Using the DECchip 21041 with Boot ROM, Serial ROM, and External Register:

An Application Note

Order Number: EC–QJLGA–TE

This application note provides information necessary to implement connections between the DECchip 21041 Ethernet LAN controller and boot ROM, serial ROM, and external register. It also describes a connection of several chips sharing one serial ROM and the format of the serial ROM programming.

Revision/Update Information: This is a new document.

Digital Equipment Corporation
Maynard, Massachusetts

April 1995

This document was prepared using VAX DOCUMENT Version 2.1.

Contents

1 Overview

The information contained in this application note describes how to connect the DECchip 21041 Ethernet LAN controller (21041) to its boot ROM, serial ROM, and external register peripheral devices. Note that connection to either a boot ROM or to an external register is not a requirement for correct operation of the controller. You can use any combination of these connections.

The programming information supplied in this application note applies to Digital-supplied device drivers. Users may use other formats supported by their own device drivers.

For detailed technical product requirements, the product developer should refer to the *DECchip 21041 PCI Ethernet LAN Controller Data Sheet* and the *DECchip 21041 PCI Ethernet LAN Controller Hardware Reference Manual*.

2 Functional Overview

The 21041 allows connection to an upgradable boot ROM (flash or EPROM) of 64KB, 128KB, or 256KB. The boot ROM typically contains code that can be executed for device-specific initialization and, possibly, a system boot function.

The 21041 also supports connection to the serial ROM for read and write operations. The serial ROM contains the IEEE address and other optional system parameters. The interface to serial ROM is fully software driven.

Connection to a general-purpose external register can be done for read and write operations. This connection allows a general-purpose bidirectional port for various applications.

The 21041 provides a 12-pin interface for all the connections described in this application note. The access control to the different devices is done by software using CSR9 and CSR10.

Table 1 lists all the 12 interface signals and their function in each connection.

Table 1 Boot ROM, Serial ROM, and External Register Interface Pins

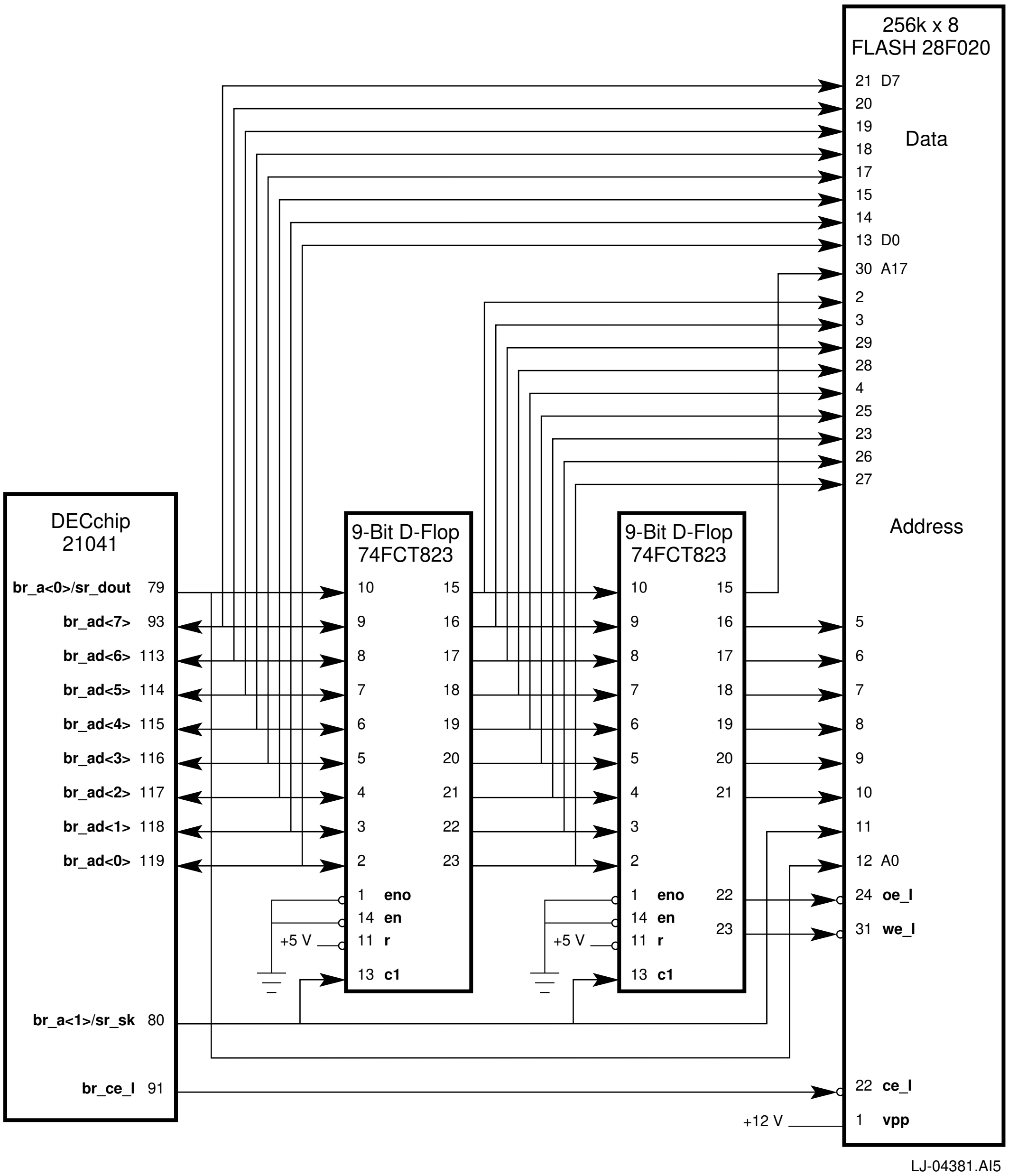
	Pin		Function	
Signal	**Number**	**Boot ROM**	**Serial ROM**	**External Register**
br_ad<6:0>	113:119	Address and data lines, **we_l**, **oe_l**	Not used	Data lines
br_ad<7>/sr_din	93	Address and data line	Serial ROM data in	Data line bit 7
br_a<1>/sr_sk	80	Address bit 1, latch control for external latches	Serial ROM clock	Not used
br_a<0>/sr_dout	79	Address bits 16, 17, and 0	Serial ROM data out	Read and write control
mode_select/br_ce_l	91	Chip enable control	Not used	Chip enable or read and write control
sr_cs	81	Not used	Chip select	Not used

3 Connection to Boot ROM

Figure 1 shows a connection of a 256KB flash boot ROM. The required components for this configuration are:

- Two 9-bit high edge-triggered latches (74FCT823)
- Flash ROM chip (28F020)

Figure 1 Boot ROM (256KB) Connection



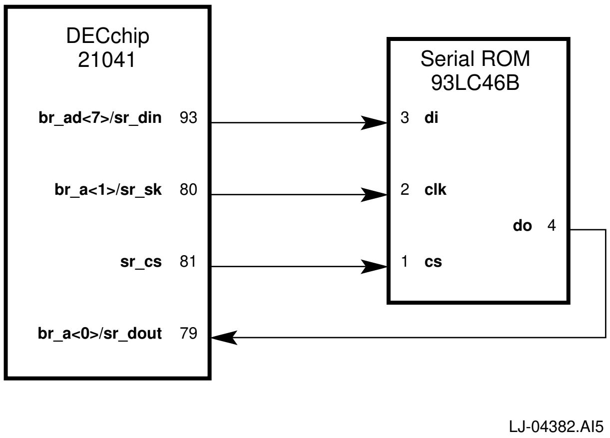
4 Connection to Serial ROM

The following sections describe connnections to the serial ROM.

4.1 Single DECchip 21041 Connection

Figure 2 shows a connection between a single DECchip 21041 and a MicroWire 1024-bit serial EPROM. No additional components are needed for this connection.

Figure 2 Serial ROM (1024-Bit) Connection



4.2 Multiple DECchip 21041 Connection

It is possible that one serial ROM device can be shared by multiple 21041 devices. For support, the serial ROM should contain specific information for each one of the chips. Section 6, Serial ROM Programming, provides the required details.

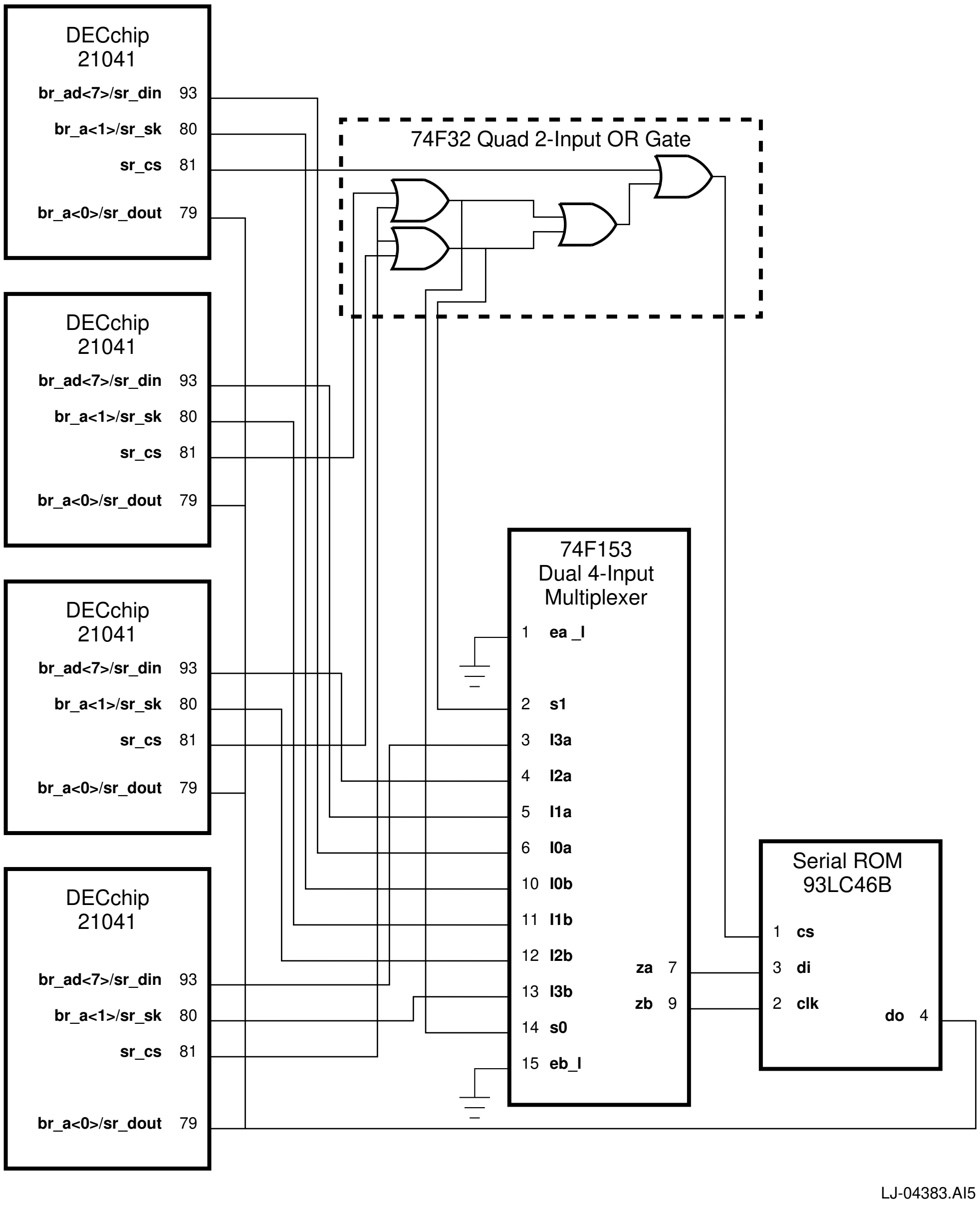
Figure 3 shows a connection of four 21041 chips sharing a single MicroWire 1024-bit serial EEPROM. The required components for this configuration are:

- Dual 4-input multiplexer chip (74F153)
- Quad 2-input OR gate chip (74F32)
- Serial ROM chip

This configuration assumes that:

- One 21041 will not try to access its boot ROM (or external register) while another 21041 has serial ROM access.
- Two 21041s will not have simultaneous access to the serial ROM.

Figure 3 Four Chips Sharing One Serial ROM



5 External Register Connection

This section describes two configuration types for using the general-purpose 8-bit external register.

- A minimum configuration without boot ROM and using the external register port in one direction only.
- A maximum configuration with boot ROM, using the external register as a bidirectional port.

5.1 Configuration of External Register Without Boot ROM

This configuration assumes that boot ROM is not used and the general-purpose external register is used for read-only or write-only operations.

Figure 4 shows a minimum type of configuration that uses the external register for write operations only.

Figure 4 External Register Connection—Write Only (No Boot ROM)

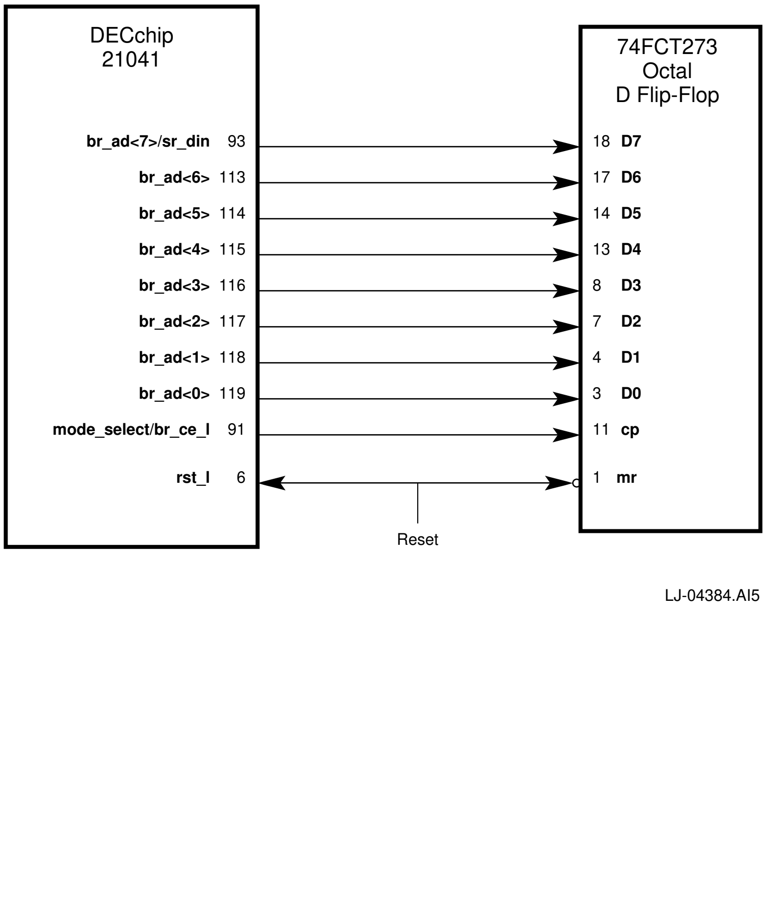
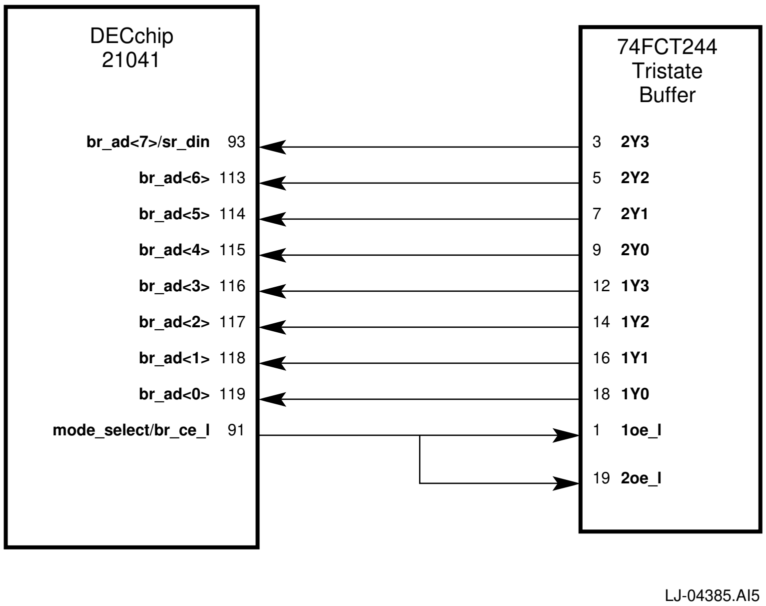


Figure 5 shows a configuration that uses the external register for read operations only. Data read by the 21041 should be driven constantly on the 74FCT244 inputs.

Figure 5 External Register Connection—Read Only (No Boot ROM)



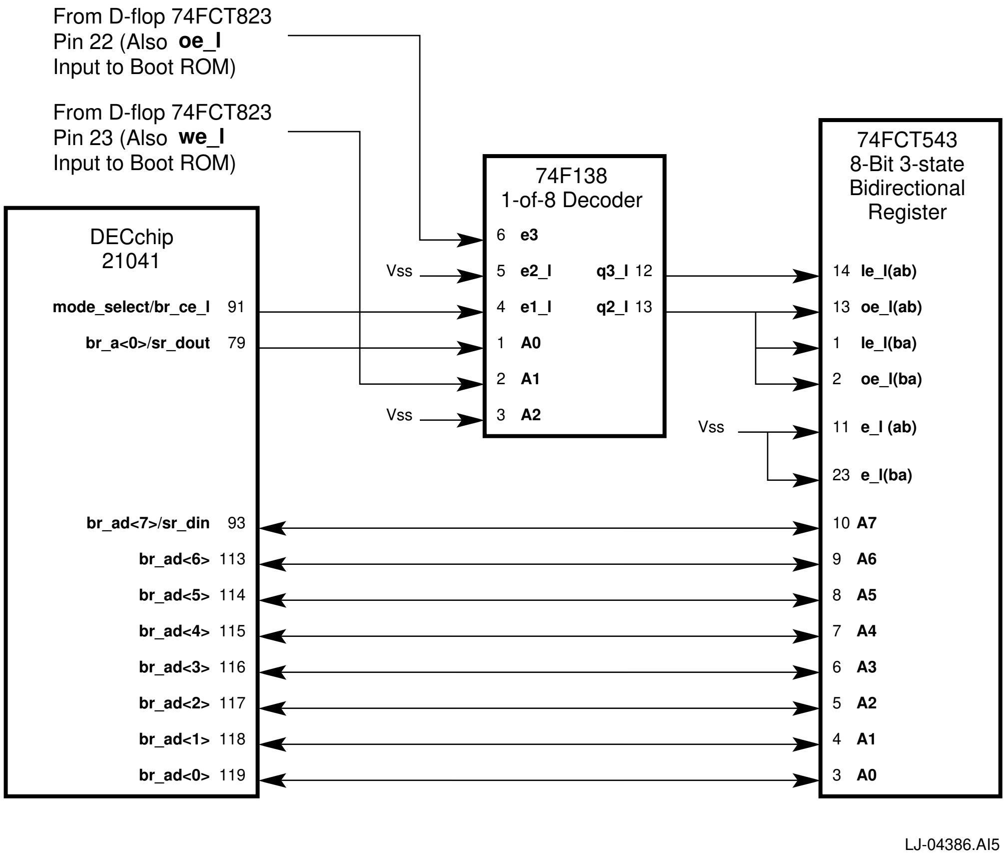
5.2 Configuration of External Register with Boot ROM

This connection assumes that both the external register and the boot ROM are used by the 21041. This connection also allows read and write accesses to the external register, making it a bidirectional general-purpose port. Note that Figure 1 shows the boot ROM connection.

Figure 6 describes the connection of the external register used for read and write operations with the boot ROM included on the adapter. The required components for this configuration are:

- 1-of-8 decoder
- Octal latched transceiver (3-state).

Figure 6 External Register Connection—Read and Write with Boot ROM



6 Serial ROM Programming

The definition for serial ROM programming that is described in this section supports multiple chips on a single board sharing a single serial ROM. It is applicable for Digital-supplied device drivers. Users may use other formats supported by their own device drivers.

Note

To optimize the ROM space usage, byte fields are used. Because the serial ROM supports only word accesses, Digital recommends that you first download the entire ROM into a memory shadow table.

Figure 7 shows the structure of the serial ROM and Table 2 describes the byte fields.

Figure 7 Serial ROM Structure

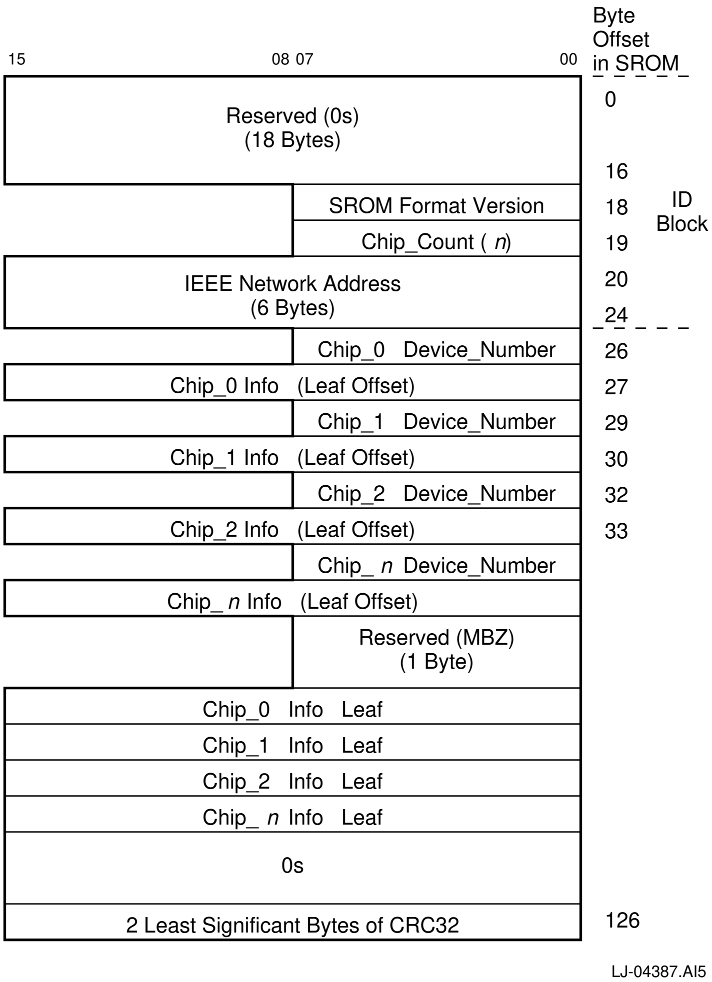


Table 2 Serial ROM Field Description

Field	Size (Bytes)	Definition
Reserved	18	Reserved, must be zero
SROM format version	1	SROM format version. Current version is 0x01
Chip_count (*n*)	1	Number of chips sharing this ROM. A single port board will have a value of 1 in this field.
IEEE network address	6	This is the IEEE address of the chip in a single chip board. In a multiple chip board, this is the base IEEE address. Every chip (0..*n*) adds its index (*n*) to this base IEEE address.
Chip_*n* Device_ Number	1	There is one such field per chip sharing the SROM. In a multi-chip board, this field contains the Device_Number value by which the *n*th chip's configuration space can be accessed on this board's secondary PCI bus. This value depends on the hardware routing of the board. The Device_Number is the *chip select* line routed from this chip to the PCI-to-PCI bridge chip on board. In a single chip board this field has no meaning and should be ignored by the driver.
Chip_*n* info	2	Byte offset (from beginning of SROM) where chip_*n* info block is located. There is one such field per chip sharing the SROM. **Note:** If multiple chips have identical information blocks, a single leaf can be shared and all leaf pointers can be set to point to it. This is correct only if the user cannot select between multiple media ports for each chip (see the following details). For example: A 4-TP port card can share one info block for all 4 chips.
Reserved	1	MBZ. Note that the location of this field depends on the number of chips supported by this card.
Chip_*n* info leaf	Chip dependent	Chip-specific information. See Figure 8 and Table 3 for details.
2 LSB of CRC32	2	Is calculated on all the words of the SROM from word[0] to the word before the CRC (word[SROM_ word_size –2]). The CRC word is derived by calculating the CRC32 of all the SROM until the last word (not including it) and taking the two least significant bytes of the result. That is, if the CRC is 4 bytes long with byte 0 being the least significant byte, then SROM_BYTE[BYTE_LEN –2] holds CRC<0> (least significant byte) and SROM_BYTE[BYTE_LEN –1] holds CRC<1>.

Figure 8 shows the info leaf format and Table 3 describes the byte fields.

Figure 8 Info Leaf Format

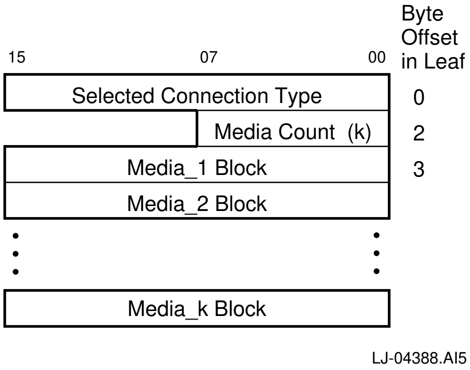


Table 3 Info Leaf Description

Field	Size (Bytes)	Meaning
Selected connection type	2	Usually, the connection type used by the chip is selected by the user in the drivers’ configuration files. However, this field has been provided to allow setup utilities that are unable to modify the configuration files and save this information in the SROM instead. The possible values are: 0x0000 - TP 0x0100 - TP with autonegotiation 0x0204 - TP full-duplex 0x0400 - TP without LinkPass test 0x0001 - BNC 0x0002 - AUI 0x0800 - AutoSense 0x0900 - AutoSense with autonegotiation If this field is not used, it must be set to 0xFFFF. Any other value is invalid and may cause unpredictable results.
Media count (k)	1	The number of media blocks present for this chip.
Media_k block	Media dependent	Describes one supported medium. There is one such field per supported medium. See details in Figure 9 and Table 4.

Figure 9 shows the media block format and Table 4 describes the byte fields.

Figure 9 Media Block Format

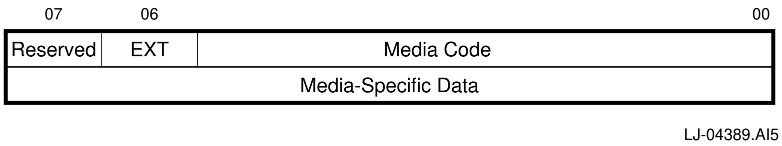


Table 4 Media Block Description

Field	Size	Meaning
Media code	6 bits	Indicates to the driver that this medium is supported by the chip. Possible values are: 00H - TP 01H - BNC 02H - AUI 04H - TP Full-Duplex
EXT	1 bit	When set to 1, indicates that the lower 16 bits of CSRs 13 through 15 (SIA registers) will be set via the SROM, and not by using the internal default values for this media type. The SIA setting values are given by the media-specific data field.
Reserved	1 bit	Reserved.
Media-specific data	6 bytes	Media-specific data. This field exists only when the EXT bit is set. This field provides the values of CSR13, CSR14, and CSR15 (Figure 10) to use instead of the driver internal defaults for this media type.

Figure 10 shows the media-specific data format.

Figure 10 Media-Specific Data Format

15 — 00
CSR13 <15:0>
CSR14 <15:0>
CSR15 <15:0>

LJ-04390.AI5

A

Serial ROM Access — Software Description

This appendix provides software code for both serial ROM read and write accesses. The routines that follow are written for the 39LC46B serial ROM device. Some modifications to the code may be required when supporting other devices.

```
/*
** Constants, variables, functions prototypes & definitions.
*/

#define SROM_93LC46B_LAST_ADDRESS   0x3F
#define SROM_93LC46B_LAST_ADDRESS_BIT      5
#define CSR9_READ   0x4000
#define CSR9_WRITE   0x2000
#define SEL_SROM   0x0800
#define DATA_1    0x0004
#define DATA_0    0x0000
#define CLK           0x0002
#define CS    0x0001

enum WIDTH {Byte,Word,Dword};

#define Byte    0
#define Word    1
#define Dword   2

typedef unsigned char   BYTE;
typedef unsigned short  WORD;
typedef unsigned long   DWORD;

#define FALSE   0
#define TRUE    1

void In32Bits(WORD port_number, enum WIDTH width, DWORD *ret_val);
void Out32Bits(WORD port_number, enum WIDTH width, DWORD value);
void Delay800nSec(void); /* Minimum time of clock high and clock low we apply
                         ** to SROM. The 93LC46B device requres a minimum of
                         ** 250nSec.
                         */

WORD CSR9;      /* Address of DC21140/DC21041 CSR9 in I/O space.
                ** This address is filled after locating DC21140/DC21041.
                */
```

```
/*
**   WriteCommandEWEN
**   ----------------
**
**   Operation:
**      Writes the Erase/Write Enable instruction to the serial ROM.
**      This enables writing to the serial ROM.
*/
void WriteCommandEWEN(void)
{
 WORD  i;

 /*
 ** Write the EWEN command to enable write/erase commands
 */
 Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       DATA_1);
 Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS | CLK | DATA_1);
 Delay800nSec();
 Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       DATA_1);
 Delay800nSec();

 Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       DATA_0);
 Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS | CLK | DATA_0);
 Delay800nSec();
 Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       DATA_0);
 Delay800nSec();

 Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       DATA_0);
 Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS | CLK | DATA_0);
 Delay800nSec();
 Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       DATA_0);
 Delay800nSec();

 for (i=0; i<5; i++)
     {
      Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       DATA_1);
      Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS | CLK | DATA_1);
      Delay800nSec();
      Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       DATA_1);
      Delay800nSec();
     }

 Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       DATA_1);
 Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS | CLK | DATA_1);
 Delay800nSec();
 Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM |            DATA_1);
 Delay800nSec();

 Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS );
 Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS | CLK);
 Delay800nSec();
 Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS );
 Delay800nSec();

 Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM);
 Delay800nSec();
}
```

```
/*   WriteCommandEWDS
**   ----------------
**
**   Operation:
**      Writes the Erase/Write Disable instruction to the serial ROM.
**      This disables writing to the serial ROM.
*/
void WriteCommandEWDS(void)
{
 WORD  i;

 /*
 ** Write the EWDS command to disable write/erase commands
 */
 Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       DATA_1);
 Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS | CLK | DATA_1);
 Delay800nSec();
 Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       DATA_1);
 Delay800nSec();

 for (i=0; i<7; i++)
     {
      Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       DATA_0);
      Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS | CLK | DATA_0);
      Delay800nSec();
      Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       DATA_0);
      Delay800nSec();
     }

 Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       DATA_0);
 Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS | CLK | DATA_0);
 Delay800nSec();
 Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM |            DATA_0);
 Delay800nSec();

 Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS );
 Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS | CLK);
 Delay800nSec();
 Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS );
 Delay800nSec();

 Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM);
 Delay800nSec();
}
```

```
/*   WriteSROM
**   ---------
**
**   Operation:
**      Writes the contents of a buffer to the SROM, word by word.
**
**   Input:
**      ROM_Address: Offset in SROM. Is auto incremented after write.
**       It is number of WORD to which Data is written.
**      Len:         Length in words.
**      Data:        Pointer to data buffer to write.
**
**   Output:
**      If an error occurs, error message is printed.
**
**   Return value:
**      FALSE if an error occurs.
*/
int WriteSROM(WORD *ROM_Address, WORD Len, WORD *Data)
{
 WORD  i, j;
 DWORD Dbit;
 DWORD Dout;
 WORD  ROM_WordAddress;
 WORD WordData;

 ROM_WordAddress = *ROM_Address;

 /*
 ** Make sure the ROM_Address is not too big for this ROM
 */
 if (ROM_WordAddress + Len - 1 > SROM_93LC46B_LAST_ADDRESS)
    {
     printf("Address or data length is too big for SROM\n");
     return(FALSE);
    }

 WriteCommandEWEN();

 /*
 ** Loop on all DATA words.
 */
 for (j=0; j<Len; j++)
     {
      /*
      ** Output the WRITE command to the SROM
      */
      Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       DATA_1);
      Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS | CLK | DATA_1);
      Delay800nSec();
      Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       DATA_1);
      Delay800nSec();

      Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       DATA_0);
      Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS | CLK | DATA_0);
      Delay800nSec();
      Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       DATA_0);
      Delay800nSec();

      Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       DATA_1);
      Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS | CLK | DATA_1);
      Delay800nSec();
      Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       DATA_1);
      Delay800nSec();
```

```
/*
** Output the WORD Address of the SROM
*/
for (i=0; i<=SROM_93LC46B_LAST_ADDRESS_BIT; i++)
    {
     Dbit = (DWORD)((ROM_WordAddress >> (SROM_93LC46B_LAST_ADDRESS_BIT-i)) & 1) << 2;
     Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       Dbit);
     Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS | CLK | Dbit);
     Delay800nSec();
     Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       Dbit);
     Delay800nSec();
    }

/*
** Output the WORD of data to the SROM
*/
WordData = *Data;
for (i=0; i<=15; i++)
    {
     Dbit = (DWORD)((WordData >> (15-i)) & 1) << 2;
     Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       Dbit);
     Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS | CLK | Dbit);
     Delay800nSec();
     Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       Dbit);
     Delay800nSec();
    }

/*
** Point at next user buffer address
*/
Data++;

/*
** Point at next SROM Address
*/
ROM_WordAddress++;

/*
** Negate the CS (chip select) to start the SROM write
*/
Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM);
Delay800nSec();

/*
** Set the CS to continue the SROM write
*/
Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS      );

/*
** Verify that the SROM is in BUSY state by reading the Dout, if Dout=0.
*/
In32Bits(CSR9, Dword, &Dout);
Dout = (Dout>>3) & 1;
if (Dout != 0)
   {
    printf("SROM did not become busy in write command\n");
    return(FALSE);
   }
```

```
      /*
      ** Wait for completion of WRITE command up to 10Msec.
      ** 10Msec = 11900 loops of 800nSec.
      */
      for (i=0; i<11900; i++)
          {
           Delay800nSec();
           In32Bits(CSR9, Dword, &Dout);
           Dout = (Dout>>3) & 1;
           if (Dout == 1)
              break;
          }

      if (Dout == 0)
         {
          printf("SROM did not end busy state in write command\n");
          return(FALSE);
         }

      /*
      ** Negate the CS to end the SROM command
      */
      Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM);
      Delay800nSec();
     }

 WriteCommandEWDS();

 /*
 ** Save the ROM byte address for next use.
 */
 *ROM_Address = ROM_WordAddress;

 return 1;
}

/*   ReadSROM
**   --------
**
**   Operation:
**      Reads the contents of the SROM (starting at a given offset),
**      into a given buffer.
**
**   Input:
**      ROM_Address: Offset in SROM. Is auto incremented after write.
**       It is number of WORD to which Data is written.
**      Len:         Length in words.
**      Data:        Pointer to data to buffer to read into.
**
**   Output:
**      If an error occurs, error message is printed.
**
**   Return value:
**      FALSE if an error occurs.
*/
int ReadSROM(WORD *ROM_Address,WORD Len,WORD *Data)
{

 WORD  i, j;
 DWORD Dbit;
 DWORD Dout;
 WORD  ROM_WordAddress;
 WORD  WordData;

 ROM_WordAddress = *ROM_Address;
```

```
/*
** Make sure the ROM_Address is not too big for this ROM
*/
if (ROM_WordAddress + Len -1 > SROM_93LC46B_LAST_ADDRESS)
   {
    printf("Address or data length is too big for SROM\n");
    return(FALSE);
   }

/*
** Loop on all DATA words.
*/
for (j=0; j<Len; j++)
    {
     /*
     ** Output the READ command to the SROM
     */
     Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       DATA_1);
     Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS | CLK | DATA_1);
     Delay800nSec();
     Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       DATA_1);
     Delay800nSec();

     Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       DATA_1);
     Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS | CLK | DATA_1);
     Delay800nSec();
     Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       DATA_1);
     Delay800nSec();

     Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       DATA_0);
     Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS | CLK | DATA_0);
     Delay800nSec();
     Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       DATA_0);
     Delay800nSec();

     /*
     ** Output the WORD Address of the SROM
     */
     for (i=0; i<=SROM_93LC46B_LAST_ADDRESS_BIT; i++)
         {
          Dbit = (DWORD)((ROM_WordAddress >> (SROM_93LC46B_LAST_ADDRESS_BIT-i)) & 1) << 2;
          Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       Dbit);
          Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS | CLK | Dbit);
          Delay800nSec();
          Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS |       Dbit);
          Delay800nSec();
         }

     /*
     ** Verify that the SROM output data became now 0.
     */
     In32Bits(CSR9, Dword, &Dout);
     Dout = (Dout>>3) & 1;
     if (Dout != 0)
        {
         printf("SROM did not become busy in read command\n");
         return(FALSE);
        }
```

```
     /*
     ** Input the WORD of data from the SROM
     */
     for (i=0; i<=15; i++)
         {
          Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS | CLK);
          Delay800nSec();
          In32Bits(CSR9, Dword, &Dout);
          WordData |= ((Dout>>3) & 1) << (15-i);
          Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM | CS      );
          Delay800nSec();
         }

     /*
     ** Put our read data in user buffer
     */
     *Data = WordData;

     /*
     ** Point at next user buffer address
     */
     Data++;

     /*
     ** Point at next SROM Address
     */
     ROM_WordAddress++;

     /*
     ** Negate the CS (chip select) to end the SROM command
     */
     Out32Bits(CSR9, Dword, CSR9_WRITE | SEL_SROM);
     Delay800nSec();
    }

 /*
 ** Save the ROM byte address for next use.
 */
 *ROM_Address = ROM_WordAddress;

 return(TRUE);
}
```

B

Serial ROM CRC Calculation

This appendix provides the routine for calculating the serial ROM CRC value.

```
unsigned const DATA_LEN 126

struct
{
   unsigned char srom_data[DATA_LEN];
   unsigned short srom_crc;
} Srom;

main()
{

   FillSromData(&Srom.srom_data);

   Srom.srom_crc = CalculateSromCrc(&Srom.srom_data);

}

unsigned short CalculateSromCrc(unsigned char *srom_data)
{
   unsigned crc = 0xFFFFFFFF;
   unsigned const POLY 0x04C11DB6

   unsigned char current_byte;
   unsigned index;
   int bit;
   unsigned msb;

   for (index = 0; index < DATA_LEN; index++)
   {
      current_byte = srom_data[index];

      for (bit = 0; bit < 8; bit++)
      {
         msb = crc >> 31;
         crc <<= 1;

         if (msb ^ (current_byte & 1))
         {
            crc ^= POLY;
            crc |= 0x00000001;
         }

         current_byte >>= 1;
      }
   }

   crc = FlipLongword(crc);

   return ((crc ^ 0xFFFFFFFF) & 0xFFFF);
}
```

```
unsigned long FlipLongword(unsigned long lw)
{
   int i;
   unsigned long result = 0;
   unsigned bit;

   for (i = 0; i < 32; i++)
   {
      result <<=1;
      bit = lw & 1;
      lw >>= 1;
      result += bit;
   }

   return (result);
}

void FillSromData(unsigned char *srom_data)
{

   /* fill the 126 bytes of data as is appropriate */
}
```

Technical Support and Ordering Information

Technical Support

If you need technical support or help deciding which literature best meets your needs, call the Digital Semiconductor Information Line:

United States and Canada	**1–800–332–2717**
TTY (United States only)	**1–800–332–2515**
Outside North America	**+1–508–568–6868**

Ordering Digital Semiconductor Products

To order the DECchip 21041 PCI Ethernet LAN Controller and Evaluation Board, contact your local distributor.

You can order the following semiconductor products from Digital:

Product	Order Number
DECchip 21041 PCI Ethernet LAN Controller	21041–AA
DECchip 21041 Evaluation Board Kit	21A41–01
DECchip 21040 Ethernet LAN Controller for PCI	21040–AA
DECchip 21040 Evaluation Board Kit	21A40–01
DECchip 21140 PCI Fast Ethernet LAN Controller	21140–AA
DECchip 21140 Evaluation Board Kit	21A40–03

Ordering Associated Literature

The following table lists some of the available Digital Semiconductor literature. For a complete list, contact the Digital Semiconductor Information Line.

Title	Order Number
DECchip 21041 PCI Ethernet LAN Controller Product Brief	EC–QAWVA–TE
DECchip 21041 PCI Ethernet LAN Controller Data Sheet	EC–QAWWA–TE
DECchip 21041 PCI Ethernet LAN Controller Hardware Reference Manual	EC–QAWXA–TE

Ordering Third-Party Literature

You can order the following third-party literature directly from the vendor:

Title	Vendor
PCI Local Bus Specification, Revision 2.0	PCI Special Interest Group N/S HH3–15A 5200 N.E. Elam Young Pkwy Hillsboro, OR 97124–6497 1–503–696–2000
Institute of Electrical and Electronics Engineers (IEEE) 802.3	IEEE Service Center 445 Hoes Lane P.O. Box 1331 Piscataway, NJ 08855–1331 1–800–678–IEEE (U.S. and Canada) 908–562–3805 (Outside U.S. and Canada)