80 /* LED control */
#define DATA_PORT        ((unsigned short)(IOPort + 0))
#define STATUS_PORT       ((unsigned short)(IOPort + 1))
#define CTRL_PORT         ((unsigned short)(IOPort + 2))
#define EPP_ADDR          ((unsigned short)(IOPort + 3))
#define EPP_DATA          ((unsigned short)(IOPort + 4))
#define HOST_A0           0x01
#define HOST_A1           0x02
#define HOST_A2           0x04
#define HOST_ICR          (0)
#define HOST_CVR          (HOST_A0)
#define HOST_ISR          (HOST_A1)
#define HOST_IVR          (HOST_A0+HOST_A1)
#define HOST_HIGH         (HOST_A0+HOST_A2)
#define HOST_MID          (HOST_A1+HOST_A2)
#define HOST_LOW          (HOST_A0+HOST_A1+HOST_A2)
int resetEPP (void)
{
    outp (CTRL_PORT, LATCH_OPTIONS);           //enable options latch
    outp (EPP_ADDR, HIPPENA + LPTRESET);       //Ena host port and assert reset
    outp (EPP_ADDR, HIPPENA + LPTRESET + LED_ON); //Ena port and assert reset
    outp (CTRL_PORT, LATCH_ADDR);             //enable addr latch
    outp (EPP_ADDR, HOST_BOOT + HOST_IVR);     //Ena flash boot, and IVR
    _Delay (100L);
    outp (CTRL_PORT, LATCH_OPTIONS);           //enable options latch
    outp (EPP_ADDR, HIPPENA + LED_ON);         //deassert reset
    outp (CTRL_PORT, LATCH_ADDR);             //enable addr latch
    _Delay (100L);
    value = inp (EPP_DATA);                    //read IVR
    if (value != 0x0f)
        return FALSE;
    return TRUE;
}
BOOL waitStatus (int mask, int value)
{
    int status;
    unsigned long time;
    int i;
    status = inp (EPP_DATA);                    //read ISR
    if ((status & mask) == value)
        return TRUE;
}

```

3

```

for (i = 0; i < 10; i++)
{
    status = inp (EPP_DATA);                //read ISR
    if ((status & mask) == value)
        return TRUE;
}
time = Timer ();
while (Timer () - time < 400L)
{
    status = inp (EPP_DATA);                //read ISR
    if ((status & mask) == value)
        return TRUE;
    DosSleep (10);
}
status = inp (EPP_DATA);                    //read ISR
if ((status & mask) == value)
    return TRUE;
return FALSE;
}
BOOL writeEPP (long lValue)
{
    int value;
    outp (EPP_ADDR, HOST_ISR);              //set address to ISR
    value = inp (EPP_DATA);                 //read ISR
    if ((value & 6) != 6)                   //if HRX not empty
    {
        if (waitStatus (6, 6) == FALSE)
            return FALSE;
    }
    outp (EPP_ADDR, HOST_HIGH);
    outp (EPP_DATA, (int)(lValue >> 16));
    outp (EPP_ADDR, HOST_MID);
    outp (EPP_DATA, (int)(lValue >> 8));
    outp (EPP_ADDR, HOST_LOW);
    outp (EPP_DATA, (int)(lValue >> 0));
    return TRUE;
}
BOOL readEPP (long *plValue)
{
    int value;
    outp (EPP_ADDR, HOST_ISR);              //set address to ISR
    value = inp (EPP_DATA);                 //read ISR
    if ((value & 1) != 1)                   //If HTX empty
    {
        if (waitStatus (1, 1) == FALSE)
            return FALSE;
    }
    outp (EPP_ADDR, HOST_HIGH);
    *plValue = ((long)inp (EPP_DATA) << 16) & 0xff0000L;
    outp (EPP_ADDR, HOST_MID);
    *plValue |= ((long)inp (EPP_DATA) << 8) & 0xff00L;
    outp (EPP_ADDR, HOST_LOW);
    *plValue |= ((long)inp (EPP_DATA) << 0) & 0xffL;
    return TRUE;
}
int getEppIsr (void)
{

```

```

outp (CTRL_PORT, LATCH_OPTIONS); //enable options latch
outp (EPP_ADDR, HIPPENA+LED_ON); //disable autoincrement mode
outp (CTRL_PORT, LATCH_ADDR); //enable addr latch
outp (EPP_ADDR, HOST_NOINT + HOST_ISR);
outp (CTRL_PORT, LATCH_OPTIONS); //enable options latch
outp (EPP_ADDR, HIPPENA+AUTOINC+LED_ON); //Ena autoincrement mode
outp (CTRL_PORT, LATCH_ADDR); //enable addr latch
return inp (EPP_DATA);
}
}
BOOL waitStatusEpp (int mask, int value)
{
    int status;
    unsigned long time;
    int i;
    status = getEppIsr ();
    if ((status & mask) == value)
        return TRUE;
    for (i = 0; i < 10; i++)
    {
        status = getEppIsr ();
        if ((status & mask) == value)
            return TRUE;
    }
    time = Timer ();
    while (Timer () - time < 400L)
    {
        status = getEppIsr ();
        if ((status & mask) == value)
            return TRUE;
        DosSleep (10);
    }
    status = getEppIsr ();
    if ((status & mask) == value)
        return TRUE;
    return FALSE;
}
}
BOOL txBufferEpp (unsigned long *buf, int count)
{
    int lValue;
    outp (CTRL_PORT, LATCH_OPTIONS); //enable options latch
    outp (EPP_ADDR, HIPPENA+AUTOINC+LED_ON); //Ena autoincrement mode
    outp (CTRL_PORT, LATCH_ADDR); //enable addr latch
    outp (EPP_ADDR, HOST_ISR); //set address to ISR
    while (count --)
    {
        lValue = inp (ioAddr); //read ISR
        if ((lValue & 6) != 6)
        {
            if (waitStatusEpp (6, 6) == FALSE) //
                return FALSE;
        }
        lValue = (int)(*buf++);
        outp (EPP_DATA, lValue >> 16);
        outp (EPP_DATA, lValue >> 8);
        outp (EPP_DATA, lValue >> 0);
    }
}

```

```
    outp (CTRL_PORT, LATCH_OPTIONS);    //enable options latch
    outp (EPP_ADDR, HIPPENA+LED_ON);    //disable autoincrement mode
    outp (CTRL_PORT, LATCH_ADDR);      //enable addr latch
    return TRUE;
}
BOOL rxBufferEpp (unsigned long *buf, int count)
{
    int    lValue;
    outp (CTRL_PORT, LATCH_OPTIONS);    //enable options latch
    outp (EPP_ADDR, HIPPENA+AUTOINC+LED_ON); //Ena autoincrement mode
    outp (CTRL_PORT, LATCH_ADDR);      //enable addr latch
    outp (EPP_ADDR, HOST_NOINT + HOST_ISR); //set address to IVR
    while (count --)
    {
        {
            lValue = inp (ioAddr);        //read ISR
            if ((lValue & 1) != 1)
            {
                if (waitStatusEpp (1, 1) == FALSE)
                    return FALSE;
            }
            lValue = (inp (EPP_DATA) << 16) & 0xff0000L;
            lValue |= (inp (EPP_DATA) << 8) & 0x00ff00L;
            lValue |= (inp (EPP_DATA) << 0) & 0x0000ffL;
            *buf++ = (unsigned long)lValue;
        }
        outp (CTRL_PORT, LATCH_OPTIONS);    //enable options latch
        outp (EPP_ADDR, HIPPENA+LED_ON);    //disable autoincrement mode
        outp (CTRL_PORT, LATCH_ADDR);      //enable addr latch
        return TRUE;
    }
}
```

3