1394 Design Schematic (TSBKGPLYNX)

The enclosed information is a copy of the documentation supplied to Texas Instruments by TechnoBox as documentation for a test board designed and built by TechnoBox for Texas Instruments. Texas Instruments does not guarantee the accuracy of this information. There are no known errors in this document.

TEXAS INSTRUMENTS

1394 Solutions Leader

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.

Errata for GPLynx Peripheral EVM board - 09/12/96

On schematic page 7:
The connections to the power programming pins on the TSB21LV03 are all tied low. Per the 1394-1995 standard table for the Self-ID Packet this is incorrect. This node is self-powered and supplies power to the cable. The amount of power will depend on the power supply used with the external power connector shown on schematic page 2.

Note that it is not neccessary to use the external FIFO. If the user designs HW to interface directly to the ISO port on the TSB12LV31 the FIFO may be disabled and data written directly to/from the ISO port. The ISO port data and control signals are brought out to the header pins on JP8, the ISO Expansion Interface.

Notes on files:

The files included on this disk are compressed using the utility PKZIP and may be uncompressed using the PKUNZIP utility.

This design does not incorporate galvanic isolation.

The files that will be uncompressed have the following extensions:

.asm - TMS320C52 assembler source code
.bat - DOS batch files
*.c - TMS320C52 C source code
.cfg - PCAD configuration files
.cmd - Command files for the DOS based TMS320C52 linker, emulator, & ROM HEX code generator
.doc - Documentation files in either Microsoft Word format or text format.
.eps - Encapsulated Postscript format printer files
.HEX - ROM code
.HXM - Hex map created by ROM HEX code generator
.LIB - Library file
.LST - Listing file
.map - Link map
.obj - TMS320C52 intermediate object code
.out - TMS320C52 executable code
.ps - Postscript format printer files
.pcb - PCAD PWB database
.sch - ORCAD format schematic pages
.src - Source code for the programming inside the FPG

Acknowledgment:

This GPLynx Starter Kit was designed for Texas Instruments by Technobox, Inc.

GPLynx Starter Kit - Revision 1

Created by:

Joe Norris
Technobox Inc.
8000 Midlantic Drive
Suite 110
Mt. Laurel, NJ 08054

609-778-5512

TSBKGPLYNX EVM

- TMS320C52 to TSB12LV21 to TSB21LV03
- TMS320C52 to ISA Bus per PC/104
- RS-232 port allows communication with the TMS320C52 contolling processor
- Source code executes DMA operations, reads, and writes

IEEE 1394 Starter Kit

Introduction

The IEEE 1394 “starter kit” design provides a Texas Instruments TSB12LV31/TSB21LV03 IEEE 1394 interface, RS232 interface, and a PC/104 expansion interface. All these units are programmed by a TMS320C52 DSP with 64Kwords of 16-bit FLASH memory to hold DSP program space, and 64K words of SRAM hold DSP data space.

A MACH445 complex PLD provides the hardware control for the design. It implements a single channel DMA controller which transfers data between the external FIFO and the SRAM.

A 1-byte to 2-byte transceiver bridges the 16-bit FIFO with the 8-bit 12LV31 ISO port. Data Space accesses from the DSP are mapped into the SRAM, so the DSP has direct access to the SRAM for variable storage, as well as IEEE 1394 buffer pools.

The PC104 expansion interface is essentially a 16-bit “ISA” bus found in Personal Computers. The “PC104” term applies for the unique mechanical packaging of the ISA bus in a “stacked” fashion, thereby allowing off-the-shelf I/O functions, such as SCSI, Video, Data Acquisition and Prototype boards to be used with the DSP starter kit. The “PC104” specification is controlled by the PC104 Consortium, and over 100 companies are currently producing PC104 modules. More information may be found at http://www.pc104.com.

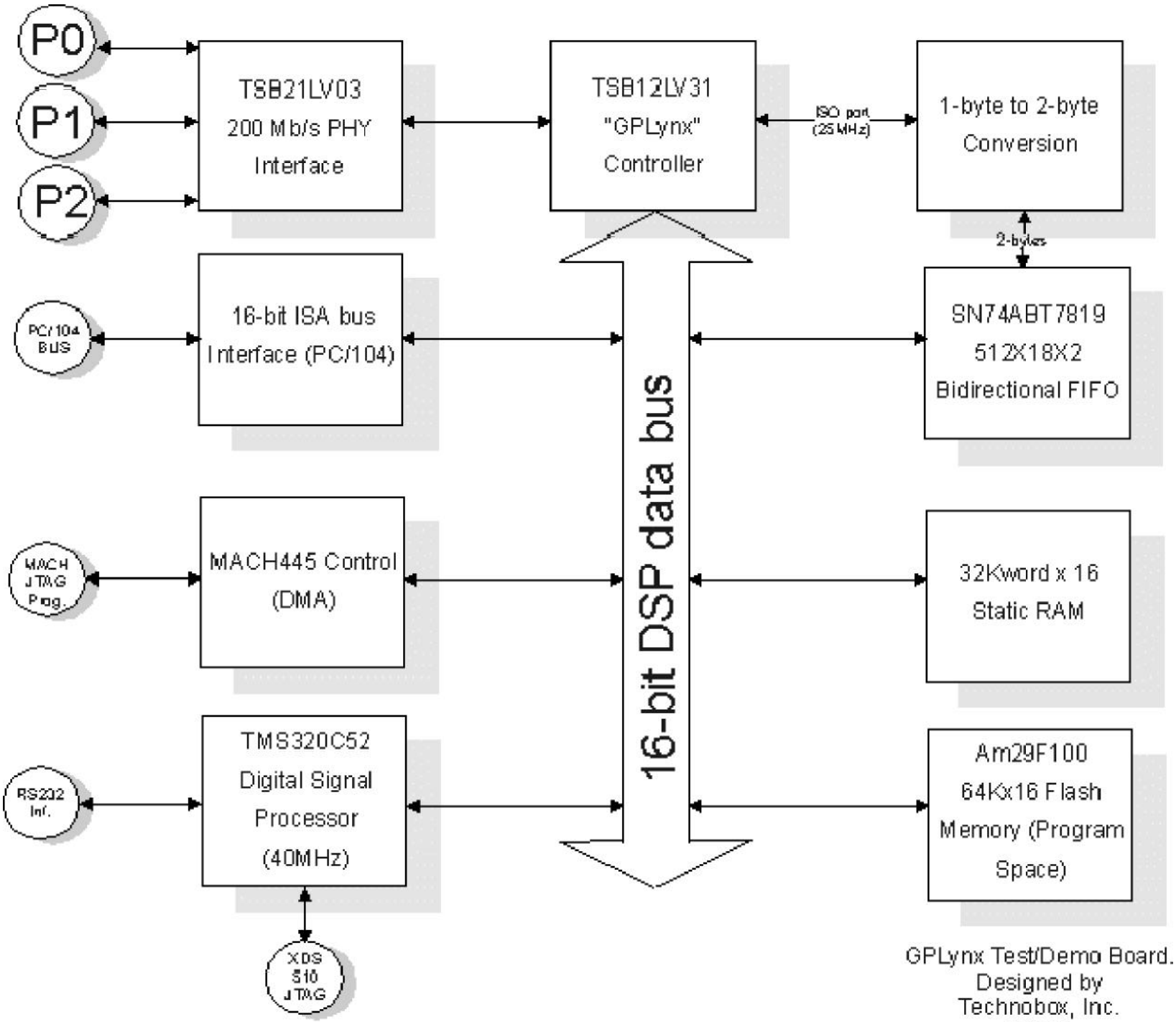
The Starter Kit design supports Bus Mastering of the PC104 bus; i.e., PC104 boards connected to the Starter Kit are slaves on the bus. The Starter Kit does not support slave access by other bus members on the PC104 bus (i.e., “Master*” signal is not supported). Note that the vast majority of PC104 boards do not support Master*.

The design supports DRQ/DACK interface on the ISA bus for handling DMA transfers from an I/O device to memory. The DRQ becomes an interrupt request to the DSP, and because of the fast interrupt context switching feature of the TMS320C52, adequate performance for many PC104 applications can be expected.

Since the performance of the ISA bus is slow compared to the DSP bus, the Starter Kit design features a programmable timer located in the MACH445 to control timing of the read/write strobes on the ISA bus. When this timer is set to a minimal value, the ISA bus is accessed at DSP bus speeds. This is useful for custom designs which are not bottlenecked by the ISA bus timing.

DRQ and IRQ lines are routed to their respective INT2, INT3, and INT4 inputs to the DSP via a MACH210 complex PLD. Thus, both ISA interrupts and DRQ requests are serviced in DSP firmware.

Power is provided by a either a Personal Computer 4-pin Hard Disk connector (+12, +5VDC), or by a standard PC104 power module. Power for the TSB21LV03 is from the Cable power regulated down by a linear regulator to 3.3 VDC. The board also sources Cable Power via a schottky diode as found in most IEEE 1394 implementations.



GPLynx Test/Demo Board.
Designed by
Technobox, Inc.
6/6/96

Programmer's guide to the IEEE 1394 GPLynx Starter Kit

Introduction

The IEEE 1394 "GPLynx" Starter Kit provides a variety of interfaces all controlled by a TMS320C52 Digital Signal Processor (DSP). The design provides a three-port IEEE 1394 200 Mb/s interfaced based on Texas Instruments 21LV03/12LV31 chipset and an expansion interface to a 16-bit industry standard ISA bus in a PC/104 form factor.

32Kwords of FLASH memory based program space visible to the TMS320C52, as well as 28K words of Static Random Access Memory (SRAM). There is also a 4K region of the TMS320C52 data space which is used to access on board registers and off board components on the PC/104 bus. The TMS320C52 I/O space is not used in this design.

This section details registers accessible from the TMS320C52. In the case of more complex functions, such as the 21LV03/12LC31 IEEE 1394 chipset, the user is referred to the relevant documentation provided by the chipset manufacture.

Although the registers presented herein are directly accessible through TMS320C52 assembly-level language, it is highly recommended that the C-language equivalents be used to implement programs for the Starter Kit. The optimizing C compilers, available from Texas Instruments, do a fine job in creating efficient machine code which can only slightly be improved upon by hand assembly coding.

As another foundation for programming the Starter Kit, it is recommended that the user examine the RS232 Monitor Program source code which is provided with the Starter Kit software package. In many cases the code used by the monitor can be duplicated or directly used by the Starter Kit programmer to accomplish the requisite tasks. For example, there are routines which perform DMA transfers between the ISO port FIFO and SRAM which would otherwise be fairly difficult to implement without a thorough understanding of the Starter Kit hardware.

Overall Memory Map

The TMS320C52 defines three disjoint memory spaces, each 64 Kwords in size. Note that the TMS320C52 is a 16-bit machine, and a 'word' in this context is 16-bits.

First, the 'program space' is the area in which TMS320C52 machine instructions are located. Programs execute out of the program space. The TMS320C52 also has some special instructions to access data within program space, but this is normally transparent to the programmer at the C-language level. In the Starter Kit design, the program space is implemented with a 64Kword x 16-bit FLASH memory; However, the hardware only allows 32K words of the FLASH memory available for use.

The code initially executed from the FLASH memory is the RS232-based Monitor Program, which provides a simple menu-driven human interface to interrogate, access, and initialize Starter Kit hardware components. The monitor provides a mechanism to upload new monitor code and re-burn the FLASH memory in-circuit accordingly.

As part of the TMS320C52 architecture, some of the program space is located in SRAM within the TMS320C52 DSP chip. This region is from 0xFE00 to 0xFFFF in program space. When the DSP accesses program instructions in this address range, the instructions are actually coming from within the chip as opposed to the external FLASH memory. Execution is significantly faster within this on-chip space; a full 40 MIPs can be achieved in the Starter Kit case. Execution from the FLASH is done with 2 DSP wait states, since the FLASH used in the design is 90-ns access. Program execution out of external FLASH effectively cuts the performance to 13 MIPS, which is still respectable for many applications.

Note that the On-Chip program space is automatically initialized with selected high-speed programs used by the RS232 Monitor Program. This code is found in the 'RAMCODE.ASM' program, and is mapped using the 'ramcode' segment in the TMS320C5X linker.

A second distinctly addressable region is 'data space.' Here, the DSP instruction set provides a rich assortment of address modes, both direct and indirect with autoincrement options. In the Starter Kit design, a 32-Kword SRAM is visible from the DSP data space. The Starter Kit hardware registers are also visible from Data Space, including access to the PC/104 expansion bus.

In the case of the GPLynx Starter Kit design, Program Space is aliased with the data space. Thus, the SRAM, which is nominally used for holding program variables and data, can also be used to hold user programs. This permits development of user code without the need to re-burn the FLASH memory.

A third addressable region is 'I/O space.' The DSP provides a limited number of address modes and instructions to access I/O space (specifically 'IN' and 'OUT'). The Starter Kit does not use I/O space. Note that the C-Language does not provided constructs to easily access I/O space, but it's very easy to access Data Space, which is why the Starter Kit implements the hardware registers in DSP Data Space.

There is yet another DSP addressable space, called 'global memory,' which is normally used in multi-processing shared memory architectures. The Starter Kit does not use global memory.

To summarize, the 'overall' memory map for the IEEE 1394 Starter Kit is as follows:

DSP space	Start	End	Description
Program	0x0000	0x7FFF	Holds FLASH code (Monitor)
Program/Data	0x8000	0xFDFF	Program space aliased w/SRAM
Program	0xFE00	0xFFFF	On chip Program Space
Data	0x0000	0x02FF	DSP internal registers
Data	0x0300	0x04FF	On-chip general purpose SRAM
Data	0x0500	0xEFFF	Starter Kit SRAM
Data	0xF000	0xFFFF	Starter Kit hardware registers
I/O	0x0000	0xFFFF	Not used in Starter Kit design

Most of the spaces as described above should be familiar to a programmer who wishes to use the Starter Kit. However, the Data Space registers in the range 0xF000 to 0xFFFF are external hardware registers peculiar to the Starter Kit design, and these are described later in this programming guide.

Starter Kit hardware Register Summary

The specific addresses for the Starter Kit registers visible within the TMS320C52 data space are as follows:

Mnemonic	Data Space Addr	Description
la_mach_rev	0xF000h	Read returns MACH445 Revision code
la_isotst	0xF100h	Read/Write ISO port test register
la_12lv31	0xF200h	Read/Write 12LV31 Microprocessor port
la_pc104adr	0xF300h	Write PC104 address register
la_trig	0xF400h	Logic Analyzer Trigger
la_ctrlstat	0xF500h	Write control register/Read Status reg.
la_rdytime	0xF600h	PC104 sample ready delay
la_xfercntr	0xF700h	Read/write transfer counter
la_dmago	0xF800h	Write here kicks off DMA transfer
la_addrhi	0xF900h	Load address counter hi
la_addrlo	0xFA00h	Load address counter low
la_pc104dat	0xFB00h	Read/Write PC104 space
la_dmactrl	0xFC00h	DMA control register (in MACH445)
la_wrprio	0xFD00h	Write PC/104 IRQ/DRQ select mux
la_ldpg	0xFE00h	Load PC/104 Page Register
la_lddack	0xFF00h	Load PC/104 Address Register

In this table the 'Mnemonic' is the same character string used in the MACH445 design, which contains the address recognition for the DSP access to hardware registers, and also used in the C-language code for the RS232 Monitor Program which is burned into the Starter Kit FLASH memory.

Accessing Starter Kit Registers from C-Language

Since most programmers will be using C-language to manipulate the Starter Kit hardware registers, this section will discuss what is involved in using C-language in this regard.

First, the register addresses need to be defined using the C-language #DEFINE statement. #DEFINE's for the Starter Kit hardware are found in the RS232 Monitor Program code (GPLNX.C), and is represented below:

```
#define   la_mach_rev     (*(volatile int *)0xF000)     /* Mach 445 revision code                    */
#define   la_isotst       (*(volatile int *)0xF100)     /* Read/write ISO port test register         */
#define   la_12lv31       (*(volatile int *)0xF200)     /* Access 12LV31 GPLynx chip              */
#define   la_pc104adr     (*(volatile int *)0xF300)     /* PC/104 address register                   */
#define   la_trig         (*(volatile int *)0xF400)     /* Logic Analyzer Trigger                    */
#define   la_ctrlstat     (*(volatile int *)0xF500)     /* Write control register/Read Status reg    */
#define   la_rdytime      (*(volatile int *)0xF600)     /* PC/104 sample ready delay                 */
#define   la_xfercntr     (*(volatile int *)0xF700)     /* Read/write transfer counter               */
#define   la_dmago        (*(volatile int *)0xF800)     /* Write here kicks off DMA transfer         */
#define   la_addrhi       (*(volatile int *)0xF900)     /* Load address counter hi                   */
#define   la_addrlo       (*(volatile int *)0xFA00)     /* Load address counter low                  */
#define   la_pc104dat     (*(volatile int *)0xFB00)     /* PC/104 data bus read/write                */
#define   la_dmactrl      (*(volatile int *)0xFC00)     /* DMA control register                      */
#define   la_wriprio2     (*(volatile int *)0xFD00)     /* Write MACH210 DRQ/IRQ mux control INT2    */
#define   la_wriprio3     (*(volatile int *)0xFD01)     /* Write MACH210 DRQ/IRQ mux control INT3    */
#define   la_wriprio4     (*(volatile int *)0xFD02)     /* Write MACH210 DRQ/IRQ mux control INT4    */
#define   la_ldpg         (*(volatile int *)0xFE00)     /* Load PC/104 page register                 */
#define   la_lddack       (*(volatile int *)0xFF00)     /* Load PC/104 DACK register                 */
```

Note that each address is first cast as a 'pointer' type (using "(volatile int *)") and then the C-language 'indirect' operator ("*") is used to access the register. Thus, a programmer can access the 'contents' of the hardware register simply by using the la_xxxxx reference.

For example, to write a constant to the PC/104 bus, assuming the bus control registers have been set up by some prior code, one can write:

```
la_pc104dat = 0x1234;            /* Write PC104 bus with constant 0x1234 */
```

Similarly to read the PC/104 bus, one writes:

```
k = la_pc104dat;                 /* Read PC/104 bus into variable "k" */
```

Note that the use of the 'volatile' modifier in the pointer cast is required so the C compiler does not optimize out the reference. If this were not used, then the global optimizer (may) observe the reference was not changed in prior code and consequently remove the code from the program.

How to access the IEEE 1394 interface (Microprocessor port)

The 12LV31 link level controller provides a 16-bit wide interface intended for control from a microprocessor. In this case, the microprocessor is a TMS320C52 Digital Signal Processor. This microprocessor port is mapped into the data space of the DSP, and can be accessed by reading/writing address la_12LV31.

Note that the 12LV31 permits configuration of the microprocessor port to be either 8- or 16-bits wide. For nominal application level operation, the hardware should be configured for 16-bit mode, and this is the default as set up by the Monitor program upon reset.

The 12LV31 registers are all 32-bits wide, and the 12LV31 hardware requires that the registers be accessed with multiple 16-bit (or 8-bit) bus cycles in a specific order; Otherwise, the registers will not be accessed properly.

The key "Driver" Routines which access the 12LV31 registers are shown below. One routine is for reading the 12LV31, and the other is for writing. Notice that the register contents are transfered between SRAM and the addressed 12LV31 register. These routines allow multiple 32-bit transfers to be performed with a contiguous set of registers in the 12LV31.

```
/* Read data from GPLynx port to SRAM */

void gplynxread(int srcadr,
            int destadr,
            int cnt)    /* cnt is in quadlets */

{
  int i,j;

  if ((ctrlreg_image & ctrl_mcmd0) == 0)
  {                                          /* 16-bit mode */
    for (i = 0; i < cnt; i++)
    {
      *(unsigned int *)(destadr + (2*i) + 1) = *(unsigned int *)(la_12lv31_addr + srcadr + (4*i));
      *(unsigned int *)(destadr + (2*i))     = *(unsigned int *)(la_12lv31_addr + srcadr + (4*i) + 2);
    };
  }
  else                                       /* 8-bit mode */
  {
    for (i = 0; i < cnt; i++)
    {
      *(unsigned int *)(destadr + (2*i) + 1)  = (*(unsigned int *)(la_12lv31_addr + srcadr + (4*i))) << 8;
      *(unsigned int *)(destadr + (2*i) + 1) |= (*(unsigned int *)(la_12lv31_addr + srcadr + (4*i) + 1)) & 0x00ff;
      *(unsigned int *)(destadr + (2*i))      = (*(unsigned int *)(la_12lv31_addr + srcadr + (4*i) + 2)) << 8;
      *(unsigned int *)(destadr + (2*i))     |= (*(unsigned int *)(la_12lv31_addr + srcadr + (4*i) + 3)) & 0x00ff;
    };
  };
}

/* Write gplynx lynx chip from SRAM */

void gplynxwrite(int srcadr,
             int destadr,
             int cnt)    /* cnt is in quadlets */

{
  int i,j;

  if ((ctrlreg_image & ctrl_mcmd0) == 0)
  {                                  /* 16-bit mode */
    for (i = 0; i < cnt; i++)
    {
      *(unsigned int *)(la_12lv31_addr + destadr + (4*i))     = *(unsigned int *)(srcadr + (2*i) + 1);
      *(unsigned int *)(la_12lv31_addr + destadr + (4*i) + 2) = *(unsigned int *)(srcadr + (2*i));
    };
  }
  else                               /* 8-bit mode  */
  {
    for (i = 0; i < cnt; i++)
    {
      *(unsigned int *)(la_12lv31_addr + destadr + (4*i))       = (*(unsigned int *)(srcadr + (2*i) + 1)) >> 8;
      *(unsigned int *)(la_12lv31_addr + destadr + (4*i) + 1)   = *(unsigned int *)(srcadr + (2*i) + 1);
      *(unsigned int *)(la_12lv31_addr + destadr + (4*i) + 2)   = (*(unsigned int *)(srcadr + (2*i))) >> 8;
      *(unsigned int *)(la_12lv31_addr + destadr + (4*i) + 3)   = *(unsigned int *)(srcadr + (2*i));
    };
  };
}
```

The above routine is called by higher-level routines to read/write any 32-bit 12LV31 register. These routines are as follows:

```
/* Return specified 32-bit register from 12lv31 chip */

unsigned long getlink(int adr)
```

```
{
  gplynxread(adr,dmabuff_addr,1);              /* Perform DMA transfer */
  return(*(unsigned long *)dmabuff_addr);
}

/* Write 32-bit quantity to specified 12lv31 chip register */

void putlink(int adr, unsigned long dat)
{
  *(unsigned long *)dmabuff_addr = dat;
  gplynxwrite(dmabuff_addr,adr,1);
}
```

How to access the IEEE 1394 interface (ISO port)

The 12LV31 link controller provides a separate byte-wide port for performing ISO packet transfers. Both ISO reads and ISO writes can be accomplished via this port, provided both the 12LV31 internal registers are configured correctly, and provided the Starter Kit Hardware is configure properly for the ISO write or read operation to be performed.

For efficient transfers, the Starter Kit hardware provides a 16-bit wide bi-directional FIFO (512 Words deep) between the DSP data bus and the ISO port. Hardware converts the 16-bit wide bus into the 8-bit wide bus compliant with the 12LV31 ISO port. DMA transfers are used to move data to/from the SRAM and the FIFO. The user need only condition the hardware to perform the ISO transfer, then use DMA transfers to send and receive ISO packet data.

To set up the ISO port hardware to read or write data, the following routines are used:

```
/* Set ISO read mode */

void setisoreadmode()
{
  la_dmactrl &= ~dmactrl_isodir;       /* Init DMA direction    */
  ctrlreg_image &= ~ctrl_isodir;
  la_ctrlstat = ctrlreg_image;         /* ctrl_isodir is for ISO test mode */
}

/* Set ISO write mode */

void setisowritemode()
{
  la_dmactrl |= dmactrl_isodir;        /* Init DMA direction    */
  ctrlreg_image |= ctrl_isodir;
  la_ctrlstat = ctrlreg_image;         /* ctrl_isodir is for ISO test mode */
}
```

Note that you MUST also configure the 12LV31 registers to indicate the ISO port direction. Specific code fragments for doing this follows:
(For the READ case):

```
  link_isomode_union.link_isomode.isodmaen = 1;
  link_isomode_union.link_isomode.isomode = 1;                          /* Set ISO mode back to GRF receive */
  putlink(link_isomode_adr,link_isomode_union.link_isomode_reg);        /* Set ISO DMA en */
  setisoreadmode();                                                     /* Default back to ISO read mode */
```

(For the WRITE case):

```
  link_isomode_union.link_isomode.isodmaen = 1;
  link_isomode_union.link_isomode.isomode = 0;                          /* Set ISO mode to xmit via ISO port */
  putlink(link_isomode_adr,link_isomode_union.link_isomode_reg);        /* Set ISO DMA en */
  isofiforst();     /* Reset ISO FIFO  prior to doing transfers */
  setisowritemode();
```

Once the 12LV31 and Starter Kit hardware is set up properly, the user effects DMA transfers between the ISO FIFO and SRAM using the following routines. Note that these routines will wait until the DMA transfer has completed, before returning to the caller. If there is a hardware error, or if the DMA count does not match the data which is available in the FIFO, then the routine will hang waiting forever.

```
/* Set up DMA controller for read transfer and start DMA transfer */
/*   Transfers always with ISO port FIFO                          */

void dmaread(int destadr,
             int cnt)

{
  la_dmactrl &= ~dmactrl_isodir;       /* Init DMA direction    */
  ctrlreg_image &= ~ctrl_isodir;
  la_ctrlstat = ctrlreg_image;         /* ctrl_isodir is for ISO test mode */
  la_xfercntr = cnt;                   /* Load transfer counter */
  la_addrhi = destadr >> 8;
  la_addrlo = destadr;                 /* Load addr cntr hi/lo  */
  la_dmago = 0;                        /* Start DMA transfer    */
  while ((la_dmactrl & dmastat_dmabusy) != 0) {};   /* Wait for xfer complete */
}
```

```
/* Set up DMA controller for write transfer and start DMA transfer */
/*   Transfers always with ISO port FIFO                           */

void dmawrite(int srcadr,
              int cnt)

{
  la_dmactrl |= dmactrl_isodir;        /* Init DMA direction    */
  ctrlreg_image |= ctrl_isodir;
  la_ctrlstat = ctrlreg_image;         /* ctrl_isodir is for ISO test mode */
  la_xfercntr = cnt;                   /* Load transfer counter */
  la_addrhi = srcadr >> 8;
  la_addrlo = srcadr;                  /* Load addr cntr hi/lo  */
  la_dmago = 0;                        /* Start DMA transfer    */
  while ((la_dmactrl & dmastat_dmabusy) != 0) {};   /* Wait for xfer complete */
}
```

A complete example of how to operate the ISO port interface using the above routines is found below. This program prompts the user for ISO block parameters, then performs an ISO packet transmit using the DMA transfer hardware.

```
/* Transmit ISO packet with N quadlets */

void xmitiso()
{
  int i,j,k;
  unsigned long x,y;
  int sp;
  int sy;
  int pkts;
  unsigned int quads;
  unsigned int adr;

  write("Enter destination ISO channel number: ");
  link_isoheader_union.link_isoheader_reg = 0;
  link_isoheader_union.link_isoheader.channelnumber = getbin();
  writeln("");
  write("Enter transmit speed (0 = 100 Mb/sec, 1 = 200 Mb/sec): ");
  sp = getbin();
  writeln("");
  link_isoheader_union.link_isoheader.speed = sp;
  write("Enter packet data length in quadlets: ");
  link_isoheader_union.link_isoheader.packetdatalength = ((quads = getbin()) << 2);
  writeln("");
  link_isocontrol_union.link_isocontrol.quadsperpacket = quads;
  write("Enter number of packets for this data block: ");
  pkts = getbin();
  writeln("");
  x = (unsigned long)quads * (unsigned long)pkts;
  link_isocontrol_union.link_isocontrol.quadsperblock_low = (unsigned int)x;
  link_isocontrol_union.link_isocontrol.quadsperblock_hi = (unsigned int)(x >> 6);
  write("Enter SY field value for first ISO packet trasmitted: ");
  sy = getbin();
  writeln("");
  link_isoheader_union.link_isoheader.syncbiten = sy;
  write("Enter DSP data space address sourcing ISO transmit data: ");
  adr = getbin();
  writeln("");

  putlink(link_isocontrol_adr,link_isocontrol_union.link_isocontrol_reg);
  putlink(link_isoheader_adr,link_isoheader_union.link_isoheader_reg);

  link_isomode_union.link_isomode.isodmaen = 1;
  link_isomode_union.link_isomode.isomode = 0;                          /* Set ISO mode to xmit via ISO port */
  putlink(link_isomode_adr,link_isomode_union.link_isomode_reg);        /* Set ISO DMA en */


  isofiforst();     /* Reset ISO FIFO  prior to doing transfers */
  setisowritemode();


    for (j = 0; j < pkts; j++)
    {
      dmawrite(adr,quads);
      adr += (quads * 2);
    };


  suspend(1000);             /* Suspend 100 ms to give time for ISO packets to complete */

  link_isomode_union.link_isomode.isodmaen = 1;
  link_isomode_union.link_isomode.isomode = 1;                          /* Set ISO mode back to GRF receive */
  putlink(link_isomode_adr,link_isomode_union.link_isomode_reg);        /* Set ISO DMA en */
  setisoreadmode();                                                     /* Default back to ISO read mode */

  writeln("ISO packets transmitted");

}
```

How to access the PC/104 interface

The PC/104 interface is accessed as follows:

1. Set up the PC/104 la_wripriox (where x=2,3,4) to multiplex the DRQ/IRQ lines from the PC104 bus to the desired INTx (x=2,3,4) interrupt lines going to the TMS320C52. You only need to do this if you are using interrupt service routines to handle PC/104 DMA and IRQ requests. Note that an RS232 Monitor Program function is available to more conveniently do this operation.
2. Set up the PC/104 RD/WR strobe timing by writing appropriate values to la_rdytime. An RS232 Monitor Program function is available for this operation as well.
3. Set up the PC/104 control register to indicate IO, MEMORY, or SYSTEM MEMORY access by writing to the la_dmactrl register. Note that PC/104 has a separate set of RD/WR strobes for each of these address spaces, and the user needs to tell the Starter Kit hardware which RD/WR strobe set to use in subsequent PC/104 bus cycles.
4. Initialize the AEN, REFRESH, TC and SBHE control signals in the la_ctrlstat register appropriately for the PC/104 cycle you wish to do.
5. Identify the specific PC/104 address you want to access by writing to la_pc104adr and la_ldpg registers. This creates a 24-bit address emitted onto the PC104 bus.
6. Access the PC/104 data by reading/writing the la_pc104dat register.

Many of the above steps can be eliminated if the user finds the defaults set up by the RS232 Monitor Program is adequate for the application. The Monitor Program defaults the PC/104 bus to IO access (i.e., use IOR and IOW strobes on the PC/104 bus), 330/270 nanosecond read/write strobe width, and AEN, REFRESH, TC and SBHE all low. This is fine for 8-bit accesses to I/O ports on the PC/104 bus, such as to RS232 interface UARTS.

As an example of how to access the PC/104 bus, consider the following program fragment which echoes RS232 characters on a COM1 port attached to the PC/104 bus:

```
/* Set PC/104 modes for System memory, memory, and I/O accesses */

void setpc104mode(int i)
{
  la_dmactrl = ((la_dmactrl & pc104mode_mask_n) | (i << 1));
}

/* Set up PC/104 address bus */

void pc104addr(unsigned long addr)
{
  la_pc104adr = addr;                          /* Main address bus                    */
  la_ldpg = (addr >> 16);                      /* Load PC/104 addr[23..16]   */
}

/* Echo COM1 characters of PCM-3410 PC/104 board */

void echocom1()
{
  unsigned long i;
  unsigned int k,j;

  setpc104mode(pc104mode_io);                  /* Access PC/104 bus I/O space */
  pc104addr(0x3fb);                            /* Address Line Control Reg   */
  la_pc104dat = 0x83;                          /* Access DLL and DLH         */
  pc104addr(0x3f8);
  la_pc104dat = 6;                             /* Write DLL for 19.2 KB               */
  pc104addr(0x3f9);
  la_pc104dat = 0;                             /* Write DLH                           */
  pc104addr(0x3fb);
  la_pc104dat = 0x03;
  writeln("Now echoing characters on PCM-3410 COM1 port at 19.2Kbaud");
  writeln("  Hit reset button to terminate");
  for(;;)
  {
    pc104addr(0x3fd);                                  /* Address ready flags                */
    while ((la_pc104dat & 0x01) == 0) {};              /* Wait for received char             */
    pc104addr(0x3f8);
    la_pc104dat = la_pc104dat;                         /* Echo back character                */
  };
}
```

12LV31/21LV03 IEEE 1394 Starter Kit register summary

Register Address	Register Mnemonic	Short Description
0xF000	la_mach_rev	Returns MACH 445 revision code
0xF100	la_isotst	ISO port test register
0xF200	la_12lv31	Access 12LV31 Microprocessor Port
0xF300	la_pc104adr	PC/104 address register LSBs
0xF400	la_trig	Logic Analyzer Trigger
0xF500(rd)	la_ctrlstat	Hardware Control Register
0xF500(wr)	la_ctrlstat	Hardware Status Register
0xF600	la_rdytime	PC/104 Sample Ready Delay
0xF700	la_xfercntr	DMA Transfer Counter
0xF800	la_dmago	Kick off DMA transfers
0xF900	la_addrhi	Load DMA address counter high byte
0xFA00	la_addrlo	Load DMA address counter low byte
0xFB00	la_pc104dat	Access PC/104 space
0xFC00	la_dmactrl	DMA control/status register + misc
0xFD00	la_wrprio2	Write PC/104 IRQ/DRQ select multiplexer for INT2 DSP input
0xFD01	la_wrprio3	Write PC/104 IRQ/DRQ select multiplexer for INT3 DSP input
0xFD02	la_wrprio4	Write PC/104 IRQ/DRQ select multiplexer for INT4 DSP input
0xFE00	la_ldpg	Load PC/104 address register bits [23..16]
0xFF00	la_lddack	Load PC/104 DACK register

12LV31/21LV03 IEEE 1394 Starter Kit Register detail

Register Address: 0xF000
Register Mnemonic: la_mach_rev
Short Description: Returns MACH 445 revision code

Register Description: Returns an 8-bit value as read from the MACH 445 PLD on the board, indicating the revision level of the MACH 445 code. Used by the Monitor program to determine if the MACH 445 revision is appropriate for the code revision.

Bit Number	Mnemonic	Read/Write	Description
0	D0	Rd	Least Significant Bit
1	D1	Rd	
2	D2	Rd	
3	D3	Rd	
4	D4	Rd	
5	D5	Rd	
6	D6	Rd	
7	D7	Rd	Most Significant Bit
8			Not Used
9			Not Used
10			Not Used
11			Not Used
12			Not Used
13			Not Used
14			Not Used
15			Not Used

Register Address: 0xF100
Register Mnemonic: la_isotst
Short Description: ISO port test register

Register Description: 8-bit Hardware register connected to 12LV31 ISO port. Used exclusively for diagnostic testing of the ISO port FIFO as provided by the Monitor Program.

Bit Number	Mnemonic	Read/Write	Description
0	D0	Rd/Wr	Least Significant Bit
1	D1	Rd/Wr	
2	D2	Rd/Wr	
3	D3	Rd/Wr	
4	D4	Rd/Wr	
5	D5	Rd/Wr	
6	D6	Rd/Wr	
7	D7	Rd/Wr	Most Significant Bit
8			Not Used
9			Not Used
10			Not Used
11			Not Used
12			Not Used
13			Not Used
14			Not Used
15			Not Used

12LV31/21LV03 IEEE 1394 Starter Kit Register detail

Register Address: 0xF200
Register Mnemonic: la_12lv31
Short Description: Access 12LV31 Microprocessor Port

Register Description: Access to 12LV31 Microprocessor port. Either 8-bit or 16-bit wide depending on MCMODE[1..0] bit settings. Normally set up for 16-bit accesses.

Bit Number	Mnemonic	Read/Write	Description
0	D0	Rd/Wr	Least Significant Bit
1	D1	Rd/Wr	
2	D2	Rd/Wr	
3	D3	Rd/Wr	
4	D4	Rd/Wr	
5	D5	Rd/Wr	
6	D6	Rd/Wr	
7	D7	Rd/Wr	
8	D8	Rd/Wr	
9	D9	Rd/Wr	
10	D10	Rd/Wr	
11	D11	Rd/Wr	
12	D12	Rd/Wr	
13	D13	Rd/Wr	
14	D14	Rd/Wr	
15	D15	Rd/Wr	Most Significant Bit

Register Address: 0xF300
Register Mnemonic: la_pc104adr
Short Description: PC/104 address register LSBs

Register Description: Register holding the least significant 16 bits of the 24-bit ISA address as emitted on the PC/104 bus. Note that this is a write-only register.

Bit Number	Mnemonic	Read/Write	Description
0	PA0	Wr	PC/104 Address bit [0]
1	PA1	Wr	PC/104 Address bit [1]
2	PA2	Wr	PC/104 Address bit [2]
3	PA3	Wr	PC/104 Address bit [3]
4	PA4	Wr	PC/104 Address bit [4]
5	PA5	Wr	PC/104 Address bit [5]
6	PA6	Wr	PC/104 Address bit [6]
7	PA7	Wr	PC/104 Address bit [7]
8	PA8	Wr	PC/104 Address bit [8]
9	PA9	Wr	PC/104 Address bit [9]
10	PA10	Wr	PC/104 Address bit [10]
11	PA11	Wr	PC/104 Address bit [11]
12	PA12	Wr	PC/104 Address bit [12]
13	PA13	Wr	PC/104 Address bit [13]
14	PA14	Wr	PC/104 Address bit [14]
15	PA15	Wr	PC/104 Address bit [15]

12LV31/21LV03 IEEE 1394 Starter Kit Register detail

Register Address: 0xF400
Register Mnemonic: la_trig
Short Description: Logic Analyzer Trigger

Register Description: A write to this address causes the "Trigger" signal on the logic analyzer headers to be asserted. The purpose of this is to provide a means for DSP firmware to trigger a logic analyzer synchronously with execution of the program.

Bit Number	Mnemonic	Read/Write	Description
0		Wr	Not Used
1		Wr	Not Used
2		Wr	Not Used
3		Wr	Not Used
4		Wr	Not Used
5		Wr	Not Used
6		Wr	Not Used
7		Wr	Not Used
8		Wr	Not Used
9		Wr	Not Used
10		Wr	Not Used
11		Wr	Not Used
12		Wr	Not Used
13		Wr	Not Used
14		Wr	Not Used
15		Wr	Not Used

Register Address: 0xF500(rd)
Register Mnemonic: la_ctrlstat
Short Description: Hardware Control Register

Register Description: A 16-bit status register is maintained in the Starter Kit hardware. This read-only register provides current hardware for a variety of signals as defined below.

Bit Number	Mnemonic	Read/Write	Description
0	stat_cystart	Rd	Cycle Start from 12LV31 (Pin #94)
1	stat_cydone	Rd	Cycle Done from 12LV31 (Pin #95)
2	stat_stat0	Rd	Status Output 0 from 12LV31 (Pin #92)
3	stat_stat1	Rd	Status Output 1 from 12LV31 (Pin #93)
4	stat_cyclout	Rd	Cycle Timer output from 12LV31 (Pin #24)
5	stat_mem16	Rd	"MEM16" input from PC/104 interface
6	stat_io16	Rd	"IO16" input from PC/104 interface
7	stat_cts	Rd	Clear to Send input from RS232 interface
8	stat_pwron	Rd	"PWRON" from 12LV31 (Pin #97)
9	stat_mclk	Rd	"MCLK" clock output from 12LV31 (Pin #55)
10	stat_marxd	Rd	"MARXD" status from 12LV31 (Pin #89)
11	stat_mirxd	Rd	"MIRXD" status from 12LV31 (Pin #90)
12	stat_isoerror	Rd	"ISOERROR" status output from 12LV31 (Pin #22)
13	stat_pktflag	Rd	"PKTFLAG" output from 12LV31 (Pin #20)
14	stat_orab	Rd	When set, indicates data avail at 7819 ISO FIFO
15	stat_dmadne	Rd	"IDMADONE" from 12LV31 (Pin #19)

12LV31/21LV03 IEEE 1394 Starter Kit Register detail

Register Address: 0xF500(wr)
Register Mnemonic: la_ctrlstat
Short Description: Hardware Status Register

Register Description: A 16-bit register in the Starter Kit hardware is used for control. Since it is a write-only register, an image of the contents of the register is maintained in the variable "ctrl_image" by the Monitor Firmware. This register is cleared by hardware reset.

Bit Number	Mnemonic	Read/Write	Description
0	ctrl_led0	Wr	When 0, turns "D2" LED on
1	ctrl_led1	Wr	When 0, turns "D3" LED on
2	ctrl_led2	Wr	When 0, turns "D4" LED on
3	ctrl_rts	Wr	When 1, asserts "Request to Send" on RS232 Inf.
4	ctrl_ra	Wr	When 0, applies hard-reset to 21LV03/12LV31
5	ctrl_aen	Wr	When 1, asserts acitve-high "AEN" on PC/104 inf.
6	ctrl_refresh	Wr	When 1, asserts "REFRESH" on PC/104 interface
7	ctrl_tc	Wr	When 1, asserts "TC" (terminal count" on PC/104
8	ctrl_sbhe	Wr	When 1, asserts "SBHE" on PC/104 interface
9	ctrl_mbusy	Wr	"BUSY" input to 12LV31 (Pin #82)
10	ctrl_cntdr	Wr	"CONTENDR" input to 12LV31 (Pin #17)
11	ctrl_idrst	Wr	"IDMARST" input to 12LV31 (Pin #98)
12	ctrl_fiforst	Wr	When 0, resets 7819 ISO port FIFO
13	ctrl_isodir	Wr	When 0, ISO transfers are from ISO port to SRAM
14	ctrl_mcmd1	Wr	"MCMODE[1]" to 12LV31 (Pin #80)
15	ctrl_mcmd0	Wr	"MCMODE[0]" to 12LV31 (Pin #79)

Register Address: 0xF600
Register Mnemonic: la_rdytime
Short Description: PC/104 Sample Ready Delay

Register Description: A 4-bit hardware field whcih defines the number of clock cycles to delay sampling of the "ready" signal on the PC/104 interface. When 0, the PC/104 cycles will be minimal. Current value can be viewed and changed using a Monitor Menu command

Bit Number	Mnemonic	Read/Write	Description
0	TM0	Wr	Least Significant Timer bit
1	TM1	Wr	
2	TM2	Wr	
3	TM3	Wr	Most Significant Timer bit
4			Not Used
5			Not Used
6			Not Used
7			Not Used
8			Not Used
9			Not Used
10			Not Used
11			Not Used
12			Not Used
13			Not Used
14			Not Used
15			Not Used

12LV31/21LV03 IEEE 1394 Starter Kit Register detail

Register Address: 0xF700
Register Mnemonic: la_xfercntr
Short Description: DMA Transfer Counter

Register Description: This 8-bit quantity is loaded with the number of Quadlets (ie, 4-byte data values) transfered between SRAM and the 7819 ISO port FIFO. It counts down to zero during a DMA transfer. Note that the register is readable as well as writable

Bit Number	Mnemonic	Read/Write	Description
0	XFR0	Rd/Wr	Transfer counter least significant bit
1	XFR1	Rd/Wr	
2	XFR2	Rd/Wr	
3	XFR3	Rd/Wr	
4	XFR4	Rd/Wr	
5	XFR5	Rd/Wr	
6	XFR6	Rd/Wr	
7	XFR7	Rd/Wr	Transfer counter most significant bit
8			Not Used
9			Not Used
10			Not Used
11			Not Used
12			Not Used
13			Not Used
14			Not Used
15			Not Used

Register Address: 0xF800
Register Mnemonic: la_dmago
Short Description: Kick off DMA transfers

Register Description: A write to the register will start the DMA transfers between ISO port FIFO and the SRAM. The data associated with this write is meaningless.

Bit Number	Mnemonic	Read/Write	Description
0		Wr	Not Used
1		Wr	Not Used
2		Wr	Not Used
3		Wr	Not Used
4		Wr	Not Used
5		Wr	Not Used
6		Wr	Not Used
7		Wr	Not Used
8		Wr	Not Used
9		Wr	Not Used
10		Wr	Not Used
11		Wr	Not Used
12		Wr	Not Used
13		Wr	Not Used
14		Wr	Not Used
15		Wr	Not Used

12LV31/21LV03 IEEE 1394 Starter Kit Register detail

Register Address: 0xF900
Register Mnemonic: la_addrhi
Short Description: Load DMA address counter high byte

Register Description: The 16-bit DMA address counter is loaded into the Starter Kit hardware via two 8-bit writes. This particular write will load DMA address counter bits [15..8].

Bit Number	Mnemonic	Read/Write	Description
0	ADDR8	Wr	DMA transfer address bit [8]
1	ADDR9	Wr	DMA transfer address bit [9]
2	ADDR10	Wr	DMA transfer address bit [10]
3	ADDR11	Wr	DMA transfer address bit [11]
4	ADDR12	Wr	DMA transfer address bit [12]
5	ADDR13	Wr	DMA transfer address bit [13]
6	ADDR14	Wr	DMA transfer address bit [14]
7	ADDR15	Wr	DMA transfer address bit [15]
8			Not Used
9			Not Used
10			Not Used
11			Not Used
12			Not Used
13			Not Used
14			Not Used
15			Not Used

Register Address: 0xFA00
Register Mnemonic: la_addrlo
Short Description: Load DMA address counter low byte

Register Description: The 16-bit DMA address counter is loaded into the Starter Kit hardware via two 8-bit writes. This particular write will load DMA address counter bits [7..0].

Bit Number	Mnemonic	Read/Write	Description
0	ADDR0	Wr	DMA transfer address bit [0]
1	ADDR1	Wr	DMA transfer address bit [1]
2	ADDR2	Wr	DMA transfer address bit [2]
3	ADDR3	Wr	DMA transfer address bit [3]
4	ADDR4	Wr	DMA transfer address bit [4]
5	ADDR5	Wr	DMA transfer address bit [5]
6	ADDR6	Wr	DMA transfer address bit [6]
7	ADDR7	Wr	DMA transfer address bit [7]
8			Not Used
9			Not Used
10			Not Used
11			Not Used
12			Not Used
13			Not Used
14			Not Used
15			Not Used

12LV31/21LV03 IEEE 1394 Starter Kit Register detail

Register Address: 0xFB00
Register Mnemonic: la_pc104dat
Short Description: Access PC/104 space

Register Description: A read or write to this DSP Data Space address will perform an ISA bus cycle on the PC/104 interface. The address of the PC/104 access is set up by prior writes to the la_pc104adr and la_ldpg registers.

Bit Number	Mnemonic	Read/Write	Description
0	PC104D0	Rd/Wr	Least Significant PC/104 data bus bit
1	PC104D1	Rd/Wr	
2	PC104D2	Rd/Wr	
3	PC104D3	Rd/Wr	
4	PC104D4	Rd/Wr	
5	PC104D5	Rd/Wr	
6	PC104D6	Rd/Wr	
7	PC104D7	Rd/Wr	
8	PC104D8	Rd/Wr	
9	PC104D9	Rd/Wr	
10	PC104D10	Rd/Wr	
11	PC104D11	Rd/Wr	
12	PC104D12	Rd/Wr	
13	PC104D13	Rd/Wr	
14	PC104D14	Rd/Wr	
15	PC104D15	Rd/Wr	Most Significant PC/104 data bus bit

Register Address: 0xFC00
Register Mnemonic: la_dmactrl
Short Description: DMA control/status register + misc

Register Description: The register, located in the MACH445 PLD, has control for the DMA hardware. It also contains some other bits related to the PC/104 interface and 12LV31 handshake/pulse mode.

Bit Number	Mnemonic	Read/Write	Description
0	dmactrl_isotstmode	Rd/Wr	When set, puts ISO port FIFO in diag test mode
1	dmactrl_pc104md0	Rd/Wr	Establishes which space on PC/104 is accessed
2	dmactrl_pc104md1	Rd/Wr	Establishes which space on PC/104 is accessed
3	dmactrl_isodir	Rd/Wr	When 0, DMA direction is ISO port to SRAM
4	dmactrl_tstrw	Rd/Wr	Used during ISO port diag test mode to clock data
5	dmactrl_pulsemode	Rd/Wr	When set, use "Pulse" mode 12LV31 CS/CA timing
6			Not Used
7	dmastat_dmabusy	Rd	When set, DMA transfer is not yet completed (busy)
8			Not Used
9			Not Used
10			Not Used
11			Not Used
12			Not Used
13			Not Used
14			Not Used
15			Not Used

12LV31/21LV03 IEEE 1394 Starter Kit Register detail

Register Address: 0xFD00
Register Mnemonic: la_wrprio2
Short Description: Write PC/104 IRQ/DRQ select multiplexer for INT2 DSP input

Register Description: The Starter Kit hardware will map, under program control, the PC/104 DRQ and IRQ lines to three interrupt request lines going into the C52 DSP -- INT2, INT3, and INT 4. The Monitor program has a menu option to set up the mapping. Please use the Monitor menu option for more understanding.

Bit Number	Mnemonic	Read/Write	Description
0	I2SEL0	Wr	INT2 input select least significant bit
1	I2SEL1	Wr	
2	I2SEL2	Wr	
3	I2SEL3	Wr	INT2 input select most significant bit
4			Not Used
5			Not Used
6			Not Used
7			Not Used
8			Not Used
9			Not Used
10			Not Used
11			Not Used
12			Not Used
13			Not Used
14			Not Used
15			Not Used

Register Address: 0xFD01
Register Mnemonic: la_wrprio3
Short Description: Write PC/104 IRQ/DRQ select multiplexer for INT3 DSP input

Register Description: The Starter Kit hardware will map, under program control, the PC/104 DRQ and IRQ lines to three interrupt request lines going into the C52 DSP -- INT2, INT3, and INT 4. The Monitor program has a menu option to set up the mapping. Please use the Monitor menu option for more understanding.

Bit Number	Mnemonic	Read/Write	Description
0	I3SEL0	Wr	INT3 input select least significant bit
1	I3SEL1	Wr	
2	I3SEL2	Wr	
3	I3SEL3	Wr	INT3 input select most significant bit
4			Not Used
5			Not Used
6			Not Used
7			Not Used
8			Not Used
9			Not Used
10			Not Used
11			Not Used
12			Not Used
13			Not Used
14			Not Used
15			Not Used

12LV31/21LV03 IEEE 1394 Starter Kit Register detail

Register Address: 0xFD02
Register Mnemonic: la_wrprio4
Short Description: Write PC/104 IRQ/DRQ select multiplexer for INT4 DSP input

Register Description: The Starter Kit hardware will map, under program control, the PC/104 DRQ and IRQ lines to three interrupt request lines going into the C52 DSP -- INT2, INT3, and INT 4. The Monitor program has a menu option to set up the mapping. Please use the Monitor menu option for more understanding.

Bit Number	Mnemonic	Read/Write	Description
0	I4SEL0	Wr	INT4 input select least significant bit
1	I4SEL1	Wr	
2	I4SEL2	Wr	
3	I4SEL3	Wr	INT4 input select most significant bit
4			Not Used
5			Not Used
6			Not Used
7			Not Used
8			Not Used
9			Not Used
10			Not Used
11			Not Used
12			Not Used
13			Not Used
14			Not Used
15			Not Used

Register Address: 0xFE00
Register Mnemonic: la_ldpg
Short Description: Load PC/104 address register bits [23..16]

Register Description: The 24-bit PC/104 address is driven by two registers in the Starter Kit Hardware. A write to this address will load bits [23..16] into the PC/104 address register.

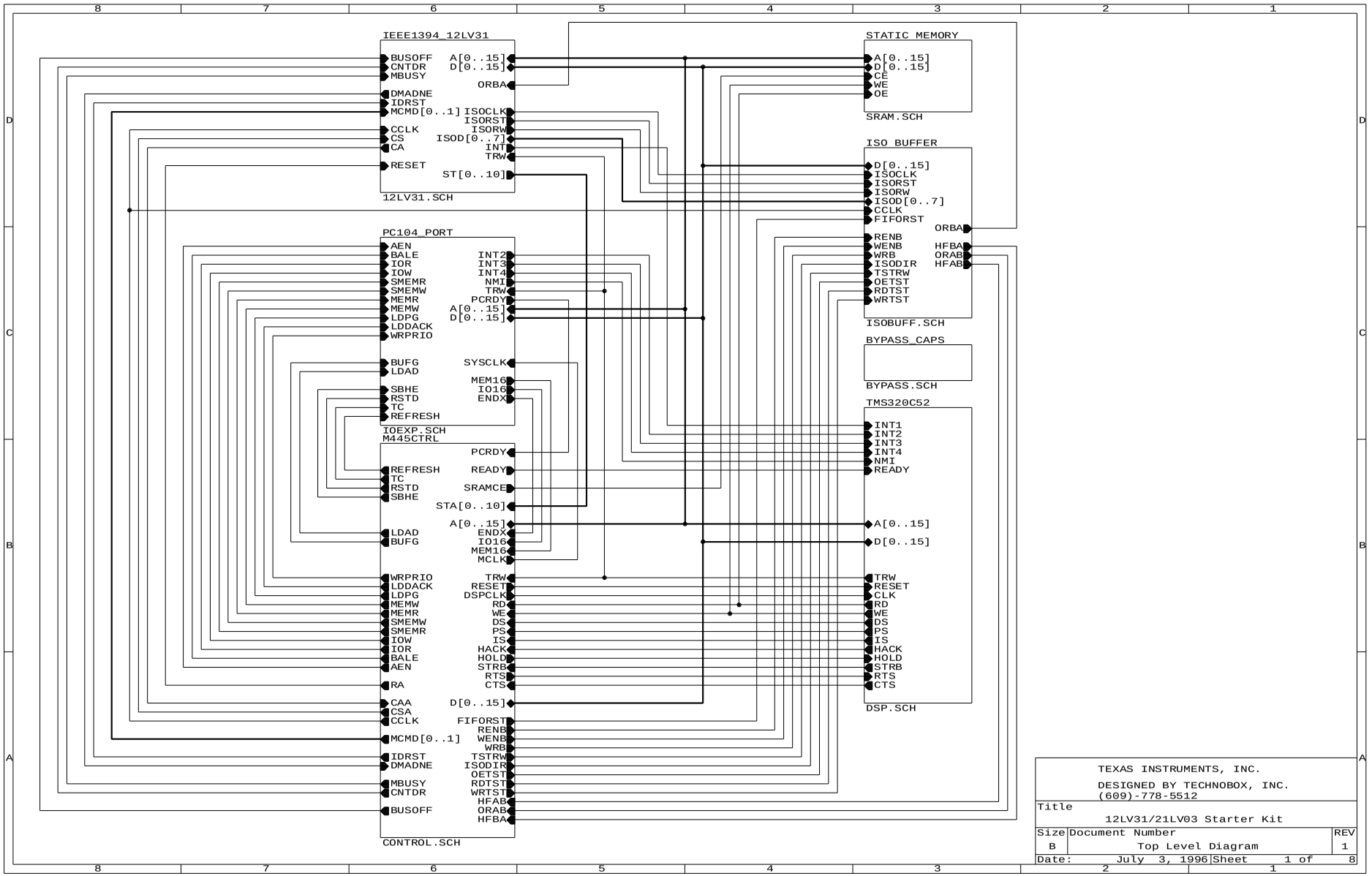
Bit Number	Mnemonic	Read/Write	Description
0	PA16	Wr	PC/104 Address bit [16]
1	PA17	Wr	PC/104 Address bit [17]
2	PA18	Wr	PC/104 Address bit [18]
3	PA19	Wr	PC/104 Address bit [19]
4	PA20	Wr	PC/104 Address bit [20]
5	PA21	Wr	PC/104 Address bit [21]
6	PA22	Wr	PC/104 Address bit [22]
7	PA23	Wr	PC/104 Address bit [23]
8			Not Used
9			Not Used
10			Not Used
11			Not Used
12			Not Used
13			Not Used
14			Not Used
15			Not Used

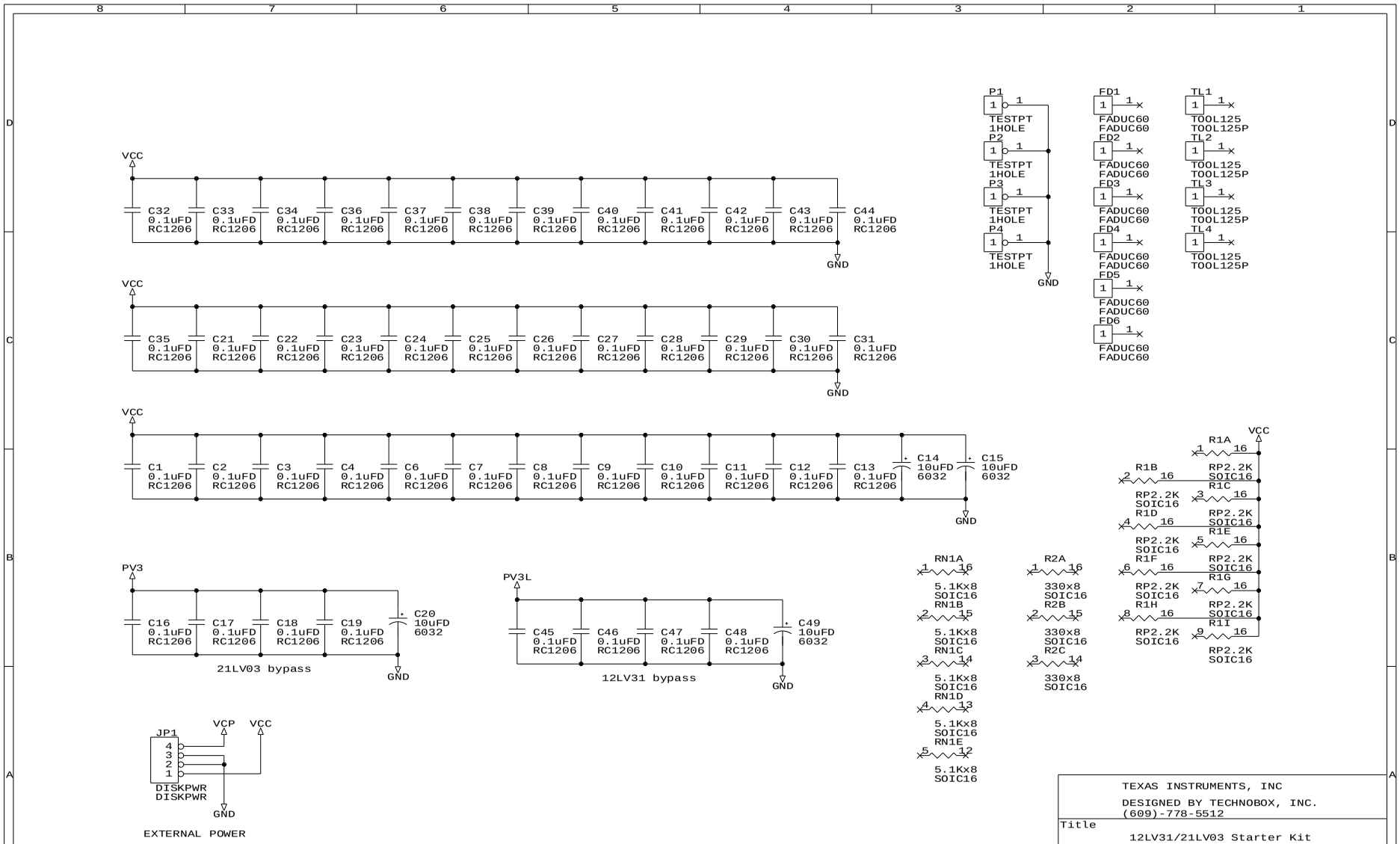
12LV31/21LV03 IEEE 1394 Starter Kit Register detail

Register Address: 0xFF00
Register Mnemonic: la_lddack
Short Description: Load PC/104 DACK register

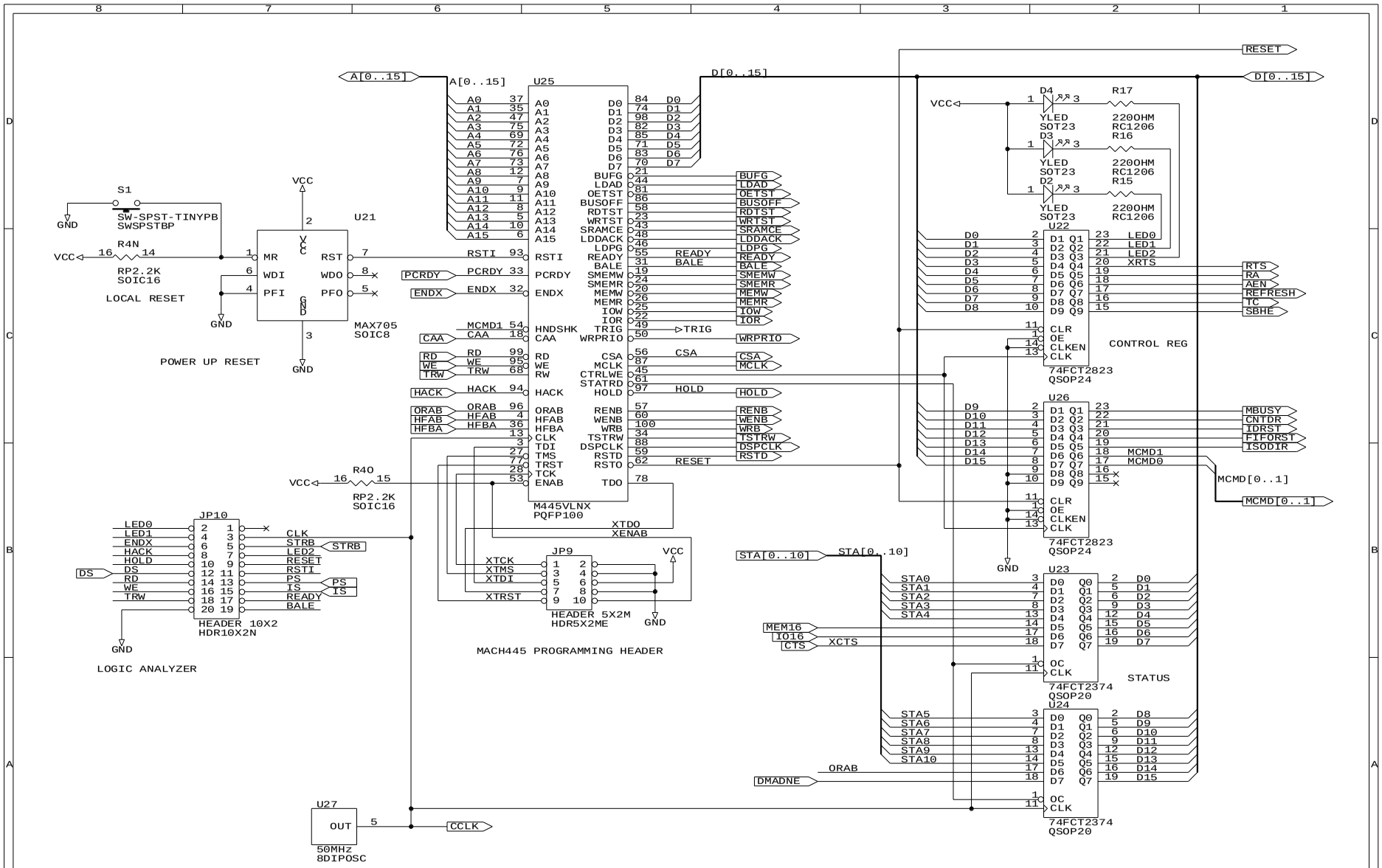
Register Description: The ISA function on the PC/104 bus includes a multi-channel DMA transfer capability which is managed by radially connected DRQ (DMA request) and DACK (DMA acknowledge) signals to DMA members on the ISA bus. This register drives the DACK lines. The corresponding DRQ lines is handled by la_wrprioX (X = 2,3,4)

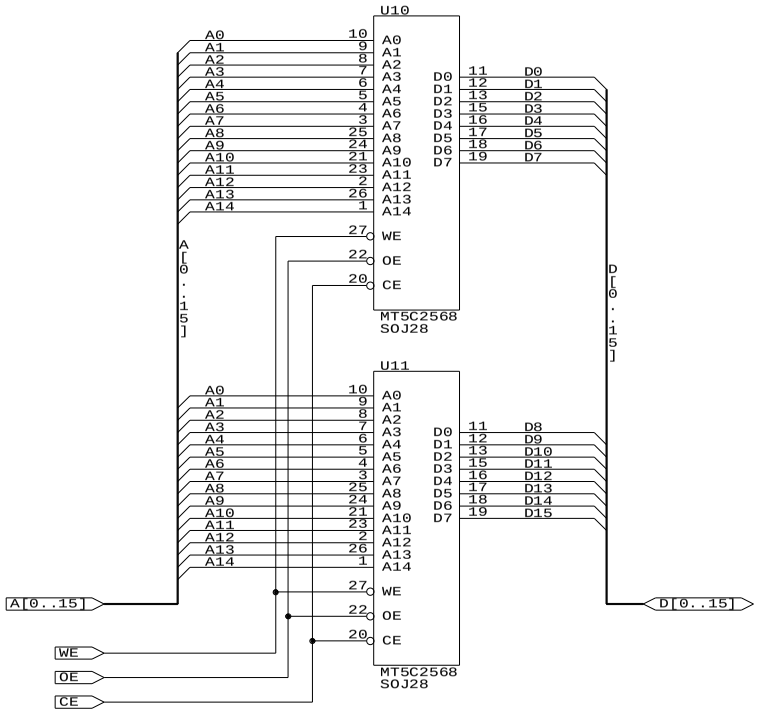
Bit Number	Mnemonic	Read/Write	Description
0	DACK0	Wr	PC/104 DMA acknowledge 0
1	DACK1	Wr	PC/104 DMA acknowledge 1
2	DACK2	Wr	PC/104 DMA acknowledge 2
3	DACK3	Wr	PC/104 DMA acknowledge 3
4	DACK4	Wr	PC/104 DMA acknowledge 4
5	DACK5	Wr	PC/104 DMA acknowledge 5
6	DACK6	Wr	PC/104 DMA acknowledge 6
7	DACK7	Wr	PC/104 DMA acknowledge 7
8			Not Used
9			Not Used
10			Not Used
11			Not Used
12			Not Used
13			Not Used
14			Not Used
15			Not Used


IEEE1394_12LV31
BUSOFF
CNTDR
MBUSY
DMADNE
IDRST
MCMD[0..1]
CCLK
CS
CA
RESET
A[0..15]
D[0..15]
ORBA
ISOCLK
ISORST
ISORW
ISOD[0..7]
INT
TRW
ST[0..10]
12LV31.SCH
STATIC MEMORY
A[0..15]
D[0..15]
CE
WE
OE
SRAM.SCH
ISO BUFFER
D[0..15]
ISOCLK
ISORST
ISORW
ISOD[0..7]
CCLK
FIFORST
ORBA
RENB
WENB
WRB
ISODIR
TSTRW
OETST
RDTST
WRTST
HFBA
ORAB
HFAB
ISOBUFF.SCH
BYPASS_CAPS
BYPASS.SCH
PC104_PORT
AEN
BALE
IOR
IOW
SMEMR
SMEMW
MEMR
MEMW
LDPG
LDDACK
WRPRIO
INT2
INT3
INT4
NMI
TRW
PCRDY
A[0..15]
D[0..15]
BUFG
LDAD
SYSCLK
MEM16
IO16
ENDX
SBHE
RSTD
TC
REFRESH
IOEXP.SCH
M445CTRL
PCRDY
REFRESH
TC
RSTD
SBHE
READY
SRAMCE
STA[0..10]
A[0..15]
LDAD
BUFG
ENDX
IO16
MEM16
MCLK
WRPRIO
LDDACK
LDPG
MEMW
MEMR
SMEMW
SMEMR
IOW
IOR
BALE
AEN
RA
CAA
CSA
CCLK
MCMD[0..1]
IDRST
DMADNE
MBUSY
CNTDR
BUSOFF
TRW
RESET
DSPCLK
RD
WE
DS
PS
IS
HACK
HOLD
STRB
RTS
CTS
D[0..15]
FIFORST
RENB
WENB
WRB
TSTRW
ISODIR
OETST
RDTST
WRTST
HFAB
ORAB
HFBA
CONTROL.SCH
TMS320C52
INT1
INT2
INT3
INT4
NMI
READY
A[0..15]
D[0..15]
TRW
RESET
CLK
RD
WE
DS
PS
IS
HACK
HOLD
STRB
RTS
CTS
DSP.SCH
TEXAS INSTRUMENTS, INC.
DESIGNED BY TECHNOBOX, INC.
(609)-778-5512
Title
12LV31/21LV03 Starter Kit
Size
B
Document Number
Top Level Diagram
REV
1
Date: July 3, 1996
Sheet 1 of 8

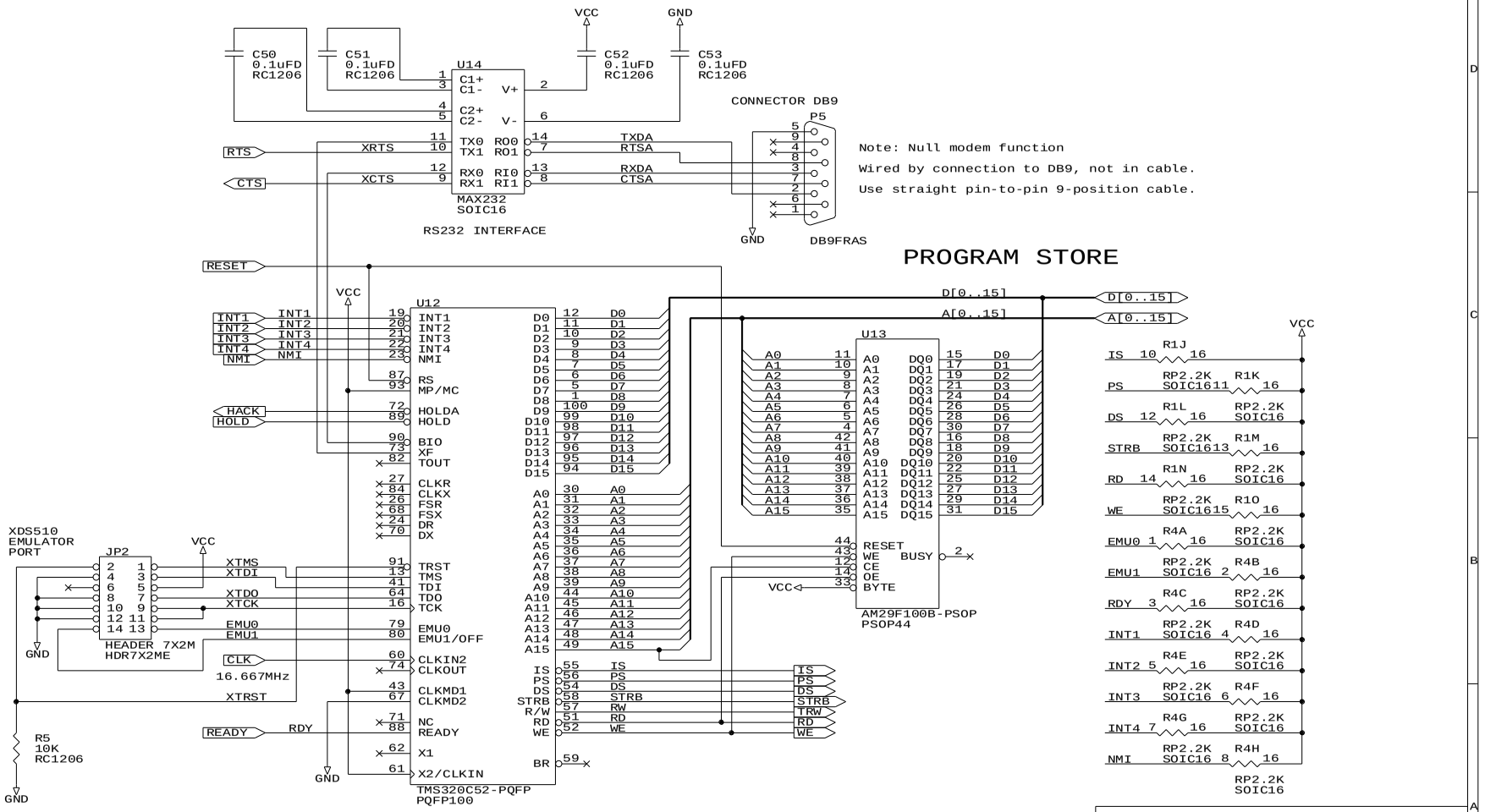


TEXAS INSTRUMENTS, INC DESIGNED BY TECHNOBOX, INC. (609)-778-5512		
Title: 12LV31/21LV03 Starter Kit		
Size: B	Document Number: Bypass Caps and Misc.	REV: 1
Date: July 7, 1996	Sheet 2 of 8	

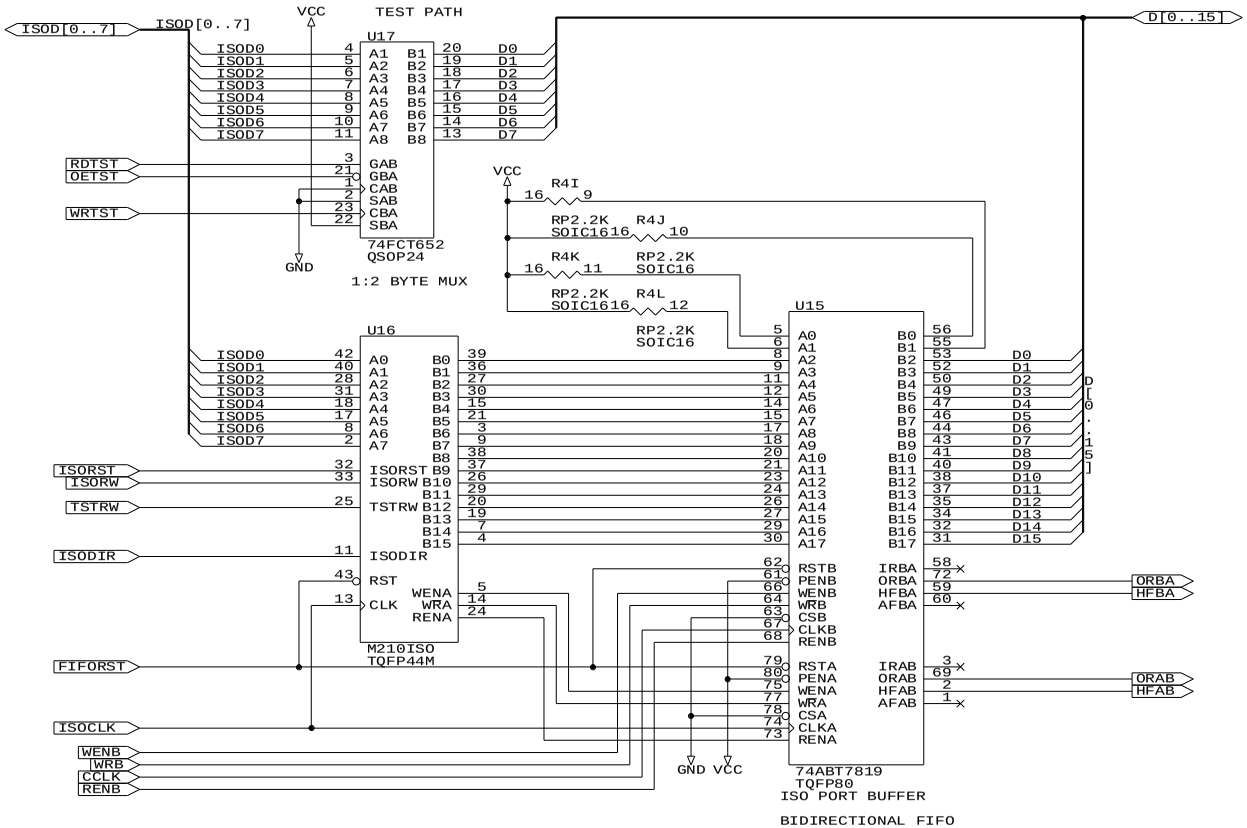

RESET
D[0..15]
A[0..15]
U25
M445VLNX
PQFP100
S1
SW-SPST-TINYPB
SWSPSTBP
R4N
RP2.2K
SOIC16
LOCAL RESET
U21
MAX705
SOIC8
POWER UP RESET
R4O
RP2.2K
SOIC16
JP10
HEADER 10X2
HDR10X2N
LOGIC ANALYZER
JP9
HEADER 5X2M
HDR5X2ME
MACH445 PROGRAMMING HEADER
U27
50MHz
8DIPOSC
CCLK
D4
R17
YLED
SOT23
220OHM
RC1206
D3
R16
D2
R15
U22
74FCT2823
QSOP24
CONTROL REG
U26
MCMD[0..1]
U23
74FCT2374
QSOP20
STATUS
U24
STA[0..10]
XCTS
ORAB
DMADNE
MEM16
IO16
CTS



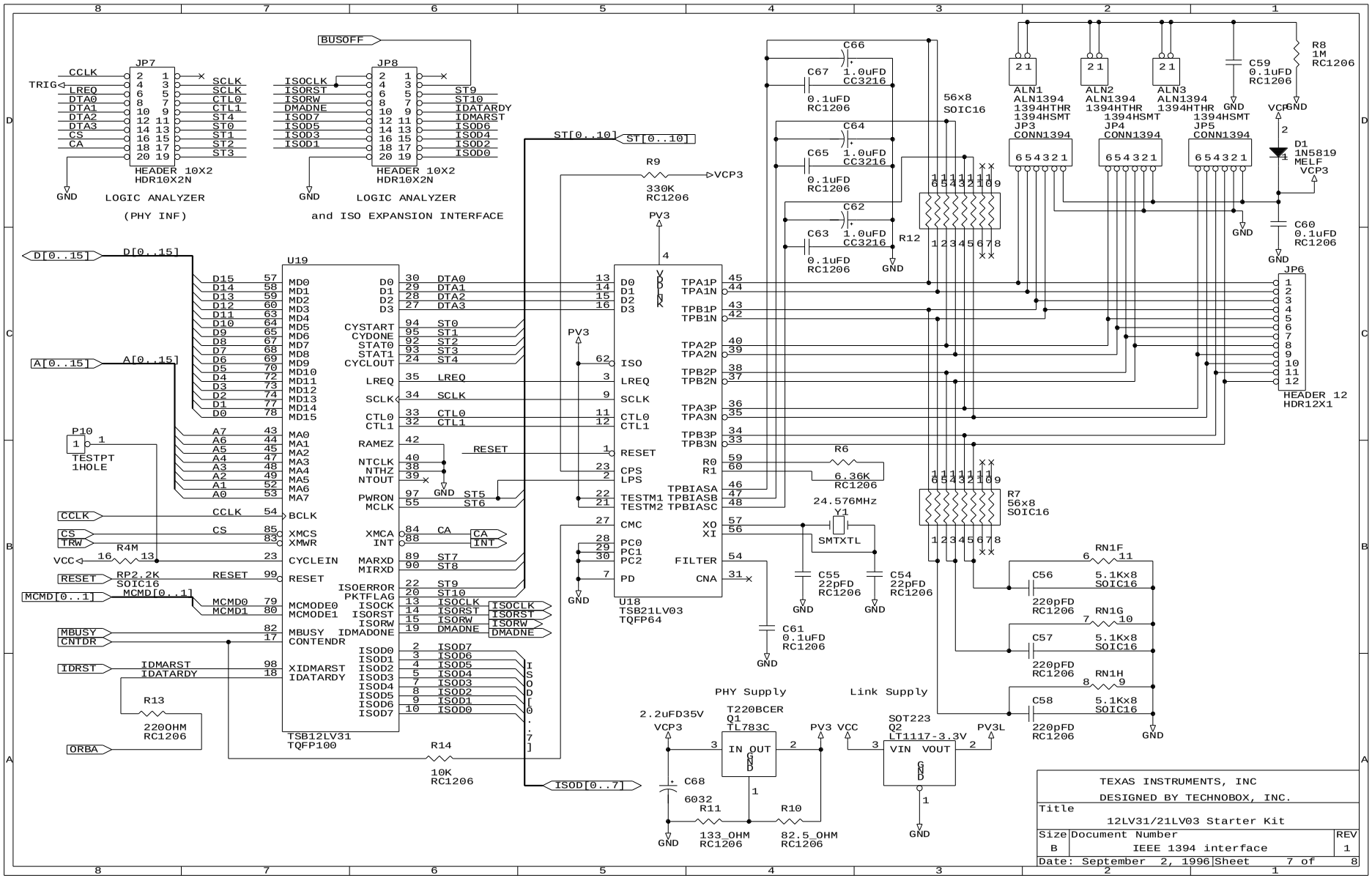
TEXAS INSTRUMENTS, INC DESIGNED BY TECHNOBOX, INC. (609)-778-5512		
Title 12LV31/21LV03 Starter Kit		
Size B	Document Number SRAM	REV 1
Date: July 3, 1996	Sheet 4 of 8	

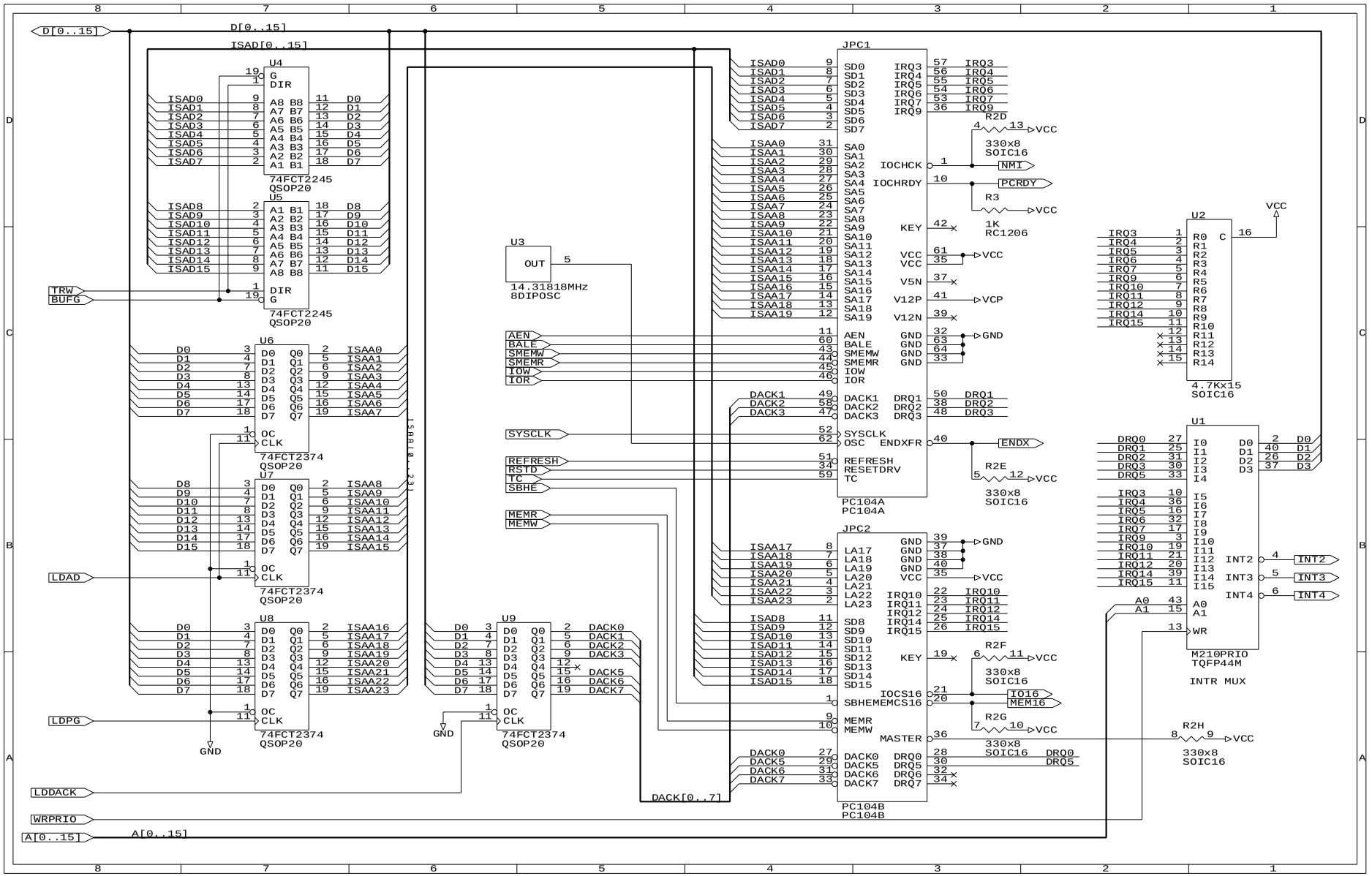


TEXAS INSTRUMENTS, INC DESIGNED BY TECHNOBOX, INC. (609)-778-5512		
Title: 12LV31/21LV03 Starter Kit		
Size: B	Document Number: DSP/FLASH/RS232	REV: 1
Date: July 3, 1996	Sheet 5 of 8	



TEXAS INSTRUMENTS, INC		
DESIGNED BY TECHNOBOX, INC.		
Title: 12LV31/21LV03 Starter Kit		
Size: B	Document Number: ISO port FIFO buffer	REV: 1
Date: August 5, 1996	Sheet 6 of 8	


JP7
HEADER 10X2
HDR10X2N
LOGIC ANALYZER
(PHY INF)
JP8
HEADER 10X2
HDR10X2N
LOGIC ANALYZER
and ISO EXPANSION INTERFACE
BUSOFF
U19
TSB12LV31
TQFP100
U18
TSB21LV03
TQFP64
R9
330K
RC1206
R12
56x8
SOIC16
R6
6.36K
RC1206
R7
56x8
SOIC16
24.576MHz
Y1
SMTXTL
R8
1M
RC1206
ALN1
ALN2
ALN3
ALN1394
1394HTHR
1394HSMT
JP3
JP4
JP5
CONN1394
D1
1N5819
MELF
VCP3
JP6
HEADER 12
HDR12X1
P10
TESTPT
1HOLE
R4M
RP2.2K
SOIC16
R13
2200HM
RC1206
R14
10K
RC1206
RN1F
RN1G
RN1H
5.1Kx8
SOIC16
C56
C57
C58
220pFD
RC1206
C55
C54
22pFD
RC1206
C61
0.1uFD
RC1206
C67
C64
C65
C62
C63
1.0uFD
CC3216
0.1uFD
RC1206
C59
C60
C66
C68
6032
2.2uFD35V
PHY Supply
Link Supply
Q1
TL783C
T220BCER
Q2
LT1117-3.3V
SOT223
R10
82.5_OHM
RC1206
R11
133_OHM
RC1206
TEXAS INSTRUMENTS, INC
DESIGNED BY TECHNOBOX, INC.
Title
12LV31/21LV03 Starter Kit
Size
B
Document Number
IEEE 1394 interface
REV
1
Date: September 2, 1996
Sheet
7 of
8


D[0..15]
ISAD[0..15]
U4
74FCT2245
QSOP20
U5
74FCT2245
QSOP20
TRW
BUFG
U6
74FCT2374
QSOP20
U7
74FCT2374
QSOP20
LDAD
U8
74FCT2374
QSOP20
LDPG
GND
U3
OUT
14.31818MHz
8DIPOSC
U9
74FCT2374
QSOP20
GND
LDDACK
WRPRIO
A[0..15]
ISAA[0..23]
DACK[0..7]
JPC1
PC104A
PC104A
AEN
BALE
SMEMW
SMEMR
IOW
IOR
SYSCLK
REFRESH
RSTD
TC
SBHE
MEMR
MEMW
IOCHCK
IOCHRDY
KEY
VCC
VCP
V5N
V12P
V12N
GND
DRQ1
DRQ2
DRQ3
ENDXFR
ENDX
R2D
R2E
R2F
R2G
R2H
330x8
SOIC16
1K
RC1206
NMI
PCRDY
R3
JPC2
PC104B
PC104B
IRQ10
IRQ11
IRQ12
IRQ14
IRQ15
IOCS16
SBHEMEMCS16
MASTER
DRQ0
DRQ5
U2
4.7Kx15
SOIC16
U1
M210PRIO
TQFP44M
INTR MUX
INT2
INT3
INT4

==========

TITLE : 1394 VersaLynx Starter Kit control PLD
FILE : C:\PLDXL33\m445glnx\m445glnx.mpf
DATE : Thu Dec 12 19:35:55 1996
ENGINEER : JOE NORRIS
COMPANY : TECHNOBOX, INC.
PROJECT : VERSA1394
REVISION : 1.0

==========

MODULES :

Document Generator	3.51
File Handler	3.3
Language Compiler	3.65
Architectural Optimizer	3.70
Device Lib Scan	3.93
Device Library	3.157
Device Partition	3.155
Device Fusemap	3.70

SWITCH VALUES :

(Value in parenthesis represents batch mode switch value)

PLCOMP PRODUCT TERM LIMIT : 512
PLOPT PRODUCT TERM LIMIT : 512
PLOPT REDUCTION : Espresso (1)
NODE GENERATION : Procedure Instantiation
Arithmetic and Relational Operators (1)

PLDocument: C:\PLDXL33\M445GLNX\M445GLNX.DOCEQUATIONS Thu Dec 12 19:35:55 1996

EQUATIONS FOR SYSTEM

INPUT SIGNALS (10) :

clk
LOW_TRUE reset_pin
LOW_TRUE endx_pin
LOW_TRUE hack
orab
hfab_pin
hfba_pin
trw
LOW_TRUE caa
pcrdy_pin

OUTPUT SIGNALS (57) :

dspclk
mclk
LOW_TRUE rsto
rstd
LOW_TRUE hold
busoff
rdtst
LOW_TRUE wrtst
LOW_TRUE oetst
renb
wenb
wrb
LOW_TRUE rd_pin
LOW_TRUE we_pin
d_pin[7..0]
isodir
a_pin[15..0]
LOW_TRUE trig
LOW_TRUE statrd
LOW_TRUE ctrlwe
LOW_TRUE sramce
LOW_TRUE csa
LOW_TRUE bufg
LOW_TRUE smemw
LOW_TRUE smemr
LOW_TRUE memw
LOW_TRUE memr
LOW_TRUE iow
LOW_TRUE ior
ready
ldpg
LOW_TRUE lddack
LOW_TRUE wrprio
LOW_TRUE ldad

bale

PHYSICAL NODE SIGNALS (76) :

reset
endx
synchack
hfba
rd
syncwr1
syncwr2
syncwe
flg_busy
doe
xfercntr[7..0]
ldxfer
decxfer
xferzero
dmactrl[5..0]
lddmactrl
rdytimer[3..0]
ldrdytime
selxfer
seldmactrl
timer[3..0]
timer_max
timer_loadrdy
a[15..0]
ldaddrhi
ldaddrlo
incaddr
acy
pcrdy
rdyclk
rdyset
rdysetx
startdma
startpc104
startlink
pagefx
dmastate[4..0]
balemach-0
balemach-1

REDUCED EQUATIONS:

dspclk.T = 1 ; "(1 term, 0 symbols)
dspclk.CLK = clk ; "(1 term, 1 symbol)

mclk.T = dspclk ; "(1 term, 1 symbol)
mclk.CLK = clk ; "(1 term, 1 symbol)

```
reset.D = reset_pin ;  "(1 term, 1 symbol)
reset.CLK = clk ;  "(1 term, 1 symbol)

rsto.D = reset ;  "(1 term, 1 symbol)
rsto.CLK = clk ;  "(1 term, 1 symbol)

rstd.D = rsto ;  "(1 term, 1 symbol)
rstd.CLK = clk ;  "(1 term, 1 symbol)

endx.D = endx_pin ;  "(1 term, 1 symbol)
endx.CLK = clk ;  "(1 term, 1 symbol)

hold.T = dmastate[0]*dmastate[1]*/dmastate[2]*
                dmastate[3]*/dmastate[4]*hold
            + /dmastate[0]*/dmastate[1]*
                /dmastate[2]*dmastate[3]*/dmastate[4]*hold
            + /dmastate[0]*/dmastate[1]*
                /dmastate[2]*/dmastate[3]*/dmastate[4]*
                flg_busy*/hfba*/hold*isodir*/startlink*
                /startpc104
            + /dmastate[0]*/dmastate[1]*
                /dmastate[2]*/dmastate[3]*/dmastate[4]*
                flg_busy*/hold*/isodir*orab*/startlink*
                /startpc104 ;  "(4 terms, 12 symbols)
hold.CLK = clk ;  "(1 term, 1 symbol)
hold.RESET = reset ;  "(1 term, 1 symbol)

synchack.D = hack ;  "(1 term, 1 symbol)
synchack.CLK = clk ;  "(1 term, 1 symbol)
synchack.RESET = reset ;  "(1 term, 1 symbol)

busoff.EQN(~) = /dmactrl[0]*/isodir ;  "(1 term, 2 symbols)

rdtst.EQN = /a_pin[10]*/a_pin[11]*a_pin[8]*
                /a_pin[9]*pagefx*rd_pin ;  "(1 term, 6 symbols)

wrtst.EQN = /a_pin[10]*/a_pin[11]*a_pin[8]*
                /a_pin[9]*pagefx*/rd_pin*we_pin ;  "(1 term, 7 symbols)

oetst.EQN = dmactrl[0]*/isodir ;  "(1 term, 2 symbols)

hfba.D = hfba_pin ;  "(1 term, 1 symbol)
hfba.CLK = clk ;  "(1 term, 1 symbol)

renb.EQN = dmastate[0]*/dmastate[1]*dmastate[2]*
                dmastate[3]*/dmastate[4]
            + /dmastate[0]*dmastate[1]*
                dmastate[2]*dmastate[3]*/dmastate[4] ;  "(2 terms, 5 symbols)

wenb.EQN = dmastate[0]*/dmastate[1]*dmastate[2]*
                /dmastate[3]*/dmastate[4]
            + /dmastate[0]*dmastate[1]*
                dmastate[2]*/dmastate[3]*/dmastate[4] ;  "(2 terms, 5 symbols)
```

```
wrb.T = dmastate[0]*dmastate[1]*/dmastate[2]*
                dmastate[3]*/dmastate[4]*/wrb
             + dmastate[0]*dmastate[1]*
                /dmastate[2]*/dmastate[3]*/dmastate[4]*
                /isodir*wrb
             + /dmastate[0]*/dmastate[1]*
                /dmastate[2]*dmastate[3]*/dmastate[4]*/wrb ;  "(3 terms, 7 symbols)
wrb.CLK = clk ;  "(1 term, 1 symbol)
wrb.PRESET = reset ;  "(1 term, 1 symbol)

rd_pin.EQN = rd ;  "(1 term, 1 symbol)
rd_pin.OE = hack ;  "(1 term, 1 symbol)

rd.T = dmastate[0]*dmastate[1]*/dmastate[2]*
                dmastate[3]*/dmastate[4]*rd
             + dmastate[0]*dmastate[1]*
                /dmastate[2]*/dmastate[3]*/dmastate[4]*isodir
                */rd
             + /dmastate[0]*/dmastate[1]*
                /dmastate[2]*dmastate[3]*/dmastate[4]*rd ;  "(3 terms, 7 symbols)
rd.CLK = clk ;  "(1 term, 1 symbol)
rd.RESET = reset ;  "(1 term, 1 symbol)

we_pin.EQN = /clk*dmastate[0]*dmastate[1]*
                dmastate[2]*dmastate[3]*/dmastate[4]
             + /clk*/dmastate[0]*/dmastate[1]*
                dmastate[2]*dmastate[3]*/dmastate[4] ;  "(2 terms, 6 symbols)
we_pin.OE = hack ;  "(1 term, 1 symbol)

syncwr1.D = we_pin ;  "(1 term, 1 symbol)
syncwr1.RESET = reset ;  "(1 term, 1 symbol)
syncwr1.CLK(~) = clk ;  "(1 term, 1 symbol)

syncwr2.D = syncwr1 ;  "(1 term, 1 symbol)
syncwr2.RESET = reset ;  "(1 term, 1 symbol)
syncwr2.CLK(~) = clk ;  "(1 term, 1 symbol)

syncwe.D = syncwr1*/syncwr2 ;  "(1 term, 2 symbols)
syncwe.CLK = clk ;  "(1 term, 1 symbol)
syncwe.RESET = reset ;  "(1 term, 1 symbol)

flg_busy.T = /dmastate[0]*/dmastate[1]*/dmastate[2]
                *dmastate[3]*/dmastate[4]*flg_busy
             + /dmastate[0]*/dmastate[1]*
                /dmastate[2]*/dmastate[3]*/dmastate[4]*
                /flg_busy*startdma*/startlink*/startpc104 ;  "(2 terms, 9 symbols)
flg_busy.CLK = clk ;  "(1 term, 1 symbol)
flg_busy.RESET = reset ;  "(1 term, 1 symbol)

doe.EQN = a_pin[10]*a_pin[11]*/a_pin[8]*
                /a_pin[9]*pagefx*rd_pin
             + a_pin[10]*/a_pin[11]*a_pin[8]*
                a_pin[9]*pagefx*rd_pin
             + /a_pin[10]*/a_pin[11]*/a_pin[8]*
```

```
                /a_pin[9]*pagefx*rd_pin ;  "(3 terms, 6 symbols)

d_pin[7].EQN = flg_busy*seldmactrl*/selxfer
             + selxfer*xfercntr[7] ;  "(2 terms, 4 symbols)
d_pin[7].OE = doe ;  "(1 term, 1 symbol)

d_pin[6].EQN = selxfer*xfercntr[6] ;  "(1 term, 2 symbols)
d_pin[6].OE = doe ;  "(1 term, 1 symbol)

d_pin[5].EQN = dmactrl[5]*seldmactrl*/selxfer
             + selxfer*xfercntr[5] ;  "(2 terms, 4 symbols)
d_pin[5].OE = doe ;  "(1 term, 1 symbol)

d_pin[4].EQN = dmactrl[4]*seldmactrl*/selxfer
             + selxfer*xfercntr[4] ;  "(2 terms, 4 symbols)
d_pin[4].OE = doe ;  "(1 term, 1 symbol)

d_pin[3].EQN = dmactrl[3]*seldmactrl*/selxfer
             + selxfer*xfercntr[3] ;  "(2 terms, 4 symbols)
d_pin[3].OE = doe ;  "(1 term, 1 symbol)

d_pin[2].EQN = dmactrl[2]*seldmactrl*/selxfer
             + selxfer*xfercntr[2] ;  "(2 terms, 4 symbols)
d_pin[2].OE = doe ;  "(1 term, 1 symbol)

d_pin[1].OE = doe ;  "(1 term, 1 symbol)
d_pin[1].EQN(~) = /dmactrl[1]*seldmactrl*/selxfer
             + selxfer*/xfercntr[1] ;  "(2 terms, 4 symbols)

d_pin[0].OE = doe ;  "(1 term, 1 symbol)
d_pin[0].EQN(~) = /dmactrl[0]*seldmactrl*/selxfer
             + selxfer*/xfercntr[0] ;  "(2 terms, 4 symbols)

xfercntr[7].T = d_pin[7]*ldxfer*/xfercntr[7]
             + /d_pin[7]*ldxfer*xfercntr[7]
             + decxfer*/ldxfer*/xfercntr[0]*
                /xfercntr[1]*/xfercntr[2]*/xfercntr[3]*
                /xfercntr[4]*/xfercntr[5]*/xfercntr[6] ;  "(3 terms, 11 symbols)
xfercntr[7].CLK = clk ;  "(1 term, 1 symbol)
xfercntr[7].RESET = reset ;  "(1 term, 1 symbol)

xfercntr[6].T = d_pin[6]*ldxfer*/xfercntr[6]
             + /d_pin[6]*ldxfer*xfercntr[6]
             + decxfer*/ldxfer*/xfercntr[0]*
                /xfercntr[1]*/xfercntr[2]*/xfercntr[3]*
                /xfercntr[4]*/xfercntr[5] ;  "(3 terms, 10 symbols)
xfercntr[6].CLK = clk ;  "(1 term, 1 symbol)
xfercntr[6].RESET = reset ;  "(1 term, 1 symbol)

xfercntr[5].T = d_pin[5]*ldxfer*/xfercntr[5]
             + /d_pin[5]*ldxfer*xfercntr[5]
             + decxfer*/ldxfer*/xfercntr[0]*
                /xfercntr[1]*/xfercntr[2]*/xfercntr[3]*
                /xfercntr[4] ;  "(3 terms, 9 symbols)
```

```
xfercntr[5].CLK = clk ;  "(1 term, 1 symbol)
xfercntr[5].RESET = reset ;  "(1 term, 1 symbol)

xfercntr[4].T = d_pin[4]*ldxfer*/xfercntr[4]
              + /d_pin[4]*ldxfer*xfercntr[4]
              + decxfer*/ldxfer*/xfercntr[0]*
                /xfercntr[1]*/xfercntr[2]*/xfercntr[3] ;  "(3 terms, 8 symbols)
xfercntr[4].CLK = clk ;  "(1 term, 1 symbol)
xfercntr[4].RESET = reset ;  "(1 term, 1 symbol)

xfercntr[3].T = d_pin[3]*ldxfer*/xfercntr[3]
              + /d_pin[3]*ldxfer*xfercntr[3]
              + decxfer*/ldxfer*/xfercntr[0]*
                /xfercntr[1]*/xfercntr[2] ;  "(3 terms, 7 symbols)
xfercntr[3].CLK = clk ;  "(1 term, 1 symbol)
xfercntr[3].RESET = reset ;  "(1 term, 1 symbol)

xfercntr[2].T = d_pin[2]*ldxfer*/xfercntr[2]
              + /d_pin[2]*ldxfer*xfercntr[2]
              + decxfer*/ldxfer*/xfercntr[0]*
                /xfercntr[1] ;  "(3 terms, 6 symbols)
xfercntr[2].CLK = clk ;  "(1 term, 1 symbol)
xfercntr[2].RESET = reset ;  "(1 term, 1 symbol)

xfercntr[1].T = d_pin[1]*ldxfer*/xfercntr[1]
              + /d_pin[1]*ldxfer*xfercntr[1]
              + decxfer*/ldxfer*/xfercntr[0] ;  "(3 terms, 5 symbols)
xfercntr[1].CLK = clk ;  "(1 term, 1 symbol)
xfercntr[1].RESET = reset ;  "(1 term, 1 symbol)

xfercntr[0].D = d_pin[0]*ldxfer
              + decxfer*/ldxfer*/xfercntr[0]
              + /decxfer*/ldxfer*xfercntr[0] ;  "(3 terms, 4 symbols)
xfercntr[0].CLK = clk ;  "(1 term, 1 symbol)
xfercntr[0].RESET = reset ;  "(1 term, 1 symbol)

ldxfer.EQN = a_pin[10]*/a_pin[11]*a_pin[8]*a_pin[9]
               *pagefx*/rd_pin*syncwe*we_pin ;  "(1 term, 8 symbols)

decxfer.EQN = dmastate[0]*/dmastate[1]*dmastate[2]*
                dmastate[3]*/dmastate[4]
              + /dmastate[0]*dmastate[1]*
                dmastate[2]*/dmastate[3]*/dmastate[4] ;  "(2 terms, 5 symbols)

xferzero.EQN = /xfercntr[0]*/xfercntr[1]*/xfercntr[2]
                */xfercntr[3]*/xfercntr[4]*
                /xfercntr[5]*/xfercntr[6]*/xfercntr[7] ;  "(1 term, 8 symbols)

dmactrl[5].D = d_pin[5]*lddmactrl
             + dmactrl[5]*/lddmactrl ;  "(2 terms, 3 symbols)
dmactrl[5].CLK = clk ;  "(1 term, 1 symbol)
dmactrl[5].RESET = reset ;  "(1 term, 1 symbol)

dmactrl[4].D = d_pin[4]*lddmactrl
```

```
            + dmactrl[4]*/lddmactrl ;  "(2 terms, 3 symbols)
dmactrl[4].CLK = clk ;  "(1 term, 1 symbol)
dmactrl[4].RESET = reset ;  "(1 term, 1 symbol)

dmactrl[3].D = d_pin[3]*lddmactrl
            + dmactrl[3]*/lddmactrl ;  "(2 terms, 3 symbols)
dmactrl[3].CLK = clk ;  "(1 term, 1 symbol)
dmactrl[3].RESET = reset ;  "(1 term, 1 symbol)

dmactrl[2].D = d_pin[2]*lddmactrl
            + dmactrl[2]*/lddmactrl ;  "(2 terms, 3 symbols)
dmactrl[2].CLK = clk ;  "(1 term, 1 symbol)
dmactrl[2].RESET = reset ;  "(1 term, 1 symbol)

dmactrl[1].D = d_pin[1]*lddmactrl
            + dmactrl[1]*/lddmactrl ;  "(2 terms, 3 symbols)
dmactrl[1].CLK = clk ;  "(1 term, 1 symbol)
dmactrl[1].RESET = reset ;  "(1 term, 1 symbol)

dmactrl[0].D = d_pin[0]*lddmactrl
            + dmactrl[0]*/lddmactrl ;  "(2 terms, 3 symbols)
dmactrl[0].CLK = clk ;  "(1 term, 1 symbol)
dmactrl[0].RESET = reset ;  "(1 term, 1 symbol)

lddmactrl.EQN = a_pin[10]*a_pin[11]*/a_pin[8]*
                /a_pin[9]*pagefx*/rd_pin*syncwe*we_pin ;  "(1 term, 8 symbols)

isodir.EQN = dmactrl[3] ;  "(1 term, 1 symbol)

rdytimer[3].T = d_pin[3]*ldrdytime*/rdytimer[3]*syncwe
            + /d_pin[3]*ldrdytime*rdytimer[3]*
              syncwe ;  "(2 terms, 4 symbols)
rdytimer[3].CLK = clk ;  "(1 term, 1 symbol)
rdytimer[3].RESET = reset ;  "(1 term, 1 symbol)

rdytimer[2].T = d_pin[2]*ldrdytime*/rdytimer[2]*syncwe
            + /d_pin[2]*ldrdytime*rdytimer[2]*
              syncwe ;  "(2 terms, 4 symbols)
rdytimer[2].CLK = clk ;  "(1 term, 1 symbol)
rdytimer[2].RESET = reset ;  "(1 term, 1 symbol)

rdytimer[1].T = d_pin[1]*ldrdytime*/rdytimer[1]*syncwe
            + /d_pin[1]*ldrdytime*rdytimer[1]*
              syncwe ;  "(2 terms, 4 symbols)
rdytimer[1].CLK = clk ;  "(1 term, 1 symbol)
rdytimer[1].RESET = reset ;  "(1 term, 1 symbol)

rdytimer[0].T = d_pin[0]*ldrdytime*/rdytimer[0]*syncwe
            + /d_pin[0]*ldrdytime*rdytimer[0]*
              syncwe ;  "(2 terms, 4 symbols)
rdytimer[0].CLK = clk ;  "(1 term, 1 symbol)
rdytimer[0].RESET = reset ;  "(1 term, 1 symbol)

ldrdytime.EQN = a_pin[10]*/a_pin[11]*/a_pin[8]*
```

```
                a_pin[9]*pagefx*/rd_pin*syncwe*we_pin ;  "(1 term, 8 symbols)

selxfer.EQN = a_pin[10]*/a_pin[11]*a_pin[8]*a_pin[9]
                *pagefx*rd_pin ;  "(1 term, 6 symbols)

seldmactrl.EQN = a_pin[10]*a_pin[11]*/a_pin[8]*
                /a_pin[9]*pagefx*rd_pin ;  "(1 term, 6 symbols)

timer[3].T = rdytimer[3]*timer[3]*timer_loadrdy
             + /rdytimer[3]*/timer[3]*
                timer_loadrdy
             + timer[0]*timer[1]*timer[2]*
                /timer_loadrdy*/timer_max ;  "(3 terms, 7 symbols)
timer[3].CLK = clk ;  "(1 term, 1 symbol)
timer[3].RESET = reset ;  "(1 term, 1 symbol)

timer[2].T = rdytimer[2]*timer[2]*timer_loadrdy
             + /rdytimer[2]*/timer[2]*
                timer_loadrdy
             + timer[0]*timer[1]*/timer_loadrdy*
                /timer_max ;  "(3 terms, 6 symbols)
timer[2].CLK = clk ;  "(1 term, 1 symbol)
timer[2].RESET = reset ;  "(1 term, 1 symbol)

timer[1].T = rdytimer[1]*timer[1]*timer_loadrdy
             + /rdytimer[1]*/timer[1]*
                timer_loadrdy
             + timer[0]*/timer_loadrdy*/timer_max ;  "(3 terms, 5 symbols)
timer[1].CLK = clk ;  "(1 term, 1 symbol)
timer[1].RESET = reset ;  "(1 term, 1 symbol)

timer[0].D = /rdytimer[0]*timer_loadrdy
             + timer[0]*/timer_loadrdy*timer_max
             + /timer[0]*/timer_loadrdy*/timer_max ;  "(3 terms, 4 symbols)
timer[0].CLK = clk ;  "(1 term, 1 symbol)
timer[0].RESET = reset ;  "(1 term, 1 symbol)

timer_max.EQN = timer[0]*timer[1]*timer[2]*timer[3] ;  "(1 term, 4 symbols)

timer_loadrdy.EQN = /dmastate[0]*/dmastate[1]*/dmastate[2]
                */dmastate[3]*/dmastate[4]*
                /startlink*startpc104 ;  "(1 term, 7 symbols)

a_pin[15].EQN = a[15] ;  "(1 term, 1 symbol)
a_pin[15].OE = hack ;  "(1 term, 1 symbol)

a_pin[14].EQN = a[14] ;  "(1 term, 1 symbol)
a_pin[14].OE = hack ;  "(1 term, 1 symbol)

a_pin[13].EQN = a[13] ;  "(1 term, 1 symbol)
a_pin[13].OE = hack ;  "(1 term, 1 symbol)

a_pin[12].EQN = a[12] ;  "(1 term, 1 symbol)
a_pin[12].OE = hack ;  "(1 term, 1 symbol)
```

```
a_pin[11].EQN = a[11] ;  "(1 term, 1 symbol)
a_pin[11].OE = hack ;  "(1 term, 1 symbol)

a_pin[10].EQN = a[10] ;  "(1 term, 1 symbol)
a_pin[10].OE = hack ;  "(1 term, 1 symbol)

a_pin[9].EQN = a[9] ;  "(1 term, 1 symbol)
a_pin[9].OE = hack ;  "(1 term, 1 symbol)

a_pin[8].EQN = a[8] ;  "(1 term, 1 symbol)
a_pin[8].OE = hack ;  "(1 term, 1 symbol)

a_pin[7].EQN = a[7] ;  "(1 term, 1 symbol)
a_pin[7].OE = hack ;  "(1 term, 1 symbol)

a_pin[6].EQN = a[6] ;  "(1 term, 1 symbol)
a_pin[6].OE = hack ;  "(1 term, 1 symbol)

a_pin[5].EQN = a[5] ;  "(1 term, 1 symbol)
a_pin[5].OE = hack ;  "(1 term, 1 symbol)

a_pin[4].EQN = a[4] ;  "(1 term, 1 symbol)
a_pin[4].OE = hack ;  "(1 term, 1 symbol)

a_pin[3].EQN = a[3] ;  "(1 term, 1 symbol)
a_pin[3].OE = hack ;  "(1 term, 1 symbol)

a_pin[2].EQN = a[2] ;  "(1 term, 1 symbol)
a_pin[2].OE = hack ;  "(1 term, 1 symbol)

a_pin[1].EQN = a[1] ;  "(1 term, 1 symbol)
a_pin[1].OE = hack ;  "(1 term, 1 symbol)

a_pin[0].EQN = a[0] ;  "(1 term, 1 symbol)
a_pin[0].OE = hack ;  "(1 term, 1 symbol)

a[15].T = a[10]*a[11]*a[12]*a[13]*a[14]*a[8]*
              a[9]*acy*incaddr*/ldaddrhi
          + a[15]*/d_pin[7]*ldaddrhi
          + /a[15]*d_pin[7]*ldaddrhi ;  "(3 terms, 12 symbols)
a[15].CLK = clk ;  "(1 term, 1 symbol)
a[15].RESET = reset ;  "(1 term, 1 symbol)

a[14].T = a[10]*a[11]*a[12]*a[13]*a[8]*a[9]*acy*
              incaddr*/ldaddrhi
          + a[14]*/d_pin[6]*ldaddrhi
          + /a[14]*d_pin[6]*ldaddrhi ;  "(3 terms, 11 symbols)
a[14].CLK = clk ;  "(1 term, 1 symbol)
a[14].RESET = reset ;  "(1 term, 1 symbol)

a[13].T = a[10]*a[11]*a[12]*a[8]*a[9]*acy*
              incaddr*/ldaddrhi
          + a[13]*/d_pin[5]*ldaddrhi
```

```
            + /a[13]*d_pin[5]*ldaddrhi ;  "(3 terms, 10 symbols)
a[13].CLK = clk ;  "(1 term, 1 symbol)
a[13].RESET = reset ;  "(1 term, 1 symbol)

a[12].T = a[10]*a[11]*a[8]*a[9]*acy*incaddr*
              /ldaddrhi
            + a[12]*/d_pin[4]*ldaddrhi
            + /a[12]*d_pin[4]*ldaddrhi ;  "(3 terms, 9 symbols)
a[12].CLK = clk ;  "(1 term, 1 symbol)
a[12].RESET = reset ;  "(1 term, 1 symbol)

a[11].T = a[10]*a[8]*a[9]*acy*incaddr*/ldaddrhi
            + a[11]*/d_pin[3]*ldaddrhi
            + /a[11]*d_pin[3]*ldaddrhi ;  "(3 terms, 8 symbols)
a[11].CLK = clk ;  "(1 term, 1 symbol)
a[11].RESET = reset ;  "(1 term, 1 symbol)

a[10].T = a[10]*/d_pin[2]*ldaddrhi
            + /a[10]*d_pin[2]*ldaddrhi
            + a[8]*a[9]*acy*incaddr*/ldaddrhi ;  "(3 terms, 7 symbols)
a[10].CLK = clk ;  "(1 term, 1 symbol)
a[10].RESET = reset ;  "(1 term, 1 symbol)

a[9].T = a[8]*acy*incaddr*/ldaddrhi
            + a[9]*/d_pin[1]*ldaddrhi
            + /a[9]*d_pin[1]*ldaddrhi ;  "(3 terms, 6 symbols)
a[9].CLK = clk ;  "(1 term, 1 symbol)
a[9].RESET = reset ;  "(1 term, 1 symbol)

a[8].T = a[8]*/d_pin[0]*ldaddrhi
            + /a[8]*d_pin[0]*ldaddrhi
            + acy*incaddr*/ldaddrhi ;  "(3 terms, 5 symbols)
a[8].CLK = clk ;  "(1 term, 1 symbol)
a[8].RESET = reset ;  "(1 term, 1 symbol)

a[7].T = a[0]*a[1]*a[2]*a[3]*a[4]*a[5]*a[6]*
              incaddr*/ldaddrlo
            + a[7]*/d_pin[7]*ldaddrlo
            + /a[7]*d_pin[7]*ldaddrlo ;  "(3 terms, 11 symbols)
a[7].CLK = clk ;  "(1 term, 1 symbol)
a[7].RESET = reset ;  "(1 term, 1 symbol)

a[6].T = a[0]*a[1]*a[2]*a[3]*a[4]*a[5]*incaddr*
              /ldaddrlo
            + a[6]*/d_pin[6]*ldaddrlo
            + /a[6]*d_pin[6]*ldaddrlo ;  "(3 terms, 10 symbols)
a[6].CLK = clk ;  "(1 term, 1 symbol)
a[6].RESET = reset ;  "(1 term, 1 symbol)

a[5].T = a[0]*a[1]*a[2]*a[3]*a[4]*incaddr*
              /ldaddrlo
            + a[5]*/d_pin[5]*ldaddrlo
            + /a[5]*d_pin[5]*ldaddrlo ;  "(3 terms, 9 symbols)
a[5].CLK = clk ;  "(1 term, 1 symbol)
```

```
a[5].RESET = reset ;  "(1 term, 1 symbol)

a[4].T = a[0]*a[1]*a[2]*a[3]*incaddr*/ldaddrlo
         + a[4]*/d_pin[4]*ldaddrlo
         + /a[4]*d_pin[4]*ldaddrlo ;  "(3 terms, 8 symbols)
a[4].CLK = clk ;  "(1 term, 1 symbol)
a[4].RESET = reset ;  "(1 term, 1 symbol)

a[3].T = a[0]*a[1]*a[2]*incaddr*/ldaddrlo
         + a[3]*/d_pin[3]*ldaddrlo
         + /a[3]*d_pin[3]*ldaddrlo ;  "(3 terms, 7 symbols)
a[3].CLK = clk ;  "(1 term, 1 symbol)
a[3].RESET = reset ;  "(1 term, 1 symbol)

a[2].T = a[0]*a[1]*incaddr*/ldaddrlo
         + a[2]*/d_pin[2]*ldaddrlo
         + /a[2]*d_pin[2]*ldaddrlo ;  "(3 terms, 6 symbols)
a[2].CLK = clk ;  "(1 term, 1 symbol)
a[2].RESET = reset ;  "(1 term, 1 symbol)

a[1].T = a[0]*incaddr*/ldaddrlo
         + a[1]*/d_pin[1]*ldaddrlo
         + /a[1]*d_pin[1]*ldaddrlo ;  "(3 terms, 5 symbols)
a[1].CLK = clk ;  "(1 term, 1 symbol)
a[1].RESET = reset ;  "(1 term, 1 symbol)

a[0].D = a[0]*/incaddr*/ldaddrlo
         + /a[0]*incaddr*/ldaddrlo
         + d_pin[0]*ldaddrlo ;  "(3 terms, 4 symbols)
a[0].CLK = clk ;  "(1 term, 1 symbol)
a[0].RESET = reset ;  "(1 term, 1 symbol)

ldaddrhi.EQN = /a_pin[10]*a_pin[11]*a_pin[8]*
               /a_pin[9]*pagefx*/rd_pin*syncwe*we_pin ;  "(1 term, 8 symbols)

ldaddrlo.EQN = /a_pin[10]*a_pin[11]*/a_pin[8]*
               a_pin[9]*pagefx*/rd_pin*syncwe*we_pin ;  "(1 term, 8 symbols)

incaddr.EQN = dmastate[0]*/dmastate[1]*dmastate[2]*
               /dmastate[4]
            + /dmastate[0]*dmastate[1]*
               dmastate[2]*/dmastate[3]*/dmastate[4]
            + /dmastate[0]*dmastate[1]*
               /dmastate[2]*dmastate[3]*/dmastate[4] ;  "(3 terms, 5 symbols)

acy.EQN = a[0]*a[1]*a[2]*a[3]*a[4]*a[5]*a[6]*
          a[7] ;  "(1 term, 8 symbols)

trig.EQN = a_pin[10]*/a_pin[11]*/a_pin[8]*
           /a_pin[9]*pagefx*/rd_pin*we_pin ;  "(1 term, 7 symbols)

statrd.EQN = a_pin[10]*/a_pin[11]*a_pin[8]*
             /a_pin[9]*pagefx*rd_pin ;  "(1 term, 6 symbols)
```

```
ctrlwe.EQN = a_pin[10]*/a_pin[11]*a_pin[8]*
             /a_pin[9]*pagefx*/rd_pin*we_pin ;  "(1 term, 7 symbols)

sramce.EQN = a_pin[15]*/pagefx ;  "(1 term, 2 symbols)

csa.D = /caa*dmastate[0]*dmastate[1]*
          /dmastate[2]*dmastate[3]*dmastate[4]
        + /dmastate[0]*/dmastate[1]*
          /dmastate[2]*/dmastate[3]*/dmastate[4]*
          startlink ;  "(2 terms, 7 symbols)
csa.CLK = clk ;  "(1 term, 1 symbol)

bufg.EQN = /a_pin[10]*a_pin[11]*a_pin[8]*a_pin[9]
             *pagefx*rd_pin
         + /a_pin[10]*a_pin[11]*a_pin[8]*
           a_pin[9]*pagefx*we_pin ;  "(2 terms, 7 symbols)

smemw.T = /dmactrl[1]*/dmactrl[2]*/dmastate[0]*
            /dmastate[1]*/dmastate[2]*/dmastate[3]*
            /dmastate[4]*/smemw*/startlink*startpc104*
            /trw
          + /dmastate[0]*dmastate[1]*
            dmastate[2]*dmastate[3]*dmastate[4]*pcrdy*
            smemw
          + /dmastate[0]*dmastate[1]*
            /dmastate[2]*dmastate[3]*dmastate[4]*smemw*
            timer_max
          + /dmastate[0]*dmastate[1]*
            dmastate[3]*dmastate[4]*endx*pcrdy*smemw ;  "(4 terms, 14 symbols)
smemw.CLK = clk ;  "(1 term, 1 symbol)
smemw.RESET = /bufg ;  "(1 term, 1 symbol)

smemr.T = /dmactrl[1]*/dmactrl[2]*/dmastate[0]*
            /dmastate[1]*/dmastate[2]*/dmastate[3]*
            /dmastate[4]*/smemr*/startlink*startpc104*trw ;  "(1 term, 11 symbols)
smemr.CLK = clk ;  "(1 term, 1 symbol)
smemr.RESET = /bufg ;  "(1 term, 1 symbol)

memw.T = dmactrl[1]*/dmactrl[2]*/dmastate[0]*
           /dmastate[1]*/dmastate[2]*/dmastate[3]*
           /dmastate[4]*/memw*/startlink*startpc104*/trw
         + /dmastate[0]*dmastate[1]*
           dmastate[2]*dmastate[3]*dmastate[4]*memw*
           pcrdy
         + /dmastate[0]*dmastate[1]*
           /dmastate[2]*dmastate[3]*dmastate[4]*memw*
           timer_max
         + /dmastate[0]*dmastate[1]*
           dmastate[3]*dmastate[4]*endx*memw*pcrdy ;  "(4 terms, 14 symbols)
memw.CLK = clk ;  "(1 term, 1 symbol)
memw.RESET = /bufg ;  "(1 term, 1 symbol)

memr.T = dmactrl[1]*/dmactrl[2]*/dmastate[0]*
           /dmastate[1]*/dmastate[2]*/dmastate[3]*
```

```
                /dmastate[4]*/memr*/startlink*startpc104*trw ;  "(1 term, 11 symbols)
memr.CLK = clk ;  "(1 term, 1 symbol)
memr.RESET = /bufg ;  "(1 term, 1 symbol)

iow.T = /dmactrl[1]*dmactrl[2]*/dmastate[0]*
                /dmastate[1]*/dmastate[2]*/dmastate[3]*
                /dmastate[4]*/iow*/startlink*startpc104*/trw
              + /dmastate[0]*dmastate[1]*
                dmastate[2]*dmastate[3]*dmastate[4]*iow*
                pcrdy
              + /dmastate[0]*dmastate[1]*
                /dmastate[2]*dmastate[3]*dmastate[4]*iow*
                timer_max
              + /dmastate[0]*dmastate[1]*
                dmastate[3]*dmastate[4]*endx*iow*pcrdy ;  "(4 terms, 14 symbols)
iow.CLK = clk ;  "(1 term, 1 symbol)
iow.RESET = /bufg ;  "(1 term, 1 symbol)

ior.T = /dmactrl[1]*dmactrl[2]*/dmastate[0]*
                /dmastate[1]*/dmastate[2]*/dmastate[3]*
                /dmastate[4]*/ior*/startlink*startpc104*trw ;  "(1 term, 11 symbols)
ior.CLK = clk ;  "(1 term, 1 symbol)
ior.RESET = /bufg ;  "(1 term, 1 symbol)

pcrdy.D = pcrdy_pin ;  "(1 term, 1 symbol)
pcrdy.CLK = clk ;  "(1 term, 1 symbol)

rdyclk.EQN = /a_pin[10]*a_pin[11]*a_pin[8]*a_pin[9]
                *pagefx*rd_pin
              + /a_pin[10]*a_pin[11]*a_pin[8]*
                a_pin[9]*pagefx*we_pin
              + /a_pin[10]*/a_pin[11]*/a_pin[8]*
                a_pin[9]*pagefx*rd_pin
              + /a_pin[10]*/a_pin[11]*/a_pin[8]*
                a_pin[9]*pagefx*we_pin ;  "(4 terms, 7 symbols)

rdyset.D = caa*/clk*dmastate[0]*/dmastate[1]*
                /dmastate[2]*dmastate[3]*dmastate[4]
              + /dmastate[0]*dmastate[1]*
                dmastate[2]*dmastate[3]*dmastate[4]*pcrdy
              + /dmastate[0]*dmastate[1]*
                /dmastate[2]*dmastate[3]*dmastate[4]*
                timer_max
              + /dmastate[0]*dmastate[1]*
                dmastate[3]*dmastate[4]*endx*pcrdy ;  "(4 terms, 10 symbols)
rdyset.CLK = clk ;  "(1 term, 1 symbol)
rdyset.RESET = reset ;  "(1 term, 1 symbol)

rdysetx.EQN(~) = /rdyset*/reset ;  "(1 term, 2 symbols)

ready.D = 0 ;  "(1 term, 0 symbols)
ready.CLK = rdyclk ;  "(1 term, 1 symbol)
ready.PRESET = rdysetx ;  "(1 term, 1 symbol)
```

```
ldpg.EQN = a_pin[10]*a_pin[11]*/a_pin[8]*a_pin[9]
             *pagefx*/rd_pin*syncwe*we_pin ;  "(1 term, 8 symbols)

lddack.EQN = a_pin[10]*a_pin[11]*a_pin[8]*a_pin[9]*
             pagefx*/rd_pin*we_pin ;  "(1 term, 7 symbols)

wrprio.EQN = a_pin[10]*a_pin[11]*a_pin[8]*/a_pin[9]
             *pagefx*/rd_pin*we_pin ;  "(1 term, 7 symbols)

ldad.EQN = /a_pin[10]*/a_pin[11]*a_pin[8]*
             a_pin[9]*pagefx*/rd_pin*we_pin ;  "(1 term, 7 symbols)

bale.D = balemach-0*/balemach-1
         + /balemach-1*ldpg ;  "(2 terms, 3 symbols)
bale.CLK = clk ;  "(1 term, 1 symbol)
bale.RESET = reset ;  "(1 term, 1 symbol)

startdma.D = /a_pin[10]*a_pin[11]*/a_pin[8]*
             /a_pin[9]*pagefx*/rd_pin*syncwe*we_pin ;  "(1 term, 8 symbols)
startdma.CLK = clk ;  "(1 term, 1 symbol)
startdma.RESET = reset ;  "(1 term, 1 symbol)

startpc104.D = /a_pin[10]*a_pin[11]*a_pin[8]*a_pin[9]
             *pagefx*rd_pin
           + /a_pin[10]*a_pin[11]*a_pin[8]*
             a_pin[9]*pagefx*we_pin ;  "(2 terms, 7 symbols)
startpc104.CLK = clk ;  "(1 term, 1 symbol)
startpc104.RESET = reset ;  "(1 term, 1 symbol)

startlink.D = /a_pin[10]*/a_pin[11]*/a_pin[8]*
             a_pin[9]*pagefx*rd_pin
           + /a_pin[10]*/a_pin[11]*/a_pin[8]*
             a_pin[9]*pagefx*we_pin ;  "(2 terms, 7 symbols)
startlink.CLK = clk ;  "(1 term, 1 symbol)
startlink.RESET = reset ;  "(1 term, 1 symbol)

pagefx.EQN = a_pin[12]*a_pin[13]*a_pin[14]*
             a_pin[15] ;  "(1 term, 4 symbols)

dmastate[4].T = dmastate[0]*/dmastate[1]*dmastate[4]*
             /startlink
           + dmastate[0]*dmastate[2]*dmastate[4]
             */startpc104
           + /dmastate[0]*/dmastate[1]*
             /dmastate[2]*dmastate[3]*/dmastate[4]
           + /dmastate[0]*/dmastate[1]*
             /dmastate[2]*/dmastate[3]*startlink
           + /dmastate[0]*/dmastate[1]*
             /dmastate[2]*/dmastate[3]*startpc104
           + /dmastate[0]*/dmastate[1]*
             dmastate[4]*/synchack ;  "(6 terms, 8 symbols)
dmastate[4].CLK = clk ;  "(1 term, 1 symbol)
dmastate[4].RESET = reset ;  "(1 term, 1 symbol)
```

```
dmastate[3].T = dmastate[0]*dmastate[1]*/dmastate[2]*
                  /dmastate[3]*/isodir
                + dmastate[0]*/dmastate[1]*
                  dmastate[2]*/dmastate[3]*hfba
                + dmastate[0]*/dmastate[1]*
                  dmastate[2]*/dmastate[3]*xferzero
                + dmastate[0]*/dmastate[1]*
                  /dmastate[2]*dmastate[3]*/dmastate[4]*
                  /synchack
                + dmastate[0]*/dmastate[1]*
                  dmastate[4]*/startlink
                + dmastate[0]*dmastate[2]*dmastate[4]
                  */startpc104
                + /dmastate[0]*/dmastate[1]*
                  /dmastate[2]*/dmastate[3]*startlink
                + /dmastate[0]*/dmastate[1]*
                  /dmastate[2]*/dmastate[3]*startpc104
                + /dmastate[0]*/dmastate[1]*
                  dmastate[4]*/synchack ;  "(9 terms, 11 symbols)
dmastate[3].CLK = clk ;  "(1 term, 1 symbol)
dmastate[3].RESET = reset ;  "(1 term, 1 symbol)

dmastate[2].CLK = clk ;  "(1 term, 1 symbol)
dmastate[2].RESET = reset ;  "(1 term, 1 symbol)
dmastate[2].D(~) = dmastate[0]*/dmastate[1]*/dmastate[3]*
                  hfba
                + dmastate[0]*/dmastate[1]*
                  /dmastate[3]*xferzero
                + dmastate[0]*/dmastate[2]*
                  dmastate[3]
                + dmastate[0]*/dmastate[2]*isodir
                + dmastate[0]*dmastate[4]*/startpc104
                + /dmastate[0]*dmastate[1]*
                  dmastate[2]*dmastate[3]*/dmastate[4]
                + /dmastate[1]*/dmastate[2]
                + /dmastate[2]*dmastate[3]*
                  /dmastate[4]*/orab
                + /dmastate[2]*dmastate[3]*
                  /dmastate[4]*xferzero
                + /dmastate[2]*dmastate[4]*/endx*
                  pcrdy*/timer_max ;  "(10 terms, 13 symbols)

dmastate[1].D = /caa*dmastate[1]*/dmastate[2]*
                  dmastate[4]
                + /dmactrl[5]*/dmastate[0]*
                  /dmastate[3]*startlink
                + dmastate[0]*/dmastate[1]*
                  dmastate[2]*hfba*/xferzero
                + dmastate[0]*/dmastate[1]*
                  /dmastate[2]*/dmastate[3]*synchack
                + dmastate[0]*dmastate[2]*dmastate[3]
                  */dmastate[4]
                + /dmastate[0]*dmastate[1]*
                  dmastate[2]
```

```
                + /dmastate[0]*dmastate[1]*
                  /dmastate[3]
                + /dmastate[0]*dmastate[1]*
                  dmastate[4]
                + /dmastate[0]*dmastate[1]*/orab*
                  /xferzero
                + /dmastate[0]*dmastate[2]*
                  /dmastate[3]
                + /dmastate[0]*/dmastate[3]*
                  /startlink*startpc104
                + dmastate[1]*/dmastate[2]*
                  /dmastate[3]*isodir
                + dmastate[2]*dmastate[4]*startpc104 ;  "(13 terms, 14 symbols)
dmastate[1].CLK = clk ;  "(1 term, 1 symbol)
dmastate[1].RESET = reset ;  "(1 term, 1 symbol)

dmastate[0].D = dmastate[0]*dmastate[1]*/dmastate[2]*
                  dmastate[3]
                + dmastate[0]*dmastate[1]*dmastate[4]
                  *startpc104
                + dmastate[0]*/dmastate[1]*
                  dmastate[2]*hfba*/xferzero
                + dmastate[0]*/dmastate[1]*
                  /dmastate[2]*/dmastate[3]
                + dmastate[0]*/dmastate[1]*
                  /dmastate[2]*/dmastate[4]*synchack
                + dmastate[0]*/dmastate[1]*
                  dmastate[4]*startlink
                + /dmastate[0]*dmastate[1]*
                  dmastate[4]*endx*pcrdy
                + /dmastate[0]*dmastate[2]*
                  dmastate[4]*pcrdy
                + dmastate[1]*dmastate[2]*
                  /dmastate[3]
                + dmastate[1]*/dmastate[2]*
                  dmastate[3]*/dmastate[4]*/orab*/xferzero
                + dmastate[1]*/dmastate[2]*
                  dmastate[4]*timer_max
                + /dmastate[1]*dmastate[2]*
                  dmastate[3]
                + /dmastate[1]*/dmastate[2]*
                  /dmastate[3]*flg_busy*/hfba*isodir*
                  /startpc104
                + /dmastate[1]*/dmastate[2]*
                  /dmastate[3]*flg_busy*/isodir*orab*
                  /startpc104
                + /dmastate[1]*/dmastate[2]*
                  /dmastate[3]*startlink ;  "(15 terms, 16 symbols)
dmastate[0].CLK = clk ;  "(1 term, 1 symbol)
dmastate[0].RESET = reset ;  "(1 term, 1 symbol)

balemach-0.D = /balemach-0*/balemach-1*ldpg ;  "(1 term, 3 symbols)
balemach-0.CLK = clk ;  "(1 term, 1 symbol)
balemach-0.RESET = reset ;  "(1 term, 1 symbol)
```

```
balemach-1.D = balemach-0 ;  "(1 term, 1 symbol)
balemach-1.CLK = clk ;  "(1 term, 1 symbol)
balemach-1.RESET = reset ;  "(1 term, 1 symbol)
```

PLDocument: C:\PLDXL33\M445GLNX\M445GLNX.DOCSOLUTIONS Thu Dec 12 19:35:55 1996

PARTITIONING CRITERIA :

WEIGHT PRICE 10 ;

TEMPLATE = MACH445 ;

PARTITIONING SOLUTIONS :

==> Solution 1: MACH445

FUSEMAP FILES FOR SOLUTION 1:

Device 1 (MACH445) : C:\PLDXL33\M445GLNX\M445GLNX.J1

Device 1 - MACH445 -- Pinout for QFP package

Pin	Type	Signal	Pin	Type	Signal
1	GND		51	GND	
2	GND		52	GND	
3	N/C		53	N/C	
4	Input		54	Input	NO-CONNECT
5	Biput	a_pin[13]	55	Biput	ready
6	Biput	a_pin[15]	56	Biput	csa
7	Biput	a_pin[9]	57	Biput	renb
8	Biput	a_pin[12]	58	Biput	rdtst
9	Biput	a_pin[10]	59	Biput	rstd
10	Biput	a_pin[14]	60	Biput	wenb
11	Biput	a_pin[11]	61	Biput	statrd
12	Biput	a_pin[8]	62	Biput	rsto
13	In/CLK	clk	63	In/CLK	
14	Vcc		64	Vcc	
15	Vcc		65	Vcc	
16	GND		66	GND	
17	GND		67	GND	
18	In/CLK	caa	68	In/CLK	trw
19	Biput	smemw	69	Biput	a_pin[4]
20	Biput	memw	70	Biput	d_pin[7]
21	Biput	bufg	71	Biput	d_pin[5]
22	Biput	ior	72	Biput	a_pin[5]
23	Biput	wrtst	73	Biput	a_pin[7]
24	Biput	smemr	74	Biput	d_pin[1]
25	Biput	iow	75	Biput	a_pin[3]
26	Biput	memr	76	Biput	a_pin[6]
27	N/C		77	N/C	
28	N/C		78	N/C	
29	GND		79	GND	
30	GND		80	GND	
31	Biput	bale	81	Biput	oetst
32	Biput	endx_pin	82	Biput	d_pin[3]
33	Biput	pcrdy_pin	83	Biput	d_pin[6]
34	Biput	isodir	84	Biput	d_pin[0]
35	Biput	a_pin[1]	85	Biput	d_pin[4]
36	Biput	hfba_pin	86	Biput	busoff
37	Biput	a_pin[0]	87	Biput	mclk
38	Biput		88	Biput	dspclk
39	Vcc		89	Vcc	
40	GND		90	GND	
41	GND		91	GND	
42	Vcc		92	Vcc	
43	Biput	sramce	93	Biput	reset_pin
44	Biput	ldad	94	Biput	hack
45	Biput	ctrlwe	95	Biput	we_pin

```
|  46 |Biput |  ldpg              | |  96 |Biput |  orab              |
|  47 |Biput |  a_pin[2]          | |  97 |Biput |  hold              |
|  48 |Biput |  lddack            | |  98 |Biput |  d_pin[2]          |
|  49 |Biput |  trig              | |  99 |Biput |  rd_pin            |
|  50 |Biput |  wrprio            | | 100 |Biput |  wrb               |
+-----+------+--------------------+ +-----+------+--------------------+
```

DEVICE SELECTION:

	Device	Manuf	Fam	Pack	Temp	ICC	TPD	Fmax	Price	User
1)	MACH445-20YC	AMD	CMOS	QFP	COM	576.0ma	25.0ns	40.0MHz	$ 56.95	0 0
2)	MACH445-15YC	AMD	CMOS	QFP	COM	589.0ma	20.0ns	50.0MHz	$ 59.95	0 0
==>	MACH445-12YC	AMD	CMOS	QFP	COM	710.0ma	19.0ns	52.6MHz	$ 77.95	0 0

PLDocument: C:\PLDXL33\M445GLNX\M445GLNX.DOCWIRELIST Thu Dec 12 19:35:55 1996

Signal	Device	Pin
NO-CONNECT	MACH445_1	54
clk	MACH445_1	13
dspclk	MACH445_1	88
mclk	MACH445_1	87
reset_pin	MACH445_1	93
rsto	MACH445_1	62
rstd	MACH445_1	59
endx_pin	MACH445_1	32
hack	MACH445_1	94
hold	MACH445_1	97
busoff	MACH445_1	86
rdtst	MACH445_1	58
wrtst	MACH445_1	23
oetst	MACH445_1	81
orab	MACH445_1	96
hfba_pin	MACH445_1	36
renb	MACH445_1	57
wenb	MACH445_1	60
wrb	MACH445_1	100
rd_pin	MACH445_1	99
we_pin	MACH445_1	95
trw	MACH445_1	68
d_pin[7]	MACH445_1	70
d_pin[6]	MACH445_1	83
d_pin[5]	MACH445_1	71
d_pin[4]	MACH445_1	85
d_pin[3]	MACH445_1	82
d_pin[2]	MACH445_1	98
d_pin[1]	MACH445_1	74
d_pin[0]	MACH445_1	84
isodir	MACH445_1	34
a_pin[15]	MACH445_1	6
a_pin[14]	MACH445_1	10
a_pin[13]	MACH445_1	5
a_pin[12]	MACH445_1	8
a_pin[11]	MACH445_1	11
a_pin[10]	MACH445_1	9
a_pin[9]	MACH445_1	7
a_pin[8]	MACH445_1	12
a_pin[7]	MACH445_1	73
a_pin[6]	MACH445_1	76
a_pin[5]	MACH445_1	72
a_pin[4]	MACH445_1	69
a_pin[3]	MACH445_1	75
a_pin[2]	MACH445_1	47
a_pin[1]	MACH445_1	35
a_pin[0]	MACH445_1	37
trig	MACH445_1	49
statrd	MACH445_1	61

| ctrlwe | MACH445_1 | 45 |
| sramce | MACH445_1 | 43 |
+---+---+---+

Signal	Device	Pin
caa	MACH445_1	18
csa	MACH445_1	56
bufg	MACH445_1	21
smemw	MACH445_1	19
smemr	MACH445_1	24
memw	MACH445_1	20
memr	MACH445_1	26
iow	MACH445_1	25
ior	MACH445_1	22
pcrdy_pin	MACH445_1	33
ready	MACH445_1	55
ldpg	MACH445_1	46
lddack	MACH445_1	48
wrprio	MACH445_1	50
ldad	MACH445_1	44
bale	MACH445_1	31

Please quote assembly service for the attached SMT printed circuit design.

Fax you quotation to (609)-778-5512, Attn: Joe Norris, Technobox, Inc.

1. Quantity 50 will be built, with purchase order issued to cover entire production lot. However, qty 3 prototypes out of the 50 piece lot will be assembled first, followed by build of the rest of the lot.
2. Timeframe: Qty 3 prototypes turned in 3 days, with delivery 1st week of April. Balance in April/May timeframe.
3. Observe doulbed sided SMT with thru hole. Thru hole connections will need to be hand soldered in this case. Approximately 200 thru-hole connections are on this board.
4. Technobox to provide all parts. If assembler wishes to automate this production, Technobox may provided parts in Tape/reel and tube/trays. Otherwise, parts will be provided bulk.
5. Assembler is to procure Solder Paste stencil - one for both sides of board. The stencil will be considered property of Technobox, and will be delivered with finished boards.
6. Technobox to provide solder paste stencil Gerber files in RS274-X format.
7. Please quote Assembly charge for prototype, production and solder paste stencil.

Parts list is as follows:

Futurebus to VMEbus Adapter Revised: March 4, 1996
FBV68LC040 Revision: 1

Technobox, Inc. (609)-778-5512
12-B The Ellipse
Suite 300
Mt. Laurel, New Jersey 08054

Bill Of Materials March 12, 1996 9:05:07 Page 1

Item	Quantity	Reference	Part
1	3	C100,C102,C105	10uFD 6032 1021
2	1	C101	100uFD,16V ELECE 1541
3	1	C103	2.2uFD35V 6032 1629
4	1	C104	1.0uFD CC3216 1007
5	52	C200,C201,C202,C203,C204, C205,C206,C207,C208,C209, C210,C213,C214,C215,C217, C218,C219,C220,C221,C223, C224,C225,C227,C228,C231, C232,C233,C234,C235,C236, C237,C238,C239,C240,C241, C242,C243,C244,C245,C246, C257,C258	0.1uFD RC1206 1004
6	3	C211,C212,C216	220pFD RC1206 1039
7	1	C222	100pFD RC1206 1012
8	1	C226	0.01uFD RC1206 1003
9	2	C229,C230	22pFD RC1206 1047
10	2	D1,D2	1N5819 MELF 1566
11	3	D3,D4,D5	YLED SOT23 1604
12	4	FD1,FD2,FD3,FD4	FADUC60 FADUC60 1619
13	1	JPC1	PC104A PC104A 1721
14	1	JPC2	PC104B PC104B 1722
15	1	JP1	DISKPWR DISKPWR 1613
16	1	JP2	HEADER 12 HDR12X1 1611
17	2	JP3,JP12	HEADER 10X2 HDR10X2N 1347
18	3	JP4,JP5,JP9	MTG1394 1394HTHR 1716
19	3	JP6,JP7,JP8	CONN1394 1394HSMT 1708
20	1	JP10	HEADER 7X2M HDR7X2ME 1557
21	1	JP11	HEADER 5X2M HDR5X2ME 1498

22	2	J1,J2	RCA JACK RCAJACK 1715
23	1	L1	47uHY PM54 1547
24	4	P1,P2,P3,P4	TESTPT 1HOLE 1244
25	1	P5	CONNECTOR DB9 DB9FRAS 1724
26	1	Q1	LM317H T220BCER 1714
27	1	R100	330x8 SOIC16 1464
28	3	R200,R204,R205	220OHM RC1206 1038
29	1	R201	6.36K RC1206 1083
30	1	R202	5.1Kx8 SOIC16 1266
31	2	R203,R211	RP2.2K SOIC16 1034
32	1	R206	560OHM RC1206 1081
33	2	R207,R213	10K RC1206 1015
34	2	R208,R212	56x8 SOIC16 1267
35	1	R209	1M RC1206 1006
36	1	R210	96OHM RC1206 1129
37	3	R214,R215,R216	1K RC1206 1029
38	1	R217	330K RC1206 1061
39	1	R218	82.5_OHM RC1206 1626
40	1	R219	243_OHM RC1206 1627
41	1	S1	SW-SPST-TINYPB SWSPSTBP 1230
42	4	TL1,TL2,TL3,TL4	TOOL125 TOOL125P 1630
43	1	U100	33.3333MHz 8DIPOSC 1067
44	1	U101	M445STRT PQFP100 1535
45	1	U102	TLC32046 PLCC28 1536
46	6	U103,U106,U112,U201,U202, U203	74FCT2374 QSOP20 1719
47	1	U104	TMS320C52-PQFP PQFP100 1534
48	4	U105,U107,U111,U114	MT5C2568 SOJ28 1209
49	1	U108	TSB11C01 SSOP56 1254
50	6	U109,U204,U206,U207,U209, U213	74FCT2245 QSOP20 1718
51	1	U110	TSB12C01A TQFP100 1550
52	1	U113	14.31818MHz 8DIPOSC 1717
53	1	U200	MAX232 SOIC16 1197
54	1	U205	M210PRIO TQFP44M 1532
55	1	U208	4.7Kx15 SOIC16 1723
56	1	U210	74FCT2823 QSOP24 1720
57	1	U211	AM29F100B-PSOP PSOP44 1607
58	1	U212	MAX705 SOIC8 1304
59	1	U214	MAX764CSA SOIC8 1549
60	1	Y1	24.576MHz SMTXTL 1564

Printed Circuit Specification - "GPLynx" - Rev 1 (9/20/97)

1. Board outline: one rectangular board, 3.775" x 3.550"
2. Material: FR4, Tg 140 to 160 degrees Centigrade
3. 6-Layer board (4 signal, 2 planes) in order: S/P/S/S/P/S
4. Drill file: Excellon format.
5. Finished plated hole size tolerance: +/- 0.003". Non-plated: +/-0.002"
6. Copper weight: per vendor's standard 6-layer board process.
7. Overall board thickness: 0.062" nom. Layer thickness as specified by Technobox.
8. Soldermask: ***Via holes are to be plugged with solder mask per vendor's recommended process.*** Then, LPI mask on primary and secondary sides is to be applied. ***Exposure of any copper at via pads on either side of the board is cause for rejection.*** A separate 'via-only' gerber file ("viaonly.lgx") is supplied to locate via holes for this purpose.
9. Soldermask: both sides, ***to cover via pads. Any copper exposure of Via pad will be cause for rejection.*** Maximum elevation of solder mask over via holes is 4 mils.
10. Silkscreen: Two Sides
11. Copper Geometry: 7 mil line, 7 mil space
12. Via pad size: 27 mil dia. Via Hole Size: suggest 13 mil dia, but will adapt to vendor recommendation.
13. Electrical Test: YES, complete for both sides. Board requires Dual-Sided Electrical Test.
14. Technobox to provide all artwork in RS-274X format.
15. Solder Mask Artwork supplied same size as copper features. Vendor to enlarge these solder mask openings in accordance with Vendor's solder mask process.
16. ***Soldermask is to be maintained between all SMT pads.*** Lack of soldermask between any pins is cause for rejection. The minimum SMT pad-to-pad spacing is 10 mils. Minimum allowable mask between these pins is 4 mils.
17. Vendor to add manufacturing logo, date code. Any vendor added test coupons will be outside the perimeter of the board.
18. 125 mil dia tooling holes are located at four corners of this board. Also serves as standoff.
19. Drill sizes supplied are nominal FINISHED hole sizes, to be adjusted by vendor according to vendor's standard plating processes.
20. Board to be trimmed to line center. Trim line to be presented in a separate gerber file, registered by targets across all layers.
21. No gold.
22. Vendor may optionally remove outer surface pads on non-plated holes.

Layer/File stackup as follows:

Gerber File Name	Layer
GPLynxa.lgx	Silk Screen top side
GPLynxb.lgx	Solder Mask top side
GPLynxc.lgx	Copper signal top side
GPLynxd.lgx	GND plane
GPLynxe.lgx	Interior signal layer
GPLynxf.lgx	Interior signal layer
GPLynxg.lgx	VCC plane (split)
GPLynxh.lgx	Copper signal bottom side
GPLynxi.lgx	Solder mask bottom side
GPLynxj.lgx	Silk Screen Legend - bottom side
trimline.lgx	Trim line w/fabrication information
viaonly.lgx	Layer containing via pads for plugging via holes

Drill specifications follow (note plated vs non-plated holes):

TOOL	COUNT	SIZE	
1	1189	0.013	
2	224	0.038	
4	5	0.064	
5	7	0.125	(non-plated)
23	4	0.070	
26	6	0.098	
27	6	0.065	(non-plated)

END OF FILE

Board Name: GPLYNX
BBS File Name:GPLYNX.ZIP

Layer Count: Six (4 signal, 2 planes)

Barriers:

BARVIA: keep out vias
BARALL: Keep out everything, all layers
BARTOP: Keep out traces, COMP layer
BARBOT: Keep out traces, SOLDER layer

Pin Type definitions for Vias when using 27 mil dia via pad:

72	*	c27d13.ps	% Vias		27 dia.
73	*	g27d13.ps	% Ground thermal	27 dia.	
74	*	v27d13.ps	% Power Thermal (VCC, PV3, PV3L)		27 dia.

(Note VCC and PV3 and PV3L signals are routed to same 'split' power plane in this design and therefore use same pin time for this design only).

Layers:

COMP: Component side signal
GND: ground plane (Route using 'GND' net)
INT1: Internal signal, component side
INT2: Internal signal, solder side
VCC: power.... Route 'VCC,' 'PV3L,' and 'PV3' nets on this layer. I'll split plane manually.
SOLDER: Solder side signal

Special trace widths: 'VCP' and 'VCP3' route with 25 mil wide trace.

Routing rules:

7 mil line, 7 mil space
27 mil diag vias (Use pin types 72,73,74)