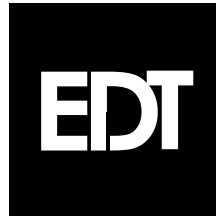


S11W

SBus to DR11 Parallel Interface

USER'S GUIDE

008-00062-11



The information in this document is subject to change without notice and does not represent a commitment on the part of Engineering Design Team, Inc. The software described in this document is furnished under a license agreement or nondisclosure agreement. The software may be used or copied only in accordance with the terms of the agreement.

No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, without the express written agreement of Engineering Design Team, Inc.

© Copyright Engineering Design Team, Inc. 1992–1997. All rights reserved.

Refer questions or problems with this manual or the hardware or software described herein to:

Engineering Design Team, Inc.
1100 NW Compton Drive, Suite 306
Beaverton, Oregon 97006

Phone: (503) 690-1234
Fax: (503) 690-1243
E-mail: info@edt.com
Web: www.edt.com

Sun, SunOS, SBus, SPARC, and SPARCstation are trademarks of Sun Microsystems, Incorporated.

UNIX and OPEN LOOK are registered trademarks of UNIX System Laboratories, Inc.

X Window System is a product of the Massachusetts Institute of Technology.

EDT and Engineering Design Team are trademarks of Engineering Design Team, Inc.

Contents

Overview	1
Installation.....	2
Installing the Hardware.....	2
Installing the Software	3
Using SunOS Version 4.1	3
Using System V Release 4 (Solaris 2.0)	4
Installing the Sample Programs	4
Included Files	5
Input and Output	6
Elements of S11W Applications	6
DMA Library Routines	7
Testing.....	15
Error Conditions	16
Hardware.....	17
SBus Interface	17
FIFO	17
Device Interface	17
Logic Levels	17
FCode Capability	18
Signals	19
Connector Pinout	19
Synchronous Control Signals.....	21
Handshake Signals	22
Asynchronous Control Signals.....	23
Unimplemented DR11W Signals.....	23
Timing	24
Registers.....	25
SBus Addresses	25
Command Register	26
Status Register	27
Configuration Register.....	28
DMA Word Count Register	29
DMA Address Register	29
DMA End Address Register.....	29
Data Register	29
Specifications	30
SBus Compliance	30
Device Data Transfer.....	30
Software	30
Power.....	30
Environmental.....	30
Physical	30
References.....	31
Appendix A ioctl() Parameters.....	32

Figures

Block Diagram of the S11W Interface.....	17
The Drivers	18
Timing Diagram	24
SBus Addresses.....	25

Tables

General DMA Library Routines	7
Loopback Diagnostic Tests	15
Error Codes and Conditions	16
Connector Pinout.....	20
Synchronous Control Signals.....	21
Handshake Signals	22
Asynchronous Control Signals.....	23
The Command Register	26
The Status Register	27
The Configuration Register.....	28
The DMA Address Register.....	29
The DMA Address Register.....	29
The DMA End Address Register	29
The Data Register.....	29

Overview

The S11W is a single-slot, 16-bit parallel input/output interface for SBus-based computer systems. The external interface conforms to the DR11W standard of Digital Equipment Corporation. The S11W features 512 bytes of FIFO storage in each direction and can support continuous data rates of up to 8 MB per second. The board also includes diagnostic capability.

The S11W fully supports Direct Virtual Memory Access (DVMA) and the requirements of the SunOS operating system. It includes a SunOS-compatible software driver, enabling you to set up multiple buffers in application memory.

The S11W supports a high-speed block mode as well as all standard DR11W protocols. In this block mode, the S11W transfers data from the SBus memory with burst transfers, using FIFO memory on the S11W to buffer the transfer. This capability is useful if your application requires high data transfer rates and does not change direction in mid-block.

The S11W also allows link mode applications, in which one S11W communicates with another or with a DR11W.

The S11W supports SBus FCodes for device identification and diagnostics. Engineering Design Team can supply customized FCode that supports booting from user devices.

Test the S11W by installing an optional loopback connector and executing the S11W diagnostics. A diagnostic program is included with the standard S11W software. The loopback connector kit is available separately. Contact Engineering Design Team or your distributor for further information.

A high-density connector attaches the S11W device to the device cable supplied with the interface. The other end of the cable terminates in two standard DR11W 40-pin connectors. High-density connectors terminate both ends of a S11W-to-S11W interface.

This document describes how to install the S11W interface and write applications for it. It is divided into the following sections:

Installation	describes how to install the S11W module and its related software.
Testing	describes how to test the S11W module.
Input and Output	describes the <i>ioctl</i> parameters, asynchronous input and output, and error conditions.
Hardware	explains the interfaces to the SBus and the device, the FIFOs, logic levels, and FCode for the PROMs.
Registers	describes the S11W hardware registers.
Signals	provides a connector pin-out diagram and describes the S11W signals and timing. It also discusses using the signals for data input and output.
Specifications	lists the specifications.
References	refers to other documents that may prove useful to you when writing applications for the S11W.

Installation

Installing the S11W interface is a two-step process. First you must physically install the board inside the host computer. Then you must install the software driver so that applications can access the S11W. Hardware installation is described in the following section. Software installation is described in the section after.

Installing the Hardware

The S11W is a single-slot SBus board. To install it, refer to your SBus host computer documentation for complete information on installing an SBus board. For example, for the following SPARCstation models this information can be found in these locations:

SPARCstation Model	Location in <i>SPARCstation Installation Guide</i>
1	Appendix A: Installing Boards, Cards, and Modules
1+	Appendix A: Installing SBus Boards and SIMMs
IPX	Appendix C: Installing SBus Cards
10	Appendix B: Opening and Closing the System Unit and <i>Installing SBus Cards in Desktop SPARCstations</i> Sun part no. 800-6635-11.

Use the following procedure to install the S11W:

CAUTION

Both the S11W and your SBus host computer contain static-sensitive components. Install the S11W at a static-free work area. If a static-free work area is not available, take the following precautions to reduce the risk of component damage:

1. Remove from the immediate area all materials that can generate or hold a static charge.
 2. Discharge yourself by touching both hands to a metal portion of the host computer's chassis before you open the host computer or open the S11W static-shielded bag.
-
-

1. Unpack the S11W from the shipping packaging. Do not remove the S11W from the static shielding bag until all you remove all other packaging materials from the area and established a static-free work area
2. Install the S11W in the SBus host, following the directions provided with the SBus host. The S11W can be installed in any DMA slot.
3. The S11W is delivered with the jumper installed. Remove this jumper if your device requires the BUSY polarity to be negative true from initial host power-up. To do so, lift the black case away from both pins of the jumper, and replace it over just one pin.

To remove the S11W, reverse the installation procedure.

The S11W connects to your device with up to 50 feet of cable. The standard EDT device cable terminates at the S11W with a high-density connector, and at the other end in two standard .100 x .100 40-pin connectors.

EDT provides several cable options for connecting the S11W to an external device. We recommend that you use one of these specially designed cables to achieve the best possible performance. Contact your

distributor or EDT for information and specifications for these cables. Each cable is shipped with instructions for connecting the S11W to an external device.

Installing the Software

The S11W can run on a Sun workstation using either SunOS Version 4.1.3 or 4.1.4, or Solaris 2.0 (System V Version 4, or SVR 4). The installation procedures differ. Both are given below.

Using SunOS Version 4.1

If you are using SunOS Version 4.1.3 or 4.1.4, use the following procedure to install the S11W driver:

1. Become root or superuser.
2. Change to the directory in which you wish to install the S11W driver.
3. Place the diskette that came with the S11W into the diskette drive.
4. The S11W driver and related files are included on a diskette in *tar* format. To copy them to your hard disk, enter:

```
tar xvf /dev/rfd0
```

The *tar* program extracts a number of files. (The list of files distributed is provided in the section entitled **Included Files**.) The S11W diskette contains versions of the S11W driver for a variety of Sun platforms and versions of the Sun operating system. The installation program installs the correct driver based on the host platform and operating system version.

5. To install the driver, enter:

```
make install
```

The makefile provided installs and loads the S11W driver.

6. During the installation, the following question appears on the display:

```
Automatically load the S11W driver during each reboot? [y|n] (y):
```

Entering *y* (or simply typing <Return>) causes the S11W driver to be loaded whenever you reboot your host computer. If you respond with *n*, you must manually reload the driver after rebooting. To do so, enter:

```
make load
```

7. During the installation, the following question appears on the display:

```
How many S11W devices do you want? (1):
```

You can install as many S11W boards in your system as you have DMA SBus slots available. Enter the number corresponding to the number of S11W boards you have installed in your system. If you simply type <Return>, one driver is installed.

NOTE: If you anticipate installing more than one S11W board into your system, install as many S11W drivers as you will ultimately require. The extra drivers will do no harm and will be there when you need them, saving you a step.

8. If the S11W module has not been installed inside the host computer, or has been installed incorrectly, the following message appears on the display:

```
Can't load this module
```

If you see this message, go back to the section entitled **Installing the Hardware** and reinstall the board.

To unload the S11W driver:

1. Change to the directory in which you placed the S11W files, if necessary.
2. Become root or superuser.
3. Enter:

```
make unload
```

Using System V Release 4 (Solaris 2.0)

If you are using Sun System V Release 4 (Solaris 2.0), use the following procedure to install the S11W driver:

1. Become root or superuser.
2. Place the diskette that came with the S11W into the diskette drive.
3. Invoke the File Manager by entering:

```
filemgr
```

or switch to the File Manager window if it is already running.

4. In the File Manager, pull down the File menu and execute the menu item Check For Floppy.
5. When the Unlabeled Floppy window appears, select Cancel.
6. At the shell prompt, enter:

```
pkgadd -d /vol/dev/aliases/floppy0 EDTS11W
```

To remove the S11W driver:

1. Become root or superuser.
2. Enter:

```
pkgrm EDTS11W
```

For further details, consult your Solaris 2.0 documentation, or call the Engineering Design Team.

Installing the Sample Programs

To install any of the example programs, enter the command:

```
make file
```

where *file* is the name of the example program you wish to install.

To build and install all the example programs, simply enter the command:

```
make
```

All example programs display a message that explains their usage when you enter their names without parameters.

Included Files

The S11W driver release diskette contains the following files (see the *readme* file for a complete, up-to-date listing):

s11w.o.sun4c	The executable S11W driver for SunOS 4.1.3 on a Sun 4C architecture such as a SPARCStation 1, 1+, 2, or IPC.
s11w.o.sun4m	The executable S11W driver for SunOS 4.1.3 on a Sun 4M architecture such as a SPARCStation 5, 10, 20, LX, Classic, or an Ultra 1 or 2.
s11w	The executable S11W driver for Solaris Version 2.x.
s11w.h	The S11W driver header file, defining <i>ioctls</i> and registers.
s11w.INSTALL	The installation script used by the S11W makefile.
makefile	The makefile for installing, loading, and unloading the S11W driver, and making example programs. Used with the SunOS <i>make</i> command to automatically install the driver or compile the example programs.
README	An ASCII file containing last-minute information about the S11W software.
setdebug.c	This file sets the debug levels for the S11W. Debug level number: <ol style="list-style-type: none"> enables <i>modstat(8)</i> in verbose mode (<i>modinfo</i> for Solaris V. 2.x) provides additional debug information traces the start and end of routines <p>Interrupt debug level number 1 shows interrupts and DMA setup.</p>
loopback.c	A set of software diagnostics for the S11W board and driver. Use this test with a loopback connector. See the Testing section of this document for more information.
speedtest.c	A simpler example program illustrating asynchronous I/O read requests to the S11W using ring-buffering and the driver handshaking mechanisms. <i>speedtest.c</i> needs an external data source such as a simple data clock.
s11wpio.c	An example program illustrating programmed I/O using <i>ioctls</i> .
s11wpiomem.c	An example program illustrating programmed I/O performed by mapping registers into application memory.
diag.c slave.c master.c	These programs show an example of communication between two S11W devices installed in the same system.
katodemo.sh	Execute this file for a demo of the KATO kernel analysis tool, which allows you to watch the S11W's performance, analyze throughput, and optimize data transfer. The KATO kernel analysis tool is available from Engineering Design Team, Inc.
KATO.README	An ASCII file containing last-minute information about the KATO demo software.
kato.demo	The executable file for the kernel analysis tool demo.
kato.frec	The record of the KATO test that is played back during the KATO demo.
s11w.c	The source for the S11W driver. This file is present only if you purchased the source from EDT.

Input and Output

The driver can perform two kinds of DMA transfers: continuous and noncontinuous. For noncontinuous transfers, the driver uses DMA system calls for *read()* and *write()*. Each *read()* and *write()* system call allocates kernel resources, during which time DMA transfers are interrupted.

To perform continuous transfers, use the ring buffers. The ring buffers area set of buffers that applications can access continuously, reading and writing as required. When the last buffer in the set has been accessed, the application then cycles back to the first buffer. See **s11w_configure_ring_buffers** for a complete description of the ring buffer parameters that you can configure.

Elements of S11W Applications

S11W applications for performing noncontinuous transfers typically include the following elements:

1. The preprocessor statement

```
#include "libs11w.h"
```

2. A call to `s11w_open()`, such as:

```
s11w_p = s11w_open(s11w_p) ;
```

3. A call to `s11w_read()` or `s11w_write()`, such as:

```
s11w_read(s11w_p, buf_ptr, 512) ;
```

or:

```
s11w_write(s11w_p, buf_ptr, 1024) ;
```

4. A call to `s11w_close()` to close the device before ending the program, such as:

```
s11w_close(s11w_p) ;
```

5. The `-ls11w` option to the compiler, to link the library file *libs11w.a* with your program

S11W applications for performing continuous transfers typically include the following elements:

1. The preprocessor statement

```
#include "s11w.h"
```

2. A call to `s11w_open()`, such as:

```
s11w_p = s11w_open(s11w_p) ;
```

3. A call to `s11w_configure_ring_buffers()` to set up the ring buffers as required, such as:

```
s11w_configure_ring_buffers(s11w_p, 1024, 4, NULL, EDT_READ) ;
```

4. A call to `s11w_start_buffers()` with an argument of 0 to initiate a continuous transfer, such as:

```
s11w_start_buffers(s11w_p, 0) ;
```

5. A call to `s11w_wait_for_buffer()` or `s11w_wait_for_next_buffer()` to , such as:

```
buf_ptr = s11w_wait_for_buffer(s11w_p, 4) ;
```

6. A call to `s11w_close()` to close the device before ending the program, as in:

```
s11w_close(s11w_p) ;
```

7. The `-ls11w` option to the compiler, to link the library file `libs11w.a` with your program

See the makefile and example programs provided for examples of compiling code using the library routines.

DMA Library Routines

The DMA library provides a set of consistent routines across many of the EDT products, with simple yet powerful ring-buffered DMA capabilities. **Table 1, DMA Library Routines** lists the general DMA library routines. In addition, if driver-specific library routines exist, they can be found in a table thereafter.

The sections that follow describe the DMA library routines in alphabetical order.

Routine	Description
<code>s11w_open</code>	Opens the S11W for application access.
<code>s11w_close</code>	Terminates access to the S11W and releases resources.
<code>s11w_read</code>	Single, application-level buffer read from the S11W.
<code>s11w_write</code>	Single, application-level buffer write to the S11W.
<code>s11w_set_defaults</code>	Restores the driver and hardware to factory-specified default state.
<code>s11w_configure_ring_buffers</code>	Configures the ring buffers.
<code>s11w_buffer_addresses</code>	Returns addresses of ring buffers.
<code>s11w_wait_for_buffer</code>	Blocks until specified buffers have completed.
<code>s11w_wait_for_next_buffer</code>	Blocks until the next buffer completes.
<code>s11w_check_next_buffer</code>	Checks whether next buffer is complete.
<code>s11w_start_buffers</code>	Begins transfer from or to specified number of buffers.
<code>s11w_cancel</code>	Shuts down the device as soon as possible, optionally resets it.
<code>s11w_cancel_current</code>	Cancels the current DMA, moves pointers to the next.
<code>s11w_stop_buffers</code>	Stops the interface after all buffers have completed.
<code>s11w_done_count</code>	Return absolute (cumulative) number of completed buffers.
<code>foi_parity_error</code>	Checks for parity error since last call.

Table 1. General DMA Library Routines

`s11w_buffer_addresses`

Description

Returns an array containing the addresses of the buffers.

Syntax

```
void **s11w_buffers(S11wDev *s11w_p);
```

Arguments

`s11w_p` S11W device handle returned from `s11w_open`.

Return

Address of an array of pointers to the ring buffers allocated by the driver or the library. The array of buffer pointers is allocated by the library. Null on error.

s11w_cancel

Description

Stops any transfers currently in progress, resets the ring buffer pointers to restart on the current buffer.

Syntax

```
int s11w_cancel(S11wDev *s11w_p);
```

Arguments

s11w_p S11W device handle returned from *s11w_open*.

Return

0 on success, -1 on failure. Sets *errno* on failure.

s11w_cancel_current

Description

Stops the current transfers, resets the ring buffer pointers to the next buffer.

Syntax

```
int s11w_cancel(S11wDev *s11w_p);
```

Arguments

s11w_p S11W device handle returned from *s11w_open*.

Return

0 on success, -1 on failure. Sets *errno* on failure.

s11w_check_next_buffer

Description

Checks whether the next buffer is complete.

Syntax

```
int s11w_check_next_buffer(S11wDev *s11w_p);
```

Arguments

s11w_p S11W device handle returned from *s11w_open*.

Return

0 on success, -1 on failure. Sets *errno* on failure.

s11w_close

Description

Closes the device associated with the device handle and frees the handle.

Syntax

```
int s11w_close(S11wDev *s11w_p);
```

Arguments

s11w_p S11W device handle returned from *s11w_open*.

Return

0 on success, -1 on failure. Sets *errno* on failure.

s11w_configure_ring_buffers

Description

Configures the SBus to DR11 Parallel Interface ring buffers. Any previous configuration is replaced, and previously allocated buffers are released.

Buffers can be allocated and maintained within the SBus to DR11 Parallel Interface library or within the user application itself.

Syntax

```
int s11w_configure_ring_buffers(S11wDev *s11w_p, int bufsize, int nbufs,
                               void *bufarray[], int data_output);
```

Arguments

s11w_p S11W device handle returned from *s11w_open*

bufsize size of each buffer. For optimal efficiency, allocate a value approximating throughput divided by 20: that is, if transfer occurs at 20 MB per second, allocate 1 MB per buffer. Buffers significantly larger or smaller can overuse memory or lock the system up in processing interrupts.

nbufs number of buffers. Must be 1 or greater. Four is recommended.

bufarray array of pointers to individual buffers if the buffers are allocated by the application. Must be NULL if in library, or have *nbufs* elements.

library Must be NULL.

user This array must be filled with the addresses of the buffers allocated by the application for the library to use.

data_direction Indicates whether this connection is to be used for input or output. Only one direction is possible per device or subdevice:

EDT_READ = 0

EDT_WRITE = 1

Return

0 on success; -1 on error. If all buffers cannot be allocated, none are allocated and an error is returned.

s11w_done

Description

Returns the cumulative count of completed buffer transfers.

Syntax

```
int s11w_done_count(S11wDev *s11w_p);
```

Arguments

s11w_p S11W device handle returned from *s11w_open*.

Return

The number of completed buffer transfers. Completed buffers are numbered consecutively starting with 0 when the S11W is opened. (Thus, the allocated buffer number is the transferred buffer number modulo the number of allocated buffers.)

s11w_open

Description

Opens the specified S11W and sets up the device handle.

Syntax

```
S11wDev *s11w_open(int unit);
```

Arguments

unit specifies the device unit number

Return

A handle of type (S11wDev *), or NULL if error. If an error occurs, check the *errno* global variable for the error number.

s11w_read

Description

Performs a read on the S11W. The UNIX 2 GB file offset bug is avoided during large amounts of input or output, that is, reading past 2^{31} does not fail. This call is not multibuffering, and no transfer is active when it completes.

Syntax

```
int s11w_read(S11wDev *s11w_p, void *buf, int size);
```

Arguments

s11w_p S11W device handle returned from *s11w_open*

buf address of buffer to read into

size size of read in bytes

Return

The return value from *read*; *errno* is set by *read* on error.

s11w_set_defaults

Description

Resets the S11W configuration to its default state:

- All three FNCT bits, the DIRS 1 and DIRS 0 bits, and all bits of the Configuration register are set to 0.
- The BLKM, BCLR, and DISINT bits are set to 1.
- Ring buffer mode is disabled and driver- or library-allocated ring buffers are released.

Syntax

```
int s11w_set_defaults(S11wDev *s11w_p);
```

Arguments

s11w_p S11W device handle returned from *s11w_open*

Return

Return value from *ioctl* call to driver: 0 on success; -1 on error. If an error occurs, check the *errno* global variable for the error number.

s11w_start_buffers

Description

Releases the specified number of buffers to the driver for transfer.

Syntax

```
int s11w_start_buffers(S11wDev *s11w_p, int bufnum);
```

Arguments

s11w_p S11W device handle returned from *s11w_open*

bufnum Number of buffers to release to the driver for transfer. An argument of 0 causes the driver to perform continuous transfers.

Return

0 on success; -1 on error. If an error occurs, check the *errno* global variable for the error number.

s11w_stop_buffers

Description

Stops DMA transfer after the current buffer has completed, whether DMA is occurring continuously or noncontinuously. If DMA is continuous, also dismantles ring buffer mode and frees the resources it consumed.

Syntax

```
int s11w_stop_buffers(S11wDev *s11w_p);
```

Arguments

s11w_p S11W device handle returned from *s11w_open*

Return

0 on success; -1 on error. If an error occurs, check the *errno* global variable for the error number.

s11w_wait_for_buffer

Description

Blocks until the specified buffer is returned from the driver.

Syntax

```
void *s11w_wait_buffers(S11wDev *s11w_p, int bufnum);
```

Arguments

s11w_p S11W device handle returned from *s11w_open*

bufnum buffer number for which to block. Completed buffers are numbered cumulatively starting with 0 when the S11W is opened. (Thus, the allocated buffer number is the transferred buffer number modulo the number of allocated buffers.)

NOTE: If you wait for all the buffers, the driver is left with none to use when this call returns, and an overrun or underrun will occur.

Return

Address of last completed buffer on success; NULL on error. If an error occurs, check the *errno* global variable for more information.

s11w_wait_for_next_buffer

Description

Blocks until the next buffer is returned from the driver. Returns immediately if a buffer is already complete. The completed buffers are numbered consecutively, so the first call to *s11w_wait_for_next_buffer* returns the address of buffer 0, the next will be 1, and so on.

Syntax

```
void *s11w_wait_next_buffer(S11wDev *s11w_p);
```

Arguments

s11w_p S11W device handle returned from *s11w_open*

Return

Address of completed buffer on success; NULL on error. If an error occurs, check the *errno* global variable for more information.

s11w_write

Description

Perform a write on the S11W. The UNIX 2 GB file offset bug is avoided during large amounts of input or output; that is, writing past 2^{31} does not fail. This call is not multibuffering, and no transfer is active when it completes.

Syntax

```
int s11w_write(S11wDev *s11w_p, void *buf, int size);
```

Arguments

s11w_p S11W device handle returned from *s11w_open*

buf address of buffer to write from

size size of write in bytes

Return

The return value from *write*; *errno* is set by *write* on error.

foi_parity_error**Description**

Checks to determine if a parity error has occurred since the last time this routine was called and returns 0 if not, 1 if so, and -1 if the routine is not supported for a particular device or an illegal argument was provided.

Syntax

```
int foi_parity_error(S11wDev *s11w_p);
```

Arguments

s11w_p S11W device handle returned from *s11w_open*

Return

0 if no error; 1 if parity error; -1 if the routine is not supported or if an illegal argument was provided.

Testing

You can perform diagnostics on the S11W by installing an optional loopback connector and executing the `loopback.c` diagnostic program included with the standard S11W software. The diagnostic program requires that the S11W driver be installed.

To test the S11W, loop output data from the SPARC back panel or the end of the device cable back to the input. The optional EDT S11W Loopback Kit, part number 012-00067, provides test connectors for both loopback configurations. The loopback kit contains complete instructions for the S11W loopback diagnostics.

The diagnostic program `loopback.c`, used in conjunction with the loopback test connectors, performs the tests described in the table below. To install the program, enter:

```
make loopback
```

Test Name	Test Operation Performed
Loopback Data Test	Write successive values from 0 to 0xFFFF, then read each word and compare the expected values with the actual values.
Swap Test	With the SWAP bit set, write and read walking ones and zeroes. This test exercises different data paths in the S11W.
Block Loopback	Write 1000 blocks of 0x1F000 words, then read back the value of the last word in each block, comparing the actual value with the expected value.
Function/Status Bits	Output all function bits, and compare the reported function bits in the Status Register with the looped-back status bits.
ATTN signal from pulsed F2	Tell the driver to get a <i>SIGIO</i> signal on ATTN, then pulse FNCT2 (which loops back to ATTN), and check for occurrence of the <i>SIGIO</i> signal.
Error on ATTN	Tell the driver to give an <i>EINTR</i> error if DINT comes in between I/O requests; pulse FNCT2; then perform a read and check for an error return.
5-second read timeout	Ask the driver to give a 5-second read timeout, then check for the duration and an <i>ETIMEDOUT</i> error return.
5-second write timeout	Ask the driver to give a 5-second write timeout, then check for the duration and an <i>ETIMEDOUT</i> error return.

Table 2. Loopback Diagnostic Tests

See the sections entitled *Registers* and *Signals* for further information.

Error Conditions

The table below shows error codes for the S11W and the error condition represented by each code.

Error Code	Failing Command	Error Condition
EINVAL	<i>ioctl()</i>	An invalid command was used, or an invalid length was specified for the buffer
EFAULT		An argument points outside the allocated address space.

Table 3. Error Codes and Conditions

Hardware

This section describes the S11W interface, registers, connectors, and timing. Figure 1 shows a block diagram of the S11W interface.

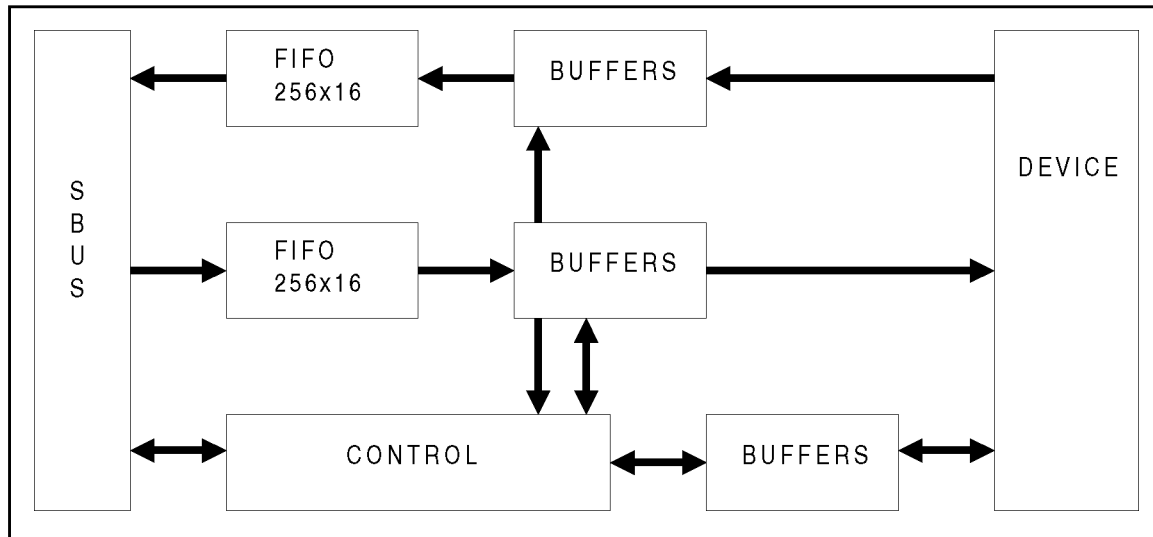


Figure 1. Block Diagram of the S11W Interface

SBus Interface

The interface to the SBus supports data transfer at 2, 4 and 16 bytes per request. The interface is implemented using programmable logic and high-speed register files. The SBus DMA address is maintained in an application-specific integrated circuit (ASIC).

FIFO

The S11W uses First-In-First-Out (FIFO) memories to buffer the data flow to and from the SBus. These FIFOs can store up to 512 bytes.

Device Interface

The device interface is implemented with Unibus Driver/Receivers and 180/390 terminators. The receivers have a 1 V hysteresis and a 2 V noise immunity. The drivers are open-collector type. The DR11W handshake, counters, and control are implemented in the ASIC with the SBus address counter.

See the *Signals* section for further details on the device signal usage.

Logic Levels

The S11W uses the standard DR11W drivers and receivers. These parts have Schmitt trigger inputs and a switching threshold set to equalize high and low noise margins. You can use TTL devices for short distances, but we do not recommend it. The drivers are open-collector drivers, capable of driving the required 105-terminating networks at each cable end.

Signals

This section describes the kinds of signals the S11W uses, how they are connected, and provides you with a suggestion for using the signals in your application for data input and output.

Connector Pinout

The S11W uses a high-density 80-pin I/O connector. The pinout and construction of this connector adapts easily to standard DR11W 40-pin, .100 x .100 connectors.

The high-density mating connector is an AMP connector, AMP part number 749111-7, with a straight-shielded backshell (AMP P/N 749196-1) or right angle backshell (AMP P/N 749205-1). The pinout described in the table below ensures that the high density connector and the P1 and P2 connectors of the EDT cable mate directly with standard 40-pin connectors.

Interpret the connector pinout table below in one of the following ways, depending upon the type of cable you are using.

EDT Model CAB-A

The column labeled AMP represents the AMP connector at one end of the cable. This end plugs into the AMP connector on the S11W. The columns labeled STD P1 and STD P2 represent standard 40-pin connectors at the ends of the Y on the other end of the cable. These ends plug into the user device.

EDT Model CAB-B

Both ends of the cable have AMP connectors, so the column labeled AMP represents both ends. The columns labeled STD P1 and STD P2 are irrelevant. Cabling is swapped; AMP 1 at one end connects to AMP 41 at the other, AMP 2 to AMP 42, and so on until AMP 40 at one end connects to AMP 80 at the other. This cable can connect two S11W modules, or one S11W to one S16D.

EDT Model CAB-D

Both ends of the cable have AMP connectors, so the column labeled AMP represents both ends. The columns labeled STD P1 and STD P2 are irrelevant. Cabling is straight through.

AMP	STD P1	DEC P1	Signal		DEC P2	STD P2	AMP
1	1	VV	DO15	DI15	VV	1	41
2	2	UU	DO00	DI00	UU	2	42
3	3	TT	DO14	DI14	TT	3	43
4	4	SS	DO01	DI01	SS	4	44
5	5	RR	DO13	DI13	RR	5	45
6	6	PP	DO02	DI02	PP	6	46
7	7	NN	DO12	DI12	NN	7	47
8	8	MM	DO03	DI03	MM	8	48
9	9	LL	DO11	DI11	LL	9	49
10	10	KK	DO04	DI04	KK	10	50
11	11	JJ	DO10	DI10	JJ	11	51
12	12	HH	DO05	DI05	HH	12	52
13	13	FF	DO09	DI09	FF	13	53
14	14	EE	DO06	DI06	EE	14	54
15	15	DD	DO08	DI08	DD	15	55
16	16	CC	DO07	DI07	CC	16	56
17	17	BB	NC	NC	BB	17	57
18	18	AA	GROUND	GROUND	AA	18	58
19	19	Z	CYCRQ B	GROUND	Z	19	59
20	20	Y	GROUND	GROUND	Y	20	60
21	21	X	END CYCLE	GO H	X	21	61
22	22	W	GROUND	GROUND	W	22	62
23	23	V	STATUS C	FNCT1	V	23	63
24	24	U	GROUND	GROUND	U	24	64
25	25	T	STATUS C	C1 CNTRL	T	25	65
26	26	S	GROUND	GROUND	S	26	66
27	27	R	STATUS B	FNCT2	R	27	67
28	28	P	GROUND	GROUND	P	28	68
29	29	N	INIT	C0 CNTL	N	29	69
30	30	M	GROUND	GROUND	M	30	70
31	31	L	STATUS A	FNCT3	L	31	71
32	32	K	BURST RQ	FNCT3	K	32	72
33	33	J	WC INC ENB	BC INC ENB	J	33	73
34	34	H	GROUND	GROUND	H	34	74
35	35	F	READY	A00	F	35	75
36	36	E	GROUND	GROUND	E	36	76
37	37	D	ACLO FNCT2	ATTN	D	37	77
38	38	C	GROUND	GROUND	C	38	78
39	39	B	CYCRQ A	BUSY	B	39	79
40	40	A	GROUND	GROUND	A	40	80

Table 4. Connector Pinout

The table below describes each signal by name, I/O type, and polarity. An I in the table indicates the signal is an input to the S11W, and an O indicates an S11W output. An H indicates the signal performs the function described in the table at a logic high (or +3 volts). An L indicates the signal performs the named function at a logic low. A P indicates the signal is programmable.

Synchronous Control Signals

The table below describes the eight signals controlling the synchronous DMA transfer cycle. These signals are sampled only during a BUSY cycle while READY is not asserted. Devices can implement these signals as required for device applications.

Name	I/O	Assert	Description
C0 CNTRL	I	H	Not supported by S11W. DR11W uses this signal to indicate writing a byte.
C1 CNTRL	I	H	If this signal is enabled by the DIRS0 and DIRS1 bits of the command register, C1 is used by a device to control the direction of DMA. C1 overrides the direction of data transfers indicated in application software, such that a <i>read()</i> system call becomes a <i>write()</i> instead.
FNCT1	O	P	This signal is used to control the device as the user defines. It is typically used as a DMA direction indicator from the host to the device. When used in this way, the device loops FUNCT1 back to C1, which indicates the direction.
FNCT2	O	P	This signal is used to control the device as the user defines. It is typically used to indicate that the device requires attention. In host-to-host applications, FUNCT2 is tied to the ATTN input of the other S11W.
FNCT3	O	P	This signal is used to control the device as the user defines.
STATUS A	I	H	Input for device status, read with STATA in status register.
STATUS B	I	H	Input for device status, read with STATB in status register.
STATUS C	I	H	Input for device status, read with STATC in status register.

Table 5. Synchronous Control Signals

Handshake Signals

These five signals perform the DR11W transfer cycle. GO and EOC are optional.

Name	I/O	Assert	Description
READY	O	H	Indicates that the S11W is ready for a DMA cycle to be initiated. This signal is low when a DMA cycle is in progress. Use this signal to qualify all other control signals, and take no other action when READY is asserted. Wait until the final BUSY edge before storing data, because an application can assert READY before the last DMA cycle is complete.
GO	O	H	A 100 ns pulse occurring at the beginning of the DMA cycle, when READY is asserted. This signal is provided for compatibility with DR11W. We recommend that your application use READY.
CYCRQ A	I	P	To initiate a transfer, the device asserts CYCRQ A after READY has cleared. By default, the rising edge of CYCRQ A starts the transfer; the high pulse on CYCRQ A must be at least 100 ns to do start the transfer. The user device ordinarily initiates CYCRQ A when READY is false, BUSY is false, and the device requires data. CYCRQ A clears when the S11W sets BUSY and the requested transfer is in progress. You can configure the S11W so that the falling edge of CYCRQ A starts the transfer using bit D1 of the configuration register.
CYCRQ B	I	P	Similar to CYCRQ A. The S11W combines CYCRQ A and B using the logical OR operation. Most applications use only CYCRQ A or drive both signals simultaneously. If the device does not use CYCRQ B, you must disable it or set it to low using the configuration register.
BUSY	O	P	The primary data transfer strobe. The S11W asserts BUSY in response to a CYCRQ. After a delay specified by the input skew, the S11W latches control signals and input data from the rising edge of BUSY. On output cycles, data is valid on the falling edge of BUSY. The polarity of BUSY is reversed in link mode applications and loopback, in order for BUSY to act as CYCRQ to the other DR11W or S11W interface. Use the configuration register to reverse the polarity of BUSY and to program input/output skew.
END CYCLE	O	H	A 100 ns pulse occurring on the falling edge of the last BUSY. This signal is provided for compatibility with DR11W. You can use the rising edge of EOC to signal the end of DMA, but this is not recommended as some DR11W emulators do not implement EOC.

Table 6. Handshake Signals

Asynchronous Control Signals

The table below describes the two signals controlling the DR11W device and host operating state, asynchronous with the control signals. These signals affect the S11W when they are asserted.

Name	I/O	Assert	Description
ATTN	I	H	Interrupt the SBus host if this signal is asserted. By default, ATTN also terminates a DMA in progress, as does DR11W. Using the S11W configuration register, you can make this interrupt independent of DMA so that devices can signal the host without disturbing DMA activity.
INIT	O	P	A programmable signal from the S11W used to reset the device. You can implement this signal as a fourth function bit if your application does not require complete DR11W compatibility.

Table 7. Asynchronous Control Signals

Unimplemented DR11W Signals

The following signals are provided for compatibility with DR11W but are not implemented by the S11W.

BURST RQ	An input to the DR11W for optimizing bus usage. The S11W optimizes SBus cycles automatically.
WC INC ENB	Allows a user device to control the DR11W word counter. Implementing this signal would interfere with S11W bus usage optimization.
BC INC ENB	Allows a user device to control the DR11W byte counter. Implementing this signal would interfere with S11W bus usage optimization.
A00	Formerly used in conjunction with the C0 and C1 signals to implement byte writes. Implementing this signal is impractical for high-speed DMA transfers.

Timing

Figure 3 shows the timing diagram for the S11W interface. The timing parameters shown in the diagram refer to the S11W during DMA transfers, in response to *read* or *write* system calls.

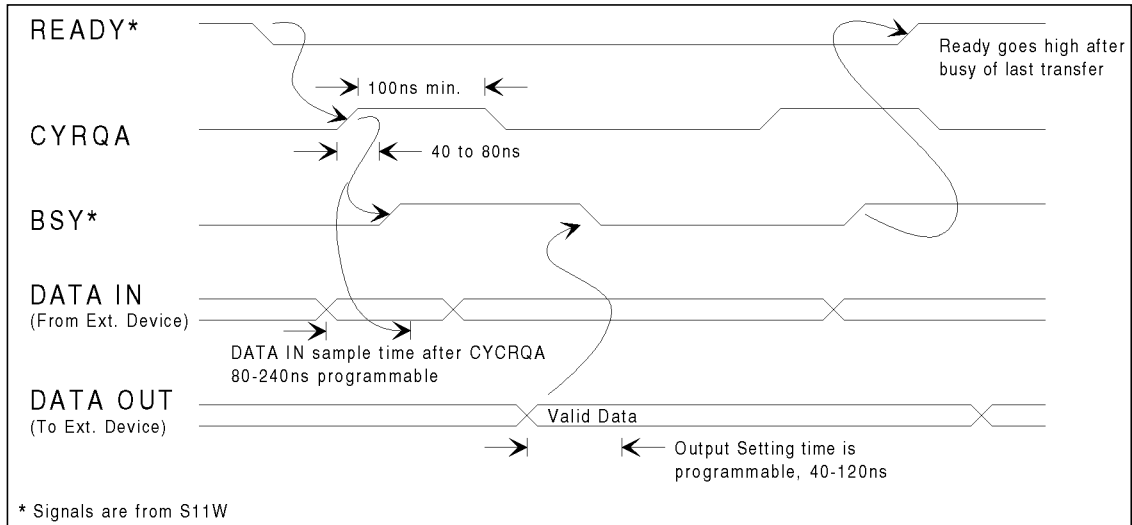


Figure 3. Timing Diagram

Registers

The S11W is configured and controlled with three 16-bit registers and two 32-bit registers, plus a data register and an FCode PROM.

Applications can access the S11W registers through the DMA library routines, or if necessary by means of *ioctl()* calls with S11W-specific parameters, as defined in the file *s11w.h*.

SBus Addresses

The addresses listed in the table below are offsets from the SBus slot base addresses. Obtain the SBus base address from the SBus host documentation. The figure below describes the S11W interface registers in detail.

0x0000.8010	data register		not used
0x0000.800C	DMA end address (write-only)		
0x0000.8008	DMA address		
0x0000.8004	not used	word count	
0x0000.8000	command/status		configuration register
			ROM byte 0x7FFF
	ROM		
0x0000.0000	ROM byte 0		
Byte	0	1	2 3
Word	0		1

Figure 4. SBus Addresses

Command Register

The command register is a 16-bit write-only register at address 0x8000.

Bit	S11W_	Description															
D15	DISINT	Disable interrupts to the SBus when set. The default is disabled.															
D14	REOD	Reset EODMA, the end-of-DMA interrupt bit in the status register. This value does not affect the state of the RDY status bit.															
D13	RATT	Reset the Attention interrupt bit in the status register. This does not affect the state of the <i>ATTN</i> signal or the <i>ATTN</i> status bit.															
D12	FCYC	Force Cycle Request, used in link applications to initiate cycle request handshakes between S11W devices. To achieve the same effect as an external cycle request, set this bit after setting the GO bit. The S11W driver takes care of this automatically in response to a read or write command, when the application sets the FCYC bit in the command word using S11S_READ_COMMAND or S11S_WRITE_COMMAND .															
D11–9		Unused															
D8	BCLR	Board Clear. Set BCLR to abort the DMA in process and clear the S11W FIFOs. This operation does not send <i>INIT</i> to the DR11W device.															
D7	INIT	Initialize device. The S11W asserts the <i>INIT</i> signal on the interface as long as <i>INIT</i> is set to 1. The <i>INIT</i> signal is not a pulse; the software determines the length of <i>INIT</i> according to the requirements of the device.															
D6–4	FNCT3 FNCT2 FNCT1	These function control bits are passed directly to the device interface. If the PFCT2 bit is asserted in the configuration register, the <i>FNCT2</i> signal pulses when you assert the <i>FNCT2</i> bit, allowing <i>ATTN</i> to be pulsed toward the device.															
D3–2	DIRS1 DIRS0	Select the direction of the DMA transfer according to this table. <i>read</i> refers to a read operation from the SBus to the DR11W device.															
		<table border="1"> <thead> <tr> <th><u>DIRS1</u></th> <th><u>DIRS0</u></th> <th><u>Direction</u></th> </tr> </thead> <tbody> <tr> <td>0</td> <td>0</td> <td>use DR11W-C1 input signal (0=read, 1=write)</td> </tr> <tr> <td>0</td> <td>1</td> <td>write</td> </tr> <tr> <td>1</td> <td>0</td> <td>use FNCT1 bit (0=read, 1=write)</td> </tr> <tr> <td>1</td> <td>1</td> <td>read</td> </tr> </tbody> </table>	<u>DIRS1</u>	<u>DIRS0</u>	<u>Direction</u>	0	0	use DR11W-C1 input signal (0=read, 1=write)	0	1	write	1	0	use FNCT1 bit (0=read, 1=write)	1	1	read
<u>DIRS1</u>	<u>DIRS0</u>	<u>Direction</u>															
0	0	use DR11W-C1 input signal (0=read, 1=write)															
0	1	write															
1	0	use FNCT1 bit (0=read, 1=write)															
1	1	read															
D1	BLKM	Assert this bit when the direction of the DMA transfer is constant throughout the entire operation. This allows you to maximize throughput with SBus burst transfers and S11W FIFOs.															
D0	GO	Initiate the DMA. <i>READY</i> is cleared, and <i>GO</i> is pulsed on the DR11W interface.															

Table 8. The Command Register

Status Register

The status register is a 16-bit read-only register at address 0x8000.

Bit	S11W_	Description
D15	INT	Interrupt pending when low.
D14	EODMA	Asserted low when the end-of-DMA interrupt is pending. When READY makes the transition to asserted, a DMA cycle is complete and EODMA is set. Clear EODMA with REOD, bit 14 of the command register. EODMA causes an SBus interrupt if interrupts are enabled—that is, when DISINT (bit 15 of the command register) is 0.
D13	INTATT	Shows when a device interrupt is pending. INTATT is set low to show a pending device interrupt when the external ATTN input is asserted. INTATT causes an SBus interrupt when DISINT (bit 15 of the command register) is 0, which enables interrupts. Clear INATT by setting RATT, bit 13 of the command register.
D12	1	Always set to 1 on the current S11W.
D11	ATTN	Reflects the state of the external S11W ATTN input.
D10–8	STATC STATB STATA	Reflects the state of the external S11W inputs STATA, STATB, and STATC.
D7	INIT	Reads back the state of the command register INIT bit
D6–4	FNCT3 FNCT2 FNCT1	Reads back the state of command register bits FNCT3, FNCT2, and FNCT1.
D3–2		Always set to 0 on the current S11W.
D1	BLKM	Reads back the state of command register bit BLKM.
D0	RDY	Indicates when SBus DMA is in progress. This bit is set whenever the S11W READY signal is not asserted (low), which indicates that a transfer is occurring. This bit is clear when the system is idle.

Table 9. The Status Register

Configuration Register

The configuration register is a 16-bit register at address 0x8002.

Bit	S11W_	Description
D15–13		Unused
D12	PFCT2	When set, the S11W FUNCT2 output pulses high for 300 ns for each time the FUNCT2 bit in the command register transitions between 0 and 1. Otherwise, the FUNCT2 output reflects the state of the FUNCT2 bit. Set this bit when using FUNCT2 in link mode in order to pulse the ATTN input of the other S11W device.
D11	NCOA	When No Clear on Attention is set, an incoming ATTN signal does not abort a DMA in progress. Set this bit if your application requires an ATTN interrupt independent of DMA transfers.
D9–10	RSVD	Reserved
D8	SWAP	SWAP determines which byte of an SBus half-word ends up on which half of the S11W 16-bit bus. When SWAP is zero, the byte order is the DR11W standard, which is the opposite of the Sun short order.
D7	RDYT	The READY timing bit determines when the S11W asserts READY on the last transfer of a DMA cycle. When this bit is zero, the S11W asserts READY during BSY of the last cycle. When this bit is one, the S11W asserts READY at the end of BSY and coincident with EOC. The DR11W standard asserts READY during BSY; but many devices function better with READY asserted after BSY if their strobe is strictly a combinatorial AND of READY and BSY.
D6–5	INSK1 INSK0	Input skew sets the time from the start of BSY until the S11W samples input data and control. An input skew of 0 represents one SBus cycle, typically 40–66 ns. Each increment adds one clock pulse.
D4–3	OUTSK1 OUTSK0	Output skew sets the time between valid output data and the BSY transition that terminates the cycle. An output skew of 0 represents a data setup minimum of 10 ns. Each increment adds one SBus clock pulse, typically 40–66 ns.
D2	ENBB	When set, this bit enables the CYCRQ _B input as required for strict DR-11 compatibility. If your application does not use CYCRQ _B , clear this bit, because an open CYCRQ _B input is pulled to an active state by the terminator and masks CYCRQ _A , thereby inhibiting all transfers.
D1	CYCP	The Cycle Request Polarity bit determines which edge of the CYCRQ _A or CYCRQ _B input initiates data transfer. If this bit is set to zero, a falling edge initiates transfer. If this bit is set to one, a rising edge initiates transfer. The DR11W standard requires a rising edge. Some devices use a falling edge so that disconnected inputs are inactive. In new designs, we recommend using the negative edge to initiate transfers, so that unplugging the device cable does not produce a clock edge.
D0	BSYP	BSYP determines the S11W external signal BSYH polarity. If BSYP is zero, the external BSY is asserted high (positive is true); this is the default. If you remove the hardware jumper on the S11W, the BSYP polarity is forced to negative true and this bit is ignored. Use the jumper for devices which require a negative-true busy as a strobe, and don't qualify BSY with READY. In this case, setting the configuration register with the jumper installed causes an active BSY transition for the device.

Table 10. The Configuration Register

DMA Word Count Register

The DMA word count register is a 16-bit register at address 0x8006.

Bit	Description
WC15–0	The word count bits specify how many 16-bit words to transfer from the S11W. Each word is one SBus half-word, or two bytes.

Table 11. The DMA Address Register

DMA Address Register

The DMA address register is a 32-bit register at address 0x8008.

Bit	Description
A31–20	These addresses latch the 1 MB page addressed by the DMA.
A19–1	These address bits contain the starting SBus address of the DMA block. When read, they display the next address to access on the SBus. When DMA is complete, the SBus address equals the next address after DMA End Address.
A0	Set to 0.

Table 12. The DMA Address Register

DMA End Address Register

The DMA address register is a 16- or 32-bit register at address 0x800C.

Bit	Description
EA31–20	Set all these bits to ones.
EA19–1	These end-address bits specify the last SBus address to transfer on the SBus side of the FIFOs.
EA0	Set to 0.

Table 13. The DMA End Address Register

Data Register

The data register is a 16-bit read-write register at address 0x8010.

Bit	Description
D15–0	When DMA is in progress, DMA writes to this register. Otherwise, data written to this register appears on the DR11 output data bus to this register. Reading this register yields the current contents of the DR11 input data bus.

Table 14. The Data Register

Specifications

The S11W conforms to the following specifications.

SBus Compliance

Number of slots:	1
Transfer size	2, 4, and 16 bytes
DVMA master	Yes
SBus memory	32 bits
Clock rate	25 MHz

Device Data Transfer

Format	16-bit parallel word
Handshake	2-wire asynchronous handshake
Transfer types	Programmed I/O, DMA block transfer, configurable DMA direction
Transfer rate	Dependent upon attached device, host load, and block size. Maximum 8 MB/second.
Signal polarity	Data is true. Control signals are programmable; the default is as specified by DR11W.
Signal timing	Programmable handshake timing.
Buffers	512-byte FIFO for input, 512-byte FIFO for output

Software

Drivers for Sun OS Version 4.1.x and System V Version 4 (Solaris 2.0)

Power

5 V at 2 A

Environmental

Temperature	Operating: 10 to 40 C Nonoperating: -20 to 60 C
Humidity	Operating: 20 to 80% noncondensing at 40 C Nonoperating: 95% noncondensing at 40 C

Physical

Dimensions	3.3" x 5.78" x 0.5"
Weight	6 oz.

References

DR11W Direct Memory Interface Module User's Guide, available from Digital Equipment Corporation, Maynard, Mass. Ask for part number EK-DR11W-UG-001.

See the UNIX man pages for *ioctl(2)*, *select(2)*, *read(2)*, *open(2)*, *write(2)*, *aioread(2)*, *aiowrite(2)*, and *aiowait(2)* for specific information about these system calls.

See the SunOS Installation documents for more information about configuring the kernel for asynchronous I/O.

Appendix A `ioctl()` Parameters

Engineering Design Team recommends that applications use the software library interface documented starting on page 6. However, if necessary, the following `ioctl` parameters are useful for application programs. Others may be defined, but are used for Engineering Design Team's internal purposes only.

S11G_STATUS Get the status register. Provide an unsigned short as an argument.

S11x_CONFIG Set (**S**) or get (**G**) the configuration register. These `ioctl()` parameters set the configuration bits described in `s11w.h` to match the requirements of the device connected to the S11W. When you load the S11W driver, the `s11w_attach()` routine initializes the configuration register to the value if **S11W_DEC_CONFIG**, as defined in `s11w.h`. Provide an unsigned short as an argument.

S11x_COMMAND

Write the given word to the command register immediately, or read back what you have written. Use this as a preliminary handshake, to set up a transfer size with a device and possibly to change FCNT lines. The command register changes with any subsequent `read` or `write` system call. Provide an unsigned short as an argument.

NOTE: Do not set the GO bit explicitly. The driver must set up the address space properly for a DMA transfer, and it then sets the GO bit for each data transfer.

Do not set the FCYC bit using this command. If you need to set it explicitly, use **S11S_WRITE_COMMAND** and **S11S_READ_COMMAND**, so that the driver can set this bit when starting the DMA transfer.

For more information, see the discussions of the `ioctl()` parameters **S11x_WRITE_COMMAND** and **S11x_READ_COMMAND**.

S11S_READ_COMMAND

Set the word to write to the command register on subsequent `read` system calls. This word is valid only until the next `open()` of the S11W. Use the **S11S_DEF_READ_COMMAND** parameter to define a default word to write to the command register on `reads` that remains valid across `open` calls to the device. Provide an unsigned short as an argument.

S11W_DEF_READ in `s11w.h` defines the default value written to the command register on `read` system calls.

S11G_READ_COMMAND

Get the word that will be written to the command register on subsequent `read` system calls. This word is valid only until the next `open()` of the S11W. Use the **S11G_DEF_READ_COMMAND** parameter to get the default word written to the command register on `reads` that remains valid across `open` calls to the device. Provide an unsigned short as an argument.

S11S_WRITE_COMMAND

Set the word to write to the command register on subsequent `write` system calls. This word is valid only until the next `open()` of the S11W. Use the **S11S_DEF_WRITE_COMMAND** parameter to define a default word to write to the command register on `reads` that remains valid across `open` calls to the device. Provide an unsigned short as an argument.

S11W_DEC_WRITE in `s11w.h` defines the default value written to the command register on `write` system calls.

S11G_WRITE_COMMAND

Get the word to write to the command register on subsequent *write* system calls. This word is valid only until the next *open()* of the S11W. Use the **S11G_DEF_WRITE_COMMAND** parameter to get the default word written to the command register that remains valid across *open* calls to the device. Provide an unsigned short as an argument.

S11x_DATA

Set or get the data register. Use this parameter to read the data currently on the S11W input signals. Setting this parameter puts data on the interface without asserting any handshake signals. Provide an unsigned short as an argument.

NOTE: Programmed I/O is much slower than direct memory access for large blocks of data.

S11S_ATT_SIG

Set the signal to send to the current process upon receipt of an *ATTN* interrupt. This *ioctl()* parameter enables device interrupts so the S11W can receive the attention signal, even when I/O is not in progress. Set the signal to zero to disable the signal upon *ATTN*. Provide an unsigned short as an argument.

S11S_ATT_ERR

If **ATT_SIG** is not enabled, cause a *read* or *write* system call to return with the specified error if the driver receives an *ATTN* since the last *read* or *write*. This also produces an error after a transfer, if the *ATTN* occurs during the transfer. The value of the *NCOA* bit of the configuration register (D11) determines whether the transfer is to be aborted with *ATTN*. Provide an unsigned short as an argument.

EDTS_RTIMEOUT

Set the timeout lengths for reads from the S11W. The timeout units are hundredths of seconds. Provide an unsigned short as an argument.

EDTS_WTIMEOUT

Set the timeout lengths for writes to the S11W. The timeout units are hundredths of seconds. Provide an unsigned short as an argument.

S11S_SEND_INIT_PULSE

Send an *INIT* pulse. The short value passed is the pulse length in hundredths of a second. Provide an unsigned short as an argument.

EDTS_SHORT_XFER

Set short transfer timeout mode. Provide an unsigned short as an argument.

Set to 0 to get timeout errors when appropriate.

Set **EDTS_SHORT_XFER** to 1 or 2 to eliminate timeout errors. Upon timeout in this mode, *read* or *write* system calls return the size of the transfer instead of -1. When using asynchronous I/O, the driver places the transfer size into the *aio_result* structure.

Set **EDTS_SHORT_XFER** to 2 to reset the timeout counter at the start of each transfer. This is useful for continuous asynchronous input and output.

EDTx_DEBUG_LEVEL

Set (**S**) or get (**G**) the debug level. A value of 1 reports driver status when *modinfo* (Solaris) or *modstat* (SunOS 4.1.x) is run. Other values are used for purposes internal to EDT. Provide an unsigned short as an argument.

EDTx_DEBUG_INTR

Set (**S**) or get (**G**) the debug interrupt level—1 generates full driver tracing and may interfere with normal driver operation. Provide an unsigned short as an argument.

S11G_DMAADDR, S11G_DMALAST

Access the hardware registers. **S11G_DMAADDR** shows where the DMA is currently accessing the buffer of the user device. These parameters require the address pointed to (the third parameter to `ioctl`) to contain unsigned integers (32 bits).

NOTE: These parameters are provided for completeness—only the driver should manipulate the data in these registers. Use the `read` and `write` (or `aioread` and `aiowrite`) system calls to perform DMA; the S11W driver already handles the details of the transfer.

S11G_LASTCOUNT

Return the last value written to the DMA count register. Provide an unsigned short as an argument.

S11G_LASTADDR

Return the last value written to the DMA address register. Provide an unsigned integer as an argument.

S11G_STAT

Get bits D8–10 in the status register, which reflect the state of the external S11W inputs STATA, STATB, and STATC. The value for these bits is returned in the least significant bits of the word. They are not shifted, so they do not appear in positions corresponding to the bits in the status register. Provide an unsigned short as an argument.

S11x_DIRS

Set (**S**) or get (**G**) bits D2 and D3, the direction bits, in the status or command register. The value for these bits is sent and returned in the least significant bits of the word. They are not shifted, so they do not appear in positions corresponding to the bits in the status or command register. Provide an unsigned short as an argument.

EDTS_TERMINATE

Shut down ring buffers and terminate DMA on all outstanding I/O requests. No argument is required.

EDTS_EODMA_SIG

Register an end-of-DMA signal when the next DMA operation has been completed. The third argument to the `ioctl` is the address of an unsigned integer specifying the number of the signal to send to the calling process. (SIGIO is recommended, as its purpose is to signal an I/O event.) This registration produces one occurrence of a signal; the signal handler must reregister each time a signal is required. Provide an unsigned short as an argument.

EDTS_NBUFIO

Initiates continuous ring-buffer mode. The third argument to the `ioctl` is the address of an unsigned integer specifying the number of buffers in the ring: legal values are between 1 and 40. The driver waits until it has received the specified number of requests for DMA operation, then allocates operating system resources to each buffer, and finally performs continuous DMA to each buffer on a round-robin basis, starting with the first request and wrapping to the beginning again after the last.

EDTG_NBUFIO

Returns the number of the buffer that has most recently completed DMA and is available for processing. Provide an unsigned integer as an argument.

EDT_FREE_BUF

Turns off continuous ring-buffer mode. No argument is required.

EDTG_DONECOUNT

Use with continuous ring-buffer mode (**EDTS_NBUFIO**). Get the count of the last buffer completed by the driver. The count starts at 0 and increments each time DMA on a buffer completes. Provide an unsigned integer as an argument.

EDTS_WAKEUP_DONECOUNT

Use with continuous ring-buffer mode (**S11S_NBUFIO**) and **S11G_DEVBUFS_COMPLETED**. This *ioctl* does not return until the count of buffers completed reaches the count supplied in the third argument to the *ioctl*. This causes the application to wait until the driver has performed the specified number of DMA operations. If the specified count has already been reached it returns immediately. Provide an unsigned integer as an argument.

The driver counter wraps at 2^{32} , and the driver is implemented to handle this behavior correctly. Therefore let the count in your application wrap as well.

EDTS_LOCKSTEP

Use with continuous ring-buffer mode (**S11S_NBUFIO**). Initiates lockstep mode, in which the driver waits for notification from the application before proceeding with DMA. The number of buffers processed before waiting are specified in the value of the address pointed to by the third argument to this *ioctl*. The driver then waits until it receives notification by means of the following *ioctl*. Provide an unsigned integer as an argument.

Setting the third argument to an address that points to 0 turns off lockstep mode.

EDT_FREERUN Turns off lockstep mode. No argument is required.

EDTS_APPBUFS_COMPLETED

Use with continuous ring-buffer mode (**S11S_NBUFIO**) and lockstep mode (**S11S_LOCKSTEP**). Instructs the driver to process the number of buffers as specified in the third argument to this *ioctl*, then wait until this *ioctl* is called again. Provide an unsigned integer as an argument.

EDTG_BYTECOUNT

Get the value of the *bytecount* variable in the driver—a variable used to count the number of bytes of DMA performed since the last time the value of *bytecount* was set. This parameter can help you keep track of how DMA is progressing. Provide an unsigned integer as an argument.

NOTE: When the value of *bytecount* reaches 4 GB, it is reset to 0. To be certain that the value is valid, therefore, use **EDTS_BYTECOUNT** to reset it to 0 before each DMA operation.

EDTS_BYTECOUNT

Set the value of the *bytecount* variable in the driver. Provide an unsigned integer as an argument.

EDTG_CUR_ADDR

Reads the contents of the current DMA address register. Provide an unsigned integer as an argument.

EDTG_CUR_COUNT

Reads the contents of the current count register. Provide an unsigned integer as an argument.

EDTG_NEXT_ADDR

Reads the contents of the next DMA address register. Provide an unsigned integer as an argument.

EDTG_NEXT_COUNT

Reads the contents of the next count register. Provide an unsigned integer as an argument.

EDTG_DMA_CMD

Reads the contents of the DMA command register. Provide an unsigned integer as an argument.

EDTG_WAITRDY

In ring buffer mode, ensures that all DMA resources are allocated before returning.