Symbios Logic
PCI-SCSI I/O Processors

Programming Guide
Version 2.1

INCREASING SCSI RELIABILITY
TolerANT
ACTIVE NEGATION TECHNOLOGY

J25972I

The products described in this publication are products of Symbios Logic Inc.

SCRIPTS and NASM are trademarks and TolerANT is a registered trademark of Symbios Logic Inc.

Ultra SCSI is the term used by the SCSI Trade Association to describe Fast-20 SCSI, as documented in the SCSI-3 Fast-20 Parallel Interface standard, X3.277-199X. Ultra2 SCSI is the term used by the SCSI Trade Association to describe Fast-40 SCSI, as documented in some versions of the SPI-2 draft standard.

It is the policy of Symbios Logic to improve products as new technology, components, software, and firmware become available. Symbios Logic, therefore, reserves the right to change specifications without notice.

The products in this manual are not intended for use in life-support appliances, devices, or systems. Use of these products in such applications without the written consent of the appropriate Symbios Logic officer is prohibited.

We use comments from our readers to improve Symbios product literature. Please e-mail any comments regarding technical documentation to pubs@symbios.com.

Purpose and Audience

This manual provides basic information on the SYM53C8XX family of PCI-SCSI I/O Processors to software developers writing device drivers for SCSI devices that use these products. It describes basic chip operation and provides detailed information on the SCRIPTS programming language, a high-level interface for controlling Symbios Logic SCSI processors. The programming examples and instructions in this guide assume that the device driver is in “C” language. The examples are primarily written for PC-based architectures, but SCRIPTS-based drivers can run on any hardware platform. More detailed information on SCSI specifications can be found in the references listed in the section “Additional Information.”

The diskette that accompanies this programming guide contains several “C” and SCRIPTS sample programs that implement many of the programming tasks discussed in this book. It also contains the NASM SCRIPTS assembler and the NVPCI SCRIPTS debugger. Please consult the “read me” file on the diskette for a list and description of all sample programs. For the most up-to-date versions of these and other sample programs, please contact the Symbios Logic electronic bulletin board.

Additional Information

ANSI SCSI-2 Standard, SCSI-3 Parallel Interface (SPI) Standard

The SCSI-2 document is the final, approved standard for SCSI-2. The SPI document is still in draft. Both of these documents can be obtained from:

Global Engineering Documents
15 Inverness Way East
Englewood, CO 80112
(800)-854-7179 or (303) 792-2181 (outside U.S.)
Ask for document number X3.131-199X (SCSI-2) or X3.253 (SPI)

SCSI Bench Reference, SCSI Encyclopedia

The SCSI Bench Reference is an applications-oriented guide to the basics of SCSI. It is intended as a quick reference guide only. The SCSI Encyclopedia is a multi-volume reference that describes all SCSI-1 and SCSI-2 commands and protocols. It contains detailed information on all topics covered. These and other reference materials can be ordered from:

ENDL Publications
14426 Black Walnut Court
Saratoga, CA 95070
(408) 867-6642

What Is SCSI? Understanding the Small Computer System Interface

This easy-to-read book contains a high-level overview of how SCSI works, based on the SCSI-1 standard. It is excellent for anyone with no exposure to SCSI. It can be obtained from:

Prentice Hall
Englewood Cliffs, NJ 07632
(201) 767-5937
Ask for document number ISBN 0-13-796855-8

Symbios Logic Electronic Bulletin Board

This BBS provides updated information on Symbios Logic SCSI products, including sample SCRIPTS.
(719) 533-7235

SCSI Electronic Bulletin Board

Contact this BBS for general information on SCSI, including the status of SCSI specifications, and electronic copies of draft standards.
(719) 533-7950

Symbios Logic Internet Anonymous FTP Site

This FTP site has general information similar to that on the electronic bulletin board. The address is: ftp.symbios.com (204.131.200.1) \pub\symchips\scsi.

Symbios Logic World Wide Web Home Page

The Symbios Logic home page has general information about our company and products. The address is http://www.symbios.com.

Revision Record

Page No.	Date	Remarks
all	8/96	Version. 2.0
	6/97	Version 2.1. Added information on SYM53C885 and SYM53C876; added chapter on programming multifunction controllers; minor typographical corrections.

Contents

Chapter 3
The SYM53C8XX Instruction Set

Chapter 4
Using the Symbios Logic Assembler

Chapter 8
Writing Device Drivers With SCRIPTS

Chapter 9
SCRIPTS Programming Topics

List of Figures

Chapter 5
The NASM Output File

Chapter 7
Integrating SCRIPTS Programs Into "C" Language Drivers

Chapter 8
Writing Device Drivers With SCRIPTS

Chapter 9
SCRIPTS Programming Topics

Chapter 10
Multi-Threaded I/O

List of Tables

Chapter 5
The NASM Output File

Chapter 6
Using the Registers to Control Chip Operations

Chapter 1

Introduction

What is Covered in This Guide

This manual provides basic information for writing device drivers that use the SYM53C810A, SYM53C815, SYM53C825A, SYM53C860, SYM53C875, and SYM53C895, SYM53C876, and the SCSI portion of the SYM53C885(this group of products is referred to as SYM53C8XX).

- This chapter introduces the SYM53C8XX features and functions, and the parts of the PCI-SCSI system that are involved in operating the chip.
- Chapter 2 describes the SCRIPTS processor and programming language in depth, including how SCRIPTS programs are integrated with “C” code to execute SCSI commands.
- Chapter 3 describes the SYM53C8XX instruction set, with detailed functional descriptions and usage guidelines for all of the instructions supported by the SYM53C8XX.
- Chapter 4 and Chapter 5 cover the Symbios Logic Assembler (NASM), including directives, and the .out file format.
- Chapter 6 contains functional and address information on the SYM53C8XX register set.
- Chapter 7 illustrates the relationship between the SCRIPTS program and the “C” language device driver.
- Chapter 8 and Chapter 9 address specific kinds of driver applications, with code samples for all applications discussed.
- Chapter 10 contains guidelines for writing SCRIPTS for multi-threaded applications.
- Chapter 11 contains specific information for programming the Symbios Logic multifunction controllers SYM53CX885 and SYM53C876.
- Chapter 12 provides guidelines that are specific to using the SYM53C8XX in a target device.
- Chapter 13 provides information on debugging SCRIPTS programs.
- The appendixes contain a listing of NASM error messages, a glossary, a register summary, and a sample multi-threaded SCRIPTS program.

This manual is written for users who are familiar with the SCSI and PCI specifications, and have a working knowledge of computer architectures and programming. The Preface of this document identifies sources for obtaining some of this background information, if needed.

Product Overview

The SYM53C8XX PCI-SCSI I/O Processor is based on the SYM53C7XX SCSI I/O Processor family architecture, with a host interface to the Peripheral Component Interconnect (PCI) bus.

The SYM53C8XX connects to the PCI bus without glue logic. The SYM53C810A and SYM53C860 are optimized for motherboard applications; a complete design can be implemented in less than four square inches of space on the motherboard. The SYM53C815, SYM53C825A, SYM53C875, and SYM53C895 are ideal for host adapter and motherboard applications, because of an added external memory interface which allows BIOS code to be placed in an external EEPROM to provide a bootable host adapter. The SYM53C825A, SYM53C875, and SYM53C895 have 4KB of onboard RAM for SCRIPTS instruction storage, to minimize PCI bus overhead by performing SCRIPTS instruction fetches without using the PCI bus. The SYM53C885 and SYM53C876 are multifunction controllers that each use only one PCI bus load. The SYM53C885 includes a SCSI I/O Processor and an Ethernet controller, and the SYM53C876 contains two independent SCSI functions.

The Symbios Logic SCSI I/O Processors are the first products to concentrate the functions of an intelligent SCSI adapter board onto a single chip. The SYM53C8XX integrates a high-performance SCSI core, a PCI bus master DMA core, and the SCSI SCRIPTS™ processor to meet the flexibility requirements of SCSI-3 and future SCSI standards. It executes multi-threaded I/O algorithms with minimum host processor intervention, reducing the protocol overhead required for SCSI operations to as little as one interrupt per SCSI I/O. The SCRIPTS language, a high-level instruction set, provides complete programmability of I/O operations and supports the flexibility needed for multi-threaded I/O algorithms. The SYM53C8XX uses SCRIPTS to provide: phase sequencing without processor intervention; automatic bus arbitration; data or phase comparison for independent SCSI algorithm decisions; and DMA interface control. All SYM53C8XX family chips are also supported by Symbios Logic software for connecting SCSI devices, including BIOS support for Symbios Logic SCSI processors and drivers for most types of SCSI peripherals under the major operating systems.

All SYM53C8XX chips feature on chip single-ended drivers; synchronous and asynchronous transfer capabilities; and Symbios Logic TolerANT® driver and receiver technology, for single-ended signal integrity in any cabling environment. They support bus mastering, automatic selection/reselection time-outs, 32-bit memory addressing, a 32-bit data bus, and PCI bursting. The features and functions of individual chips in the SYM53C8XX family are summarized in Table 1-1.

Note: the SCSI portion of the SYM53C885 is functionally comparable to the SYM53C875. For specific information on the features and functions of the SYM53C885, refer to the SYM53C885 Data Manual. For specific information on programming the Ethernet function of the SYM53C885, refer to the Symbios Logic PCI-Ethernet Programming Guide.

Note: the SYM53C876 has two SCSI functions, each comparable to the SYM53C875. For specific information on the features and functions of the SYM53C876, refer to the SYM53C876 Data Manual.

Table 1-1
Features and Functions of SYM53C8XX Family Chips

	SYM53C810A	SYM53C860	SYM53C815	SYM53C825A, SYM53C825AJ	SYM53C875, SYM53C875J, SYM53C875JB, SYM53C875N	SYM53C895
Max. Transfer Rate	5 MB/s async. 10 MB/s sync.	5 MB/s async. 20 MB/s sync. (w/ Ultra SCSI)	5 MB/s async. 10 MB/s sync.	10 MB/s async. 20 MB/s sync.	10 MB/s async. 40 MB/s sync. (w/ Ultra SCSI)	10 MB/s async. 80 MB/s sync. (w/ Ultra2 SCSI)
DMA FIFO Size (bytes)	80	80	64	88 or 536	88 or 536	112 or 816
Synchronous Offset (levels)	8	8	8	16	16	31
SCRIPTS RAM	no	no	no	yes	yes	yes
Differential SCSI	no	no	no	yes	yes	LVD and high voltage differential
Wide SCSI	no	no	no	yes	yes	yes
External Memory Interface	no	no	yes	yes	yes	yes

Table 1-1 (Continued)
Features and Functions of SYM53C8XX Family Chips

	SYM53C810A	SYM53C860	SYM53C815	SYM53C825A, SYM53C825AJ	SYM53C875, SYM53C875J, SYM53C875JB, SYM53C875N	SYM53C895
Instruction Prefetch	yes	yes	no	yes	yes	yes
Load/Store Instructions	yes	yes	no	yes	yes	yes
Enhanced Move Register Capability	no	no	no	yes	yes	yes
SCSI Selected As ID Bits	yes	yes	no	yes	yes	yes
Number of 32-bit SCRATCH Registers	2	2	2	10	10	10
PCI Caching	yes	yes	no	yes	yes	yes
Selectable IRQ Disable	yes	yes	no	yes	yes	yes
Big/Little Endian support	Little Endian	Little Endian	Big or Little Endian	Big or Little Endian (except 53C825AJ)	Big or Little Endian (except 53C875J, 53C875JB)	Big or Little Endian
Package	100 PQFP	100 PQFP	128 PQFP	160 PQFP	160 PQFP, 169 BGA, 208 PQFP	208 PQFP

Figure 1-1 is a block diagram of the SYM53C8XX, with a map of SCSI data and control paths through the chips.

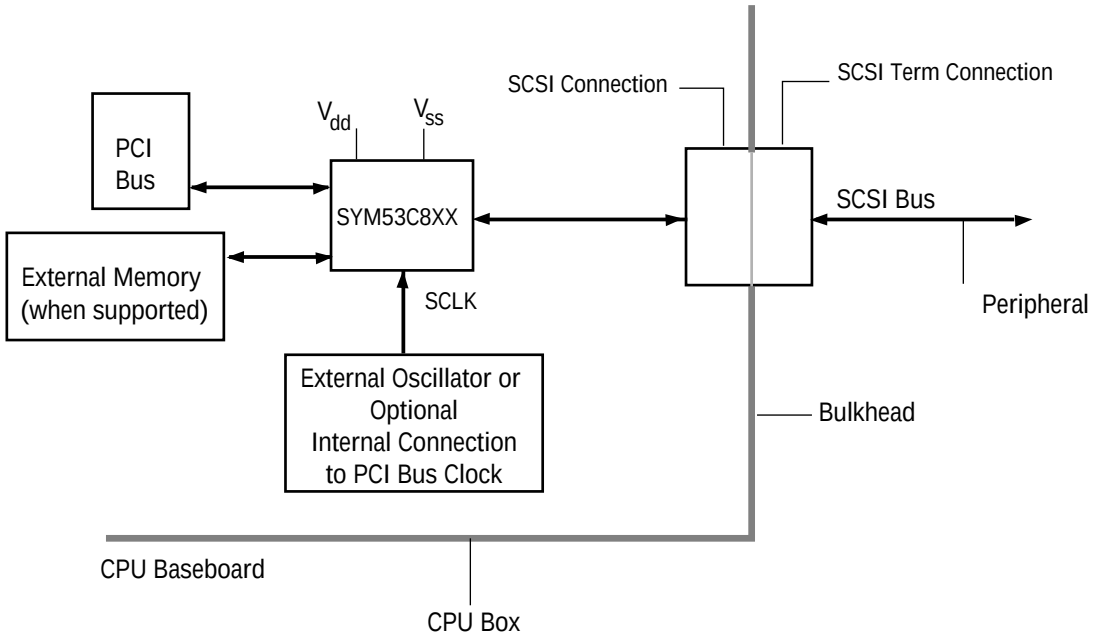


Figure 1-1
SYM53C8XX Block Diagram

Benefits of Ultra SCSI and Ultra2 SCSI

Ultra SCSI is an extension of the SCSI-3 standard that expands the bandwidth of the SCSI bus and allows faster synchronous SCSI transfer rates. When enabled, Ultra SCSI performs 20 megatransfers per second, which results in approximately doubling the synchronous transfer rates of fast SCSI-2. The SYM53C860 and SYM53C875 can perform 8-bit or 16-bit Ultra SCSI synchronous transfers as fast as 20 MB/s or 40 MB/s.

Ultra2 SCSI extends SCSI performance beyond Ultra SCSI rates, up to 40 megatransfers per second. It also defines a new physical interface, Low Voltage Differential SCSI (LVD), that retains the reliability of high voltage differential SCSI while allowing a longer cable and more devices on the bus than Ultra SCSI. The SYM53C895 can perform 16-bit, Ultra2 SCSI synchronous transfers as fast as 80 MB/s.

The advantages of Ultra SCSI and Ultra2 SCSI are most noticeable in heavily loaded systems, or large-block size applications such as video on-demand and image processing. One advantage of Ultra SCSI and Ultra2 SCSI are that they significantly improve SCSI bandwidth while preserving existing hardware and software investments. Symbios Logic Ultra SCSI and Ultra2 SCSI chips are all compatible with Fast-SCSI software; the only changes required are to enable the chip to negotiate for the faster synchronous transfer rates. The SYM53C860 and SYM53C875 can use the same board socket as an SYM53C810A and SYM53C825A, respectively, with the addition of an 80MHz SCLK. The SYM53C875 contains an internal SCSI clock doubler, allowing it to transfer data at Ultra SCSI rates with a 40MHz clock. The SYM53C895 contains an internal SCSI clock quadrupler, allowing it to transfer data at Ultra2 SCSI rates with a 40 MHz clock.

Some changes to existing cabling or system designs may be needed to maintain signal integrity at Ultra SCSI synchronous transfer rates. These design issues are discussed in the Ultra SCSI and Ultra2 SCSI chip data manuals.

System Overview

To execute SCSI SCRIPTS programs, the SYM53C8XX requires only a SCSI SCRIPTS starting address; all subsequent instructions are fetched from external memory or internal SCRIPTS RAM (when supported). The SYM53C8XX fetches up to eight dwords at a time across the DMA interface and loads them into the internal chip registers. When the chip is operating at its highest frequency, instruction fetching and decoding takes as little as 500 nanoseconds (ns). The chip fetches instructions until a SCRIPTS interrupt occurs or until an external, unexpected event (such as a hardware error) causes an interrupt. The full set of SCSI features in the instruction set allows re-entry to the algorithm at any point. This high level interface can be used for both normal operation and exception conditions.

How SCRIPTS Operations Control the SYM53C8XX

A typical SCRIPTS operation is illustrated in Figure 1-2. Before SCRIPTS operation begins, the host processor writes the Data Structure Address (DSA, 10-13h) register value to initialize the pointer for table indirect operations. To begin SCRIPTS operation, the host processor writes the starting address of the SCRIPTS instructions into the DMA SCRIPTS Pointer Register (DSP, 2C-2Fh) register of the SYM53C8XX. Once it receives this address, the SYM53C8XX becomes a bus master and fetches the first SCRIPTS instruction. The SYM53C8XX executes all steps of the instruction, moving through the appropriate bus phases and interrupting only when the SCRIPTS operation is completed or the SYM53C8XX requires service from the external processor. This leaves the host processor free for other tasks. The SYM53C8XX fetches the next instruction, and the process begins again.

Software developers can develop SCSI SCRIPTS source code in any text editor. The Symbios Assembler (NASM) assembles SCRIPTS code into an array of assembled SCRIPTS instructions that can be included in the main “C” language program and linked together to

create an executable driver. When compiled, these programs control the operation of the SYM53C8XX.

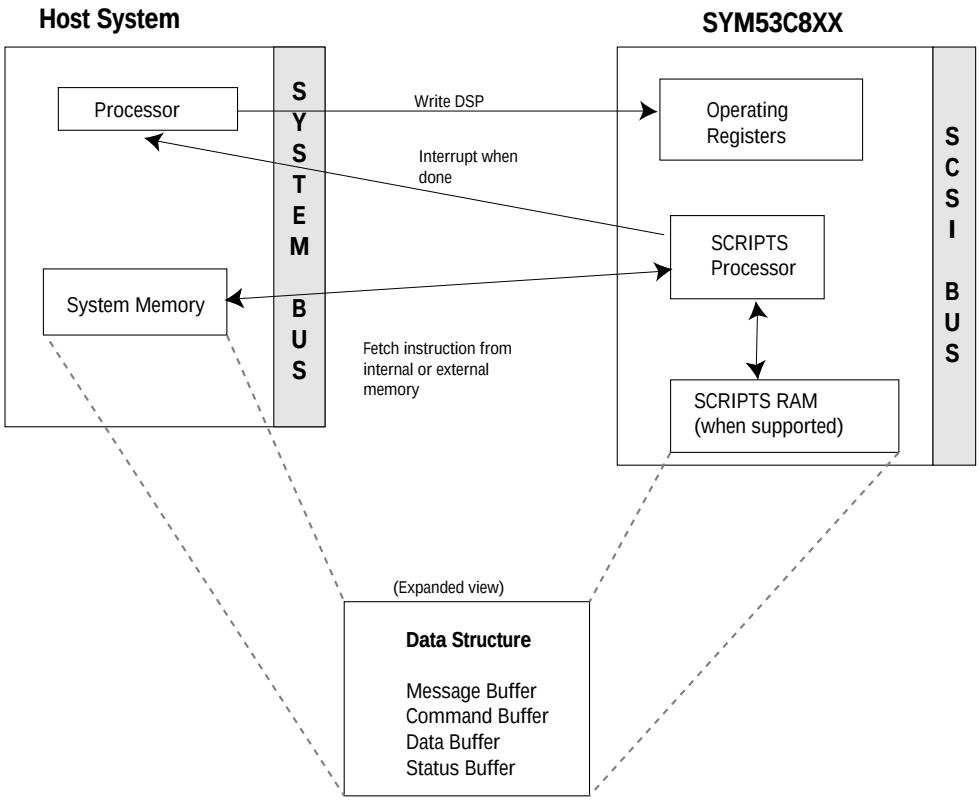


Figure 1-2
Typical SCRIPTS Operation

Conventions

The following types of notation are used in this programming guide to represent screen displays, command line entries, and variables in the code examples:

Table 1-2
Conventions Used in This Programming Guide

Item	Definition	Example
square braces []	optional items in instruction examples	`CALL [REL] Address, [{IF | WHEN} [NOT] CARRY]`
courier font	used for code samples, filenames, command line information, prompts, etc. that appear in body text	`program.exe`
All Caps	Keywords	`JUMP [REL] Address, [{IF | WHEN} [NOT] CARRY]`
Curly braces {}	choose between items enclosed in curly braces	`SELECT [ATN] {FROM Address | ID}, [REL] Address`
{} "..."	the item enclosed in the curly braces can be repeated as often as desired	`SET {ACK|ATN|TARGET|CARRY} [and {ACK|ATN|TARGET|CARRY}... ]`
|	OR, select one item from a list	`INTFLY int_value, [{IF | WHEN} [NOT] CARRY]`
\	line continuation	`RELATIVE baselabel \`

Chapter 2

Programming the SYM53C8XX With SCRIPTS

The SCRIPTS Processor

The advantages of SCSI SCRIPTS and the SCRIPTS processor can be utilized only with the SYM53C7XX and SYM53C8XX families of SCSI processors. The SCRIPTS processor is a specially designed processor, located in the SYM53C8XX, that permits instructions to be fetched from internal or external memory. Algorithms written in the SCSI SCRIPTS language are assembled to control the SCSI and DMA modules. Complex SCSI bus sequences, including multiple SCRIPTS instructions, execute independently of the host processor.

SCSI SCRIPTS reside in host computer memory or internal SCRIPTS RAM during system operation, allowing for fast execution. If instructions reside in external memory, the SYM53C8XX chip fetches SCRIPTS programs from memory using bus master DMA transfers. If instructions reside in SCRIPTS RAM, they are fetched directly from RAM without generating PCI bus traffic. The SCRIPTS processor allows users to fine tune SCSI operations such as adjusting to new device types, adapting to changes in SCSI logical definitions, or quickly incorporating new options (such as vendor unique commands or new SCSI specifications). The SCRIPTS processor fetches SCRIPTS instructions from system memory to control operation of the SYM53C8XX. The SCRIPTS processor does not compile code; SCRIPTS programs must be assembled for execution by the NASM assembler and then compiled with a standard “C” compiler as part of a “C” program. Third generation SCSI devices can be programmed with SCRIPTS using only a few hundred lines of SCRIPTS code. SCRIPTS are independent of the CPU, operating system, or system bus being used, so they are portable across platforms.

SCRIPTS and the SCSI Bus Phases

One important advantage of SCSI SCRIPTS is that the SCRIPTS language corresponds directly to SCSI protocol. In conjunction with the high level language syntax, it provides an excellent vehicle to master the complexity of SCSI. The one-to-one relationship between protocol phases and SCRIPTS instructions means that SCRIPTS can be customized to specific operations on the SCSI bus, and that SCSI software development is simplified by using SCRIPTS. SCSI uses the bus phases in the order shown in Table 2-1. This table also shows the SCSI SCRIPTS instructions that correspond to the SCSI bus phases for initiator and target roles.

Table 2-1
SCSI Protocol and SCRIPTS instructions

Bus Phase	Definition	SCRIPTS instruction (initiator role)	SCRIPTS instruction (target role)
Bus Free	This phase indicates that the SCSI bus is available.		
Arbitration	This phase allows the initiator to gain control of the SCSI bus.	SELECT ATN	RESELECT
Selection	During this phase, the initiator selects a target device to perform the desired function. The Attention option notifies the target that upon successful selection the initiator desires to send further messages.	SELECT ATN	WAIT SELECT
Reselection	The target reselects with the initiator during this phase	WAIT RESELECT	RESELECT
Message Out	During this phase, the initiator may send messages to the target, such as queuing information and error recovery information.	MOVE WHEN MSG_OUT	MOVE WITH MSG_OUT
Command	During this phase, the initiator may send a command in the form of a command descriptor block (CDB) to the target buffer.	MOVE WHEN CMD	MOVE WITH CMD

Table 2-1
SCSI Protocol and SCRIPTS instructions (Continued)

Bus Phase	Definition	SCRIPTS instruction (initiator role)	SCRIPTS instruction (target role)
Data In/Out	Data In and Data Out phases are used to send data to the initiator or to the target and are used dependent on the information transferred during the Command phase. This phase is optional. For example, a Test Unit Ready command does not require a data transfer.	MOVE	MOVE
Status	During this phase, the initiator will receive status information from the target about the previously executed CDB.	MOVE WHEN STATUS	MOVE WITH STATUS
Message In	During this phase, the initiator will receive messages from the target. These messages can acknowledge or reject previously sent initiator messages. They also can provide other information like queuing, disconnect, or parity errors.	MOVE WHEN MSG_IN	MOVE WITH MSG_IN
Disconnect	This phase is used to end the initiator's connection with the bus.	WAIT DISCONNECT	DISCONNECT
	After successful completion of an I/O operation and a request for disconnect, the bus returns to the Bus Free state, indicating that it is now available.	WAIT DISCONNECT	DISCONNECT

Assembling SCSI SCRIPTS

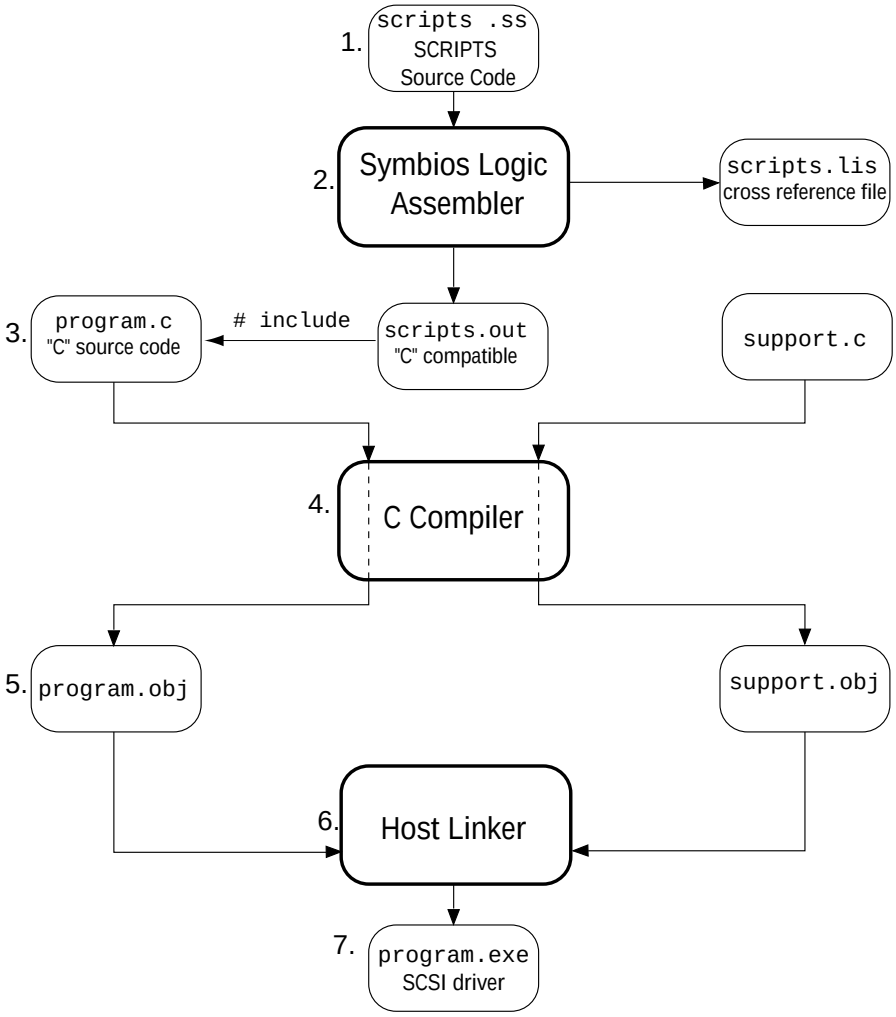
SCRIPTS are assembled with the Symbios Logic Assembler (NASM™), a DOS command line-driven assembler that supports Symbios Logic SCSI processors (SYM53C7XX and SYM53C8XX). NASM assembles SCSI SCRIPTS for inclusion in SCSI device driver software programs. NASM is described in detail in Chapter 4.

SCSI SCRIPTS programs can be created with any text editor that generates ASCII files. These source files must be transformed from their text form into the SCRIPTS processor's instruction language before they can be executed by the SYM53C8XX. This is

accomplished by running NASM. NASM generates an output file (`.out`) that is compatible with all standard "C" compilers, as well as a cross-reference list (`.lis`) file that includes the source instruction and the assembled output on the same line. The `.lis` file is useful for debugging code. All instructions and data are represented as hexadecimal numbers in C-style array declarations. The `.out` file can be included in the "C" program and linked together with other system support object files to form the final executable code.

When the executable is run, areas of host memory are reserved for SCSI data transfer buffers and the SCRIPTS instructions. The instructions, which look like 32-bit integer arrays to the "C" program, are loaded into the appropriate area of memory by the "C" code. The driver program loads the address of the first instruction into the SYM53C8XX to begin SCRIPTS execution.

Figure 2-1
Assembling SCSI SCRIPTS



1. Write SCSI SCRIPTS source code
2. Assemble the source code using the Symbios Logic Assembler
3. Write “C” language source code and include assembled SCRIPTS code
4. Compile all code using a “C” compiler
5. The result is object (.obj) code
6. Link all object modules together
7. The result is an executable program

Using SCSI SCRIPTS

SCRIPTS Data Sizes

Address	a 32-bit number
Value	a 32-bit number
Count	a 24-bit number
Data	an 8-bit number
ID	a 4-bit encoded SCSI ID

SCSI SCRIPTS Language Elements

name — A name is a string of one or more consecutive characters. It may consist of letters, numbers, underscores, and dollar signs, but must begin with an alphabetic character. When used for labels, externals, and variables in the relative data area, names are passed on to the host development system and are subject to the host's syntactic restrictions. Names cannot be reserved words in the host language. For example, Turbo C, which is used as the host development system for NASM, does not allow names to begin with a digit or to contain a dollar sign ($). Therefore, the SCSI SCRIPTS writer for DOS and Turbo C should avoid names of this form.

label — A label is a name followed by a colon. Labels are symbolic addresses that can be used as transfer control destination points (such as jump or call destinations). Labels are case-sensitive.

comment — Comments are used to notate the SCRIPTS. They are optional and are ignored by the compiler. Comments begin with a semi-colon and continue to the end of a line.

SCSI SCRIPTS Expressions

Arithmetic Operators

Symbol	Meaning
+	addition
-	subtraction

Bitwise Operators

Symbol	Meaning
&	Logical AND
|	Logical OR
XOR	Exclusive OR
SHL	Shift left
SHR	Shift right

The value of all expressions is automatically extended to 32 bits. When expressions are used in a context where the evaluated value is

less than 32 bits, the least significant bits will be used. For example, if an expression is used to represent a count, normally 24 bits, for a Move instruction, the evaluated value will be truncated to 24 bits. The user will be notified if the expression has been truncated and if the value of the expression is changed during truncation. The symbols for the bitwise operators are used only for register manipulations. Any other instruction using comparison must spell out AND or OR.

SCSI SCRIPTS Keywords

The SCSI SCRIPTS keywords have eight types: Declarative, Conditional, Logical, Flag Field, Qualifier, Action, SCSI Phase, and Register Name. Keywords are written in all capital letters for clarity, but are not case-sensitive. Refer to Chapter 4 for detailed descriptions of individual keywords.

Description of SCRIPTS Instructions

This section contains an overview of the types of instructions supported by SCRIPTS. Each instruction, including all legal forms, is described in detail in Chapter 3.

I/O Instructions

The I/O Instruction type is selected when the two high order bits of the DCMD register are 01, with op code bit values of 000-100. I/O Instructions perform SCSI operations such as Selection and Reselection. Each function is a direct command to the SCSI portion of the SYM53C8XX. The I/O operations, chosen with the op code bits in the DCMD register, are:

Op Code	Target	Initiator
000	RESELECT	SELECT, SELECT WITH ATN
001	DISCONNECT	WAIT FOR DISCONNECT
010	WAIT FOR SELECT	WAIT FOR RESELECT
011	SET	SET
100	CLEAR	CLEAR

Memory Move Instructions

The Memory Move Instruction type is selected when the two high order bits of the DCMD register are 11. The Memory Move instruction allows you to transfer data from one 32-bit memory location to another. The source or the destination may be a chip register. A 24-bit byte counter allows large moves to occur with no intervention from the host processor. If both addresses are in system memory, the SYM53C8XX functions as a high-speed DMA controller, able to move data at sustained speeds up to 47 megabytes per second (MB/s) without using the host processor or its cache memory. Data is moved from the source address into the chip's DMA FIFO and then out to the destination address. This instruction type does not allow indirect addressing, so the physical 32-bit address must be in the SCRIPTS instruction.

In chips that support instruction prefetching, the NOFLUSH qualifier can be used to prevent the prefetch buffer from being flushed when the chip performs a Memory to Memory Move instruction.

Transfer Control Instructions

The Transfer Control instruction type is selected when the two high order bits of the DCMD register are 10. Transfer Control Instructions perform SCRIPTS operations such as JUMP, CALL, RETURN, and INTERRUPT. These instructions allow comparisons of current phase values on the SCSI bus or the first byte of data on any asynchronous incoming bytes, and transfer control to another address depending on the results of the comparison test. These operations may conduct a test of the ALU carry bit, and may enable interrupt on the fly, so that the interrupt instruction will not halt the SCRIPTS processor.

Read/Write Instructions

Read/Write Instructions perform the following register operations:

Move from SFBR	Moves the SCSI First Byte Received (SFBR) register (08h) to a specified register address
Move to SFBR	Moves a specified register value to the SFBR register
Read/Modify/Write	Reads a specified register, modifies it, and writes the result back into the same register

The Read/Write Instruction type is selected when the two high order bits of the DCMD register are 01, with the op code bit values from 101-111. Read/Write Instructions perform various register operations, depending on the value of the operator bits as shown on page 3- 35.

Block Move Instructions

The Block Move instruction type is selected when the two high order bits of the DCMD register are 00. The Block Move instruction transfers data (user data or SCSI information) to or from user memory from or to the SCSI bus. The data may come from any memory address, so scatter/gather operations for user data are transparent to the chip and the external processor. A separate Block Move instruction is written for each piece of data to be moved. This instruction allows indirect and table indirect addressing.

Load and Store Instructions

Load and Store instructions are available only in the SYM53C810A/53C860/53C825A/53C875. They are a more efficient way than the Memory Move instruction to move data directly to/from memory from/to an internal register because they have two dwords instead of three and require one PCI bus ownership instead of two. These instructions will move a maximum of four bytes. The Load/Store instruction type is selected when the three high order bits of the DCMD register are 111. The memory address may map to external memory space or to the SCRIPTS RAM.

Big and Little Endian Byte Addressing

The guidelines in this section will help assure proper byte lane ordering in Big or Little Endian designs. Please check the features list for each chip to determine which products support Big and/or Little Endian addressing.

Big Endian addressing is used primarily in designs based on Motorola processors. The SYM53C8XX treats D(31-24) as the lowest physical memory address. Little Endian is used primarily in designs based on Intel processors. This mode treats D(7-0) as the lowest physical memory address.

Table 2-2
Big and Little Endian
Byte Addressing

System data bus	(31-24)	(23-16)	(15-8)	(7-0)
53C8XX pins	31-24	23-16	15-8	7-0
Register	SCNTL3	SCNTL2	SCNTL1	SCNTL0
Little Endian addr	03h	02h	01h	00h
Big Endian addr	00h	01h	02h	03h

Order of SCRIPTS Instructions

To ensure that SCSI SCRIPTS instructions are in the correct order, each SCRIPTS routine must be compiled in the target architecture. The "C" output (.OUT) file lists arrays of dword (32-bit) values, which are stored in the memory by the processor and in the correct order for the subsequent execution. For a Little Endian SCRIPTS instruction to execute on a Big Endian machine, the bytes will need to be reversed before execution. A PROM cannot be moved from one environment to another without re-ordering bytes within each word. The best way to guarantee correct byte ordering is to make sure the SCRIPTS are placed in memory with the op code byte on the same byte lane as the DCMD register in the SYM53C8XX.

Operating Register Access from Firmware

To develop code that works in either mode, use equates for the register names with an endian switch specified at compile time to include the appropriate set of address values. Note that the change is only for byte access. If 32 bits are accessed, there is no address change from Big to Little Endian.

Operating Register Access from SCRIPTS Routines

NASM uses logical names to access registers. Names do not change when the mode changes, nor does the binary code required to access a register.

User Data Byte Ordering

Data transfers to or from system memory from or to the SCSI bus always start at the beginning address and continue until the last byte is sent. No internal re-ordering of the data for either mode occurs. A serial stream of data is assumed, and the first byte on the SCSI bus is associated with the lowest address in system memory, regardless of Big or Little Endian.

Chapter 3

The SYM53C8XX Instruction Set

Overview

This section describes the SYM53C8XX SCSI I/O Processor instruction set. Additional information may be found in the SYM53C8XX product data manuals. The first section of this chapter contains an alphabetical list of all SCSI instructions. Each instruction is presented with a detailed description and usage guidelines. The second section of the chapter presents illustrations of how all of the instruction types are expressed in SCSI SCRIPTS language, NASM output, and the binary form that is executed by the SCSI processor. The SYM53C8XX Family supports the following SCRIPTS instructions, grouped by instruction type. The individual instruction entries list the SYM53C8XX family members that support each instruction.

Table 3-1
SCRIPTS Instructions Supported by the SYM53C8XX Family

Instruction Type	Commands
I/O	RESELECT, SELECT, SELECT WITH ATN, DISCONNECT, WAIT DISCONNECT, WAIT SELECT, WAIT RESELECT, SET, CLEAR
Memory Move	MOVE MEMORY
Transfer Control	JUMP, CALL, RETURN, INTERRUPT, INTFLY
Read/Write	MOVE REGISTER
Block Move	MOVE, CHMOV
Load/Store	LOAD, STORE

CALL

```
CALL {REL(Address) | Address} [, {IF | WHEN}[NOT][ATN | Phase] [AND
| OR] [data[AND MASK data]]]

CALL {REL(Address) | Address} [, {IF | WHEN}[NOT][Carry]
```

Supported by:

All Symbios Logic PCI-SCSI I/O Processors

Definition:

SCSI Transfer Control, Call subroutine

Operands:

REL indicates the use of relative addressing by setting the high order bit in the DBC register.

Address is the location to which execution will be transferred if the subroutine is called. This address is stored in the second dword of the instruction.

WHEN forces the SCRIPTS engine to wait for a valid SCSI bus phase before continuing. A valid phase is indicated by assertion of the SREQ/ signal.

IF causes the SYM53C8XX to check immediately for a valid SCSI bus phase without waiting. IF should not be used when comparing for a phase, as this could yield unpredictable results. The only exception is if a WHEN conditional was used just prior to the IF conditional, for any given sequence of phase checks.

NOT negates the comparison. It clears the True bit if present, otherwise the True bit is set.

Phase is used to specify the Message, Command/Data, and Input/Output bit values that identify the SCSI phase in the instruction. The desired phase value is compared with the actual values of the SCSI phase lines before the SYM53C8XX performs the instruction. This field is only valid for initiator mode and should not be used in target mode.

ATN is used to indicate that a jump should take place based on an initiator SATN/ signal. This field is valid only for target mode and should not be used in initiator mode.

data when this field is present it represents an 8-bit value that is stored in the data field of the instruction. In addition the Compare Data bit is set.

MASK when this field is present it represents an 8-bit value that is stored in the mask field of the instruction. Any bit that is set in the mask causes the corresponding bit in the data byte to be ignored at the time of the comparison.

CARRY is used to indicate that a jump should take place based on the value of the carry bit in the ALU. Carry comparisons cannot take place at the same time as data and phase comparisons.

Example:

```
CALL REL (Address), WHEN DATA_OUT
```

Format:

DCMD Register			DBC Register										DSPS Register
31 30	29 27	26 24	23	22	21	20	19	18	17	16	15 8	7 0	31 0
10	001	XXX	X	0	0	0	X	X	X	X	X...X	X..X	Call Address
Instr Type	Op code	SCSI Phase	Rel Addr Mode	RES	Carry Test	RES	True	Comp Data	Comp Phase	Wait	Mask	Data	Call Address or offset

Fields:

Op code - Transfer Control Instruction, Call subroutine

SCSI Phase - These bits reflect the actual values of the SCSI phase lines. The bit values are defined below.

Phase	Message	Command / Data	Input / Output
DATA_OUT	0	0	0
DATA_IN	0	0	1
COMMAND	0	1	0
STATUS	0	1	1
RES4	1	0	0
RES5	1	0	1
MESSAGE_OUT	1	1	0
MESSAGE_IN	1	1	1

* Note: 0 - False, negated; 1 - True, asserted, For these phases, SEL is negated and BSY is asserted.

* Res4 and Res5 are reserved SCSI phases. These combinations should not be used for standard SCSI implementations

Relative Addr Mode - Relative Addressing Mode indicates that the 24-bit value in DSPS is to be used as an offset from DSP

Carry Test - When this bit is set, True/False comparisons may be made based on the ALU Carry bit

True - Transfer on TRUE/FALSE condition
0 - Transfer if condition is FALSE
1 - Transfer if condition is TRUE

Compare Data - Compare data byte to first byte of the received data
0 - Do not compare data
1 - Perform comparison

Compare Phase - Compare current SCSI phase to SCSI phase field or SATN/. This bit is set whenever the Phase operand is used.
0 - Do not compare phase
1 - Perform comparison

Wait - Wait for valid phase. This bit is set by the WHEN operand in the instruction, and cleared by the IF operand.
0 - Perform comparison immediately
1 - Wait for valid phase (SREQ/ asserted by target)

Mask - 8-bit field that is used to mask the value in SFBR before the comparison with the data field in the instruction takes place. As a result of this operation, any bits that are set will cause the corresponding bit in the data byte to be ignored. If this field is not specified, a mask of 0x00 is used.

Data - 8-bit field that is compared with the incoming data in SFBR after the mask operation with the mask byte takes place. Comparison indicates either an equal or not equal condition. If the Data field is not specified, the compare data bit is cleared and 0x00 is coded for both the mask and data bytes.

Call Addr - a 32-bit address (or 24-bit offset, if relative addressing is used) where execution will continue if the subroutine is called.

Description:

The SCSI CALL instruction is a conditional subroutine call that causes the next SCRIPTS instruction to be fetched from memory at the 32-bit call address (or 24- bit offset). It is invoked if all conditions in the instruction or data are met. If the comparison is false, the SCRIPTS processor will not branch to the destination but will instead fetch the next in-line instruction and continue execution. If the subroutine is called, the next in-line instruction address is stored in the chip's TEMP register, and will be restored to the DSP register in response to a RETURN instruction following the CALL.

When the optional *data* field is used, it is compared to the first byte of the most recent asynchronous data, message, command, or status byte received. The user's SCSI SCRIPTS program can determine which routine to execute next based on actual data values received. Using a series of these compares, the algorithm can process complex sequences with no intervention required by the external processor.

When the optional *MASK* keyword and its associated value are specified, the SCRIPTS processor allows selective comparisons of bits within the data byte. This comparison is illustrated in Figure 3-1. During the comparison, any bits that are set in the mask data will cause the corresponding bit in the data byte to be ignored for the comparison.

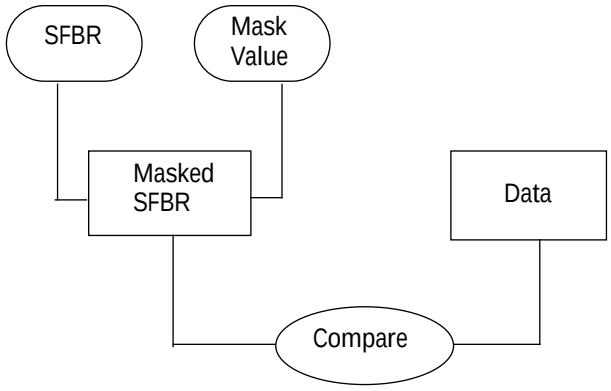


Figure 3-1
Use of the Mask Keyword

Notes:

SCRIPTS does not directly support nested CALLs. If two CALL instructions are issued without any intervening RETURN instruction, then the first return address in the chip's TEMP register is overwritten by the second CALL and lost.

The REL keyword, which indicates relative addressing, is unrelated to the declarative keyword RELATIVE that establishes relative buffers.

Legal Forms:

```
CALL address
CALL address, IF ATN
CALL address, IF Phase
CALL address, IF CARRY
CALL address, IF data
CALL address, IF data AND MASK data
CALL address, IF ATN AND data
CALL address, IF ATN AND data AND MASK data
CALL address, IF Phase AND data
CALL address, IF Phase AND data AND MASK data
CALL address, WHEN Phase
CALL address, WHEN CARRY
CALL address, WHEN data
CALL address, WHEN data AND MASK data
CALL address, WHEN Phase AND data
CALL address, WHEN Phase AND data AND MASK data
CALL address, IF NOT ATN
CALL address, IF NOT Phase
CALL address, IF NOT CARRY
CALL address, IF NOT data
CALL address, IF NOT data AND MASK data
```

```
CALL address, IF NOT ATN OR data
CALL address, IF NOT ATN OR data AND MASK data
CALL address, IF NOT Phase OR data
CALL address, IF NOT Phase OR data AND MASK data
CALL address, WHEN NOT Phase
CALL address, WHEN NOT CARRY
CALL address, WHEN NOT data
CALL address, WHEN NOT data AND MASK data
CALL address, WHEN NOT Phase OR data
CALL address, WHEN NOT Phase OR data AND MASK data
CALL REL(address)
CALL REL(address), IF ATN
CALL REL(address), IF Phase
CALL REL(address), IF CARRY
CALL REL(address), IF data
CALL REL(address), IF data AND MASK data
CALL REL(address), IF ATN AND data
CALL REL(address), IF ATN AND data AND MASK data
CALL REL(address), IF Phase AND data
CALL REL(address), IF Phase AND data AND MASK data
CALL REL(address), WHEN Phase
CALL REL(address), WHEN CARRY
CALL REL(address), WHEN data
CALL REL(address), WHEN data AND MASK data
CALL REL(address), WHEN Phase AND data
CALL REL(address), WHEN Phase AND data AND MASK data
CALL REL(address), IF NOT ATN
CALL REL(address), IF NOT Phase
CALL REL(address), IF NOT CARRY
CALL REL(address), IF NOT data
CALL REL(address), IF NOT data AND MASK data
CALL REL(address), IF NOT ATN OR data
CALL REL(address), IF NOT ATN OR data AND MASK data
CALL REL(address), IF NOT Phase OR data
CALL REL(address), IF NOT Phase OR data AND MASK data
CALL REL(address), WHEN NOT Phase
CALL REL(address), WHEN NOT CARRY
CALL REL(address), WHEN NOT data
CALL REL(address), WHEN NOT data AND MASK data
CALL REL(address), WHEN NOT Phase OR data
CALL REL(address), WHEN NOT Phase OR data AND MASK data
```

CHMOV

```
CHMOV {FROM | count,} [PTR] address,{WITH | WHEN} phase
```

Supported by: SYM53C825A, SYM53C875, SYM53C885, SYM53C876, SYM53C895

Definition: Wide SCSI Block Move

Operands: **FROM** indicates table indirect addressing mode

Note: FROM and PTR must not be used in the same instruction.

count is the number of bytes to transfer across the SCSI bus.

PTR sets the indirect bit if present, it is clear otherwise.

Note: FROM and PTR must not be used in the same instruction.

address is the 32-bit starting address of the data in memory, unless PTR is present. If PTR is present, **address** represents the location of the starting address.

WITH/WHEN set the mode for the device; WITH for target mode and WHEN for initiator mode.

Phase is used to specify the Message, Command/Data, and Input/Output bit values that identify the SCSI phase in the instruction. The desired phase value is compared with the actual values of the SCSI phase lines before the SYM53C8XX performs the instruction. This field is only valid for initiator mode and should not be used in target mode.

Example:

```
CHMOV FROM dev_1 WITH Data_In
CHMOV 6, data_buf, WHEN Data_Out
```

Format:

DCMD Register					DBC Register	DSPS register
31 30	29	28	27	26 24	23 0	31 0
00	X	X	X	XXX	XX... XX	XX... XX
Instruction type	Indirect	Table Indirect	Op code	SCSI Phase	Byte Count	Dest Addr

Fields:

Instruction type - 00 = Block Move

Indirect - Indirect Addressing Mode
0 - Use destination field as an address
1 - Use destination field as an address to an address

Table Indirect - Table Indirect Addressing Mode
0 - Use Absolute addressing mode
1 - Use destination address as offset from the value of DSA register.

Op code- Defines whether the instruction will be executed as a Block Move or a Chained Block Move. This bit value has different meanings, depending on whether the chip is operating in target or initiator role.

	Target	Initiator
MOVE	Op code = 0	Op code = 1
CHMOV	Op code = 1	Op code = 0

SCSI Phase - These bits reflect the actual values of the SCSI phase lines. The bit values are defined below.

Phase	Message	Command / Data	Input / Output
DATA_OUT	0	0	0
DATA_IN	0	0	1
COMMAND	0	1	0
STATUS	0	1	1
RES4	1	0	0
RES5	1	0	1
MESSAGE_OUT	1	1	0
MESSAGE_IN	1	1	1
* Note: 0 - False, negated; 1 - True, asserted. For these phases, SEL is negated and BSY is asserted. * Res4 and Res5 are reserved SCSI phases. These combinations should not be used for standard SCSI implementations			

Byte Count - 24-bit number indicating the number of bytes to transfer.

Dest Addr - Address to perform data transfer on, or offset from the DSA to fetch table indirect information.

Description:

There are various forms of the Chained Block Move instruction. The “address” and “count” specify the address and byte count fields of the instruction. If the optional keyword “PTR” is present, then the indirect bit will be set. If PTR is present, the address specified in the instruction is the address of the pointer to the data in memory. “Phase” specifies the phase field of the instruction. WITH or WHEN are used to specify the Block Move function codes. WITH is used to signal the target role which sets the phase values, and WHEN is the initiator “test for phase” feature.

The 53C8XX waits for a valid phase (initiator) or drives the phase lines (target). In the initiator role, it performs a comparison looking for a match between the phase specified in the SCRIPTS instruction and the actual value on the bus. If the phases do not match, an external interrupt occurs. A test prior to the Move instruction could be used to avoid this interrupt. If the phase does match, data is then transferred in or out according to the phase lines. When the count

goes to zero, the SYM53C8XX fetches the next sequential SCRIPTS instruction.

The Chained Move instruction transfers data to and from memory locations. Data may come from **any** data location, so scatter/gather operations are transparent to the chip and external processor.

When the SYM53C8XX executes several CHMOV instructions and the ends are on an **odd** byte boundary, the chip temporarily stores the residual byte in the SODL register (send operations) or SWIDE register (receive operations). The SYM53C8XX takes the first byte from the subsequent CHMOV or MOVE instruction and lines it up with the residual byte in order to complete a **wide** transfer and maintain a continuous wide data flow on the SCSI bus.

For more information on Chained Block Move Instructions, please see the appropriate SYM53C8XX data manual.

Notes:

Legal Forms:

```
CHMOV count, address, WITH phase
CHMOV count, address, WHEN phase
CHMOV count, PTR address, WITH phase
CHMOV count, PTR address, WHEN phase
CHMOV FROM address, WITH phase
CHMOV FROM address, WHEN phase
```

CLEAR

```
CLEAR {ACK | ATN | TARGET | CARRY} [and{ACK | ATN | TARGET | CARRY}
... ]
```

Supported by: All Symbios Logic PCI-SCSI I/O Processors

Definition: Deasserts SCSI ACK or ATN, or clears internal flags

Operands:
ACK - clears the Assert SCSI ACK bit
ATN - clears the Assert SCSI ATN bit
TARGET - clears the Set Target role bit
CARRY - clears the CARRY bit in the ALU

Examples:
```
CLEAR TARGET
CLEAR ACK and TARGET
```

Format:

DCMD Register			DBC Register								DSPS Register
31 30	29 27	26 24	23 11	10	9	8 7	6	5 4	3	2 0	31 0
01	100	000	0. 0	X	X	0 0	X	00	X	000	00...00
Instr Type	Op code	RES	RES	Set/ Clear Carry	Set/ Clear Target	RES	Assert SCSI ACK	RES	Assert SCSI ATN	RES	RES

Fields:

Instruction Type - I/O

Op code - Clear instruction

Set/Clear Carry
1 - clears the Carry bit in the ALU
0 - has no effect

Set /Clear Target Mode
1 - places the chip into initiator mode
0 - has no effect

Set/Clear SCSI ACK
1 - deasserts the SCSI acknowledge signal
0 - has no effect

Set/Clear SCSI ATN
1 - deasserts the SCSI attention signal
0 - has no effect

Description: The chip deasserts the signals indicated in the instruction. Currently four bits are defined, allowing the SCSI SACK, target role, and SATN bits to be cleared as well as the CARRY bit in the ALU. Bit 10 is for CARRY, bit 9 is for target, bit 6 is for Acknowledge, and bit 3 is for Attention.

Notes:

Legal Forms:

```
CLEAR ACK
CLEAR ATN
CLEAR TARGET
CLEAR CARRY
CLEAR ACK and ATN
CLEAR ACK and TARGET
CLEAR ACK and CARRY
CLEAR ATN and TARGET
CLEAR ATN and CARRY
CLEAR TARGET and CARRY
CLEAR ACK and ATN and TARGET
CLEAR ACK and ATN and CARRY
CLEAR ACK and ATN and TARGET and CARRY
```

DISCONNECT

```
DISCONNECT
```

Supported by: All Symbios Logic PCI-SCSI I/O Processors

Definition: Perform disconnect

Operands: None

Example:
```
DISCONNECT
```

Format:

DCMD Register			DBC Register	DSPS Register
31 30	29 25	24	23 0	31 0
01	00100	0	0000.....000	00.....00
Instr Type	Op Code	RES	Reserved	Reserved

Fields: **Instruction Type** - I/O

Op Code—Disconnect instruction

Description: The DISCONNECT instruction causes the chip (when in target role) to physically disconnect from the bus.

Notes: This instruction has no effect on the initiator if it is issued by a target. To disconnect from the SCSI bus, use the SET TARGET instruction before this instruction.

Legal Forms:
```
DISCONNECT
```

INT

```
INT int_value [, {IF | WHEN}[NOT][ATN | Phase][AND | OR] [data[AN
   MASK data]]]
INT int_value [, {IF | WHEN}[NOT] CARRY]
```

Supported by: All Symbios Logic PCI-SCSI I/O Processors

Definition: SCSI Transfer Control - Generate Interrupt and halt SCRIPTS operation

Operands: **int_value** is a user defined 32-bit value that will be available in the DSPS register at the time of the interrupt.

WHEN forces the SCRIPTS engine to wait for a valid SCSI bus phase before continuing. A valid phase is indicated by assertion of the SREQ/ signal.

IF causes the SYM53C8XX to check immediately for a valid SCSI bus phase without waiting. IF should not be used when comparing for a phase, as this could yield unpredictable results. The only exception is if a WHEN conditional was used just prior to the IF conditional, for any given sequence of phase checks.

NOT negates the comparison. It clears the True bit if present, otherwise the True bit is set.

Phase is used to specify the Message, Command/Data, and Input/Output bit values that identify the SCSI phase in the instruction. The desired phase value is compared with the actual values of the SCSI phase lines before the SYM53C8XX performs the instruction. This field is only valid for initiator mode and should not be used in target mode.

ATN indicates that an interrupt should take place based on an initiator SATN/ signal. This field is valid only for target mode and should not be used in initiator mode.

data represents an 8-bit value that is stored in the data field of the instruction. In addition the Compare Data bit is set.

MASK represents an 8-bit value that is stored in the mask field of the instruction. Any bit that is set in the mask causes the corresponding bit in the data byte to be ignored at the time of the comparison.

CARRY indicates that an interrupt should take place based on the value of the carry bit in the ALU. Carry comparisons cannot take place at the same time as data and phase comparisons.

Example:

```
INT 0x00000001, WHEN NOT COMMAND
INT 0x200010F7, IF 0xF8 AND MASK 0x07
```

Format:

DCMD Register			DBC Register								DSPS Register
31 30 29 27		26 24	23 22	21 20		19	18	17	16	15 8	7 0	31 0
10	011	XXX	0 0	0	0	X	X	X	X	X...X	X..X	X...X
Instr Type	Op code	SCSI Phase	RES	Carry Test	RES	True	Comp Data	Comp Phase	Wait	Mask	Data	int_value

Fields:

Instruction Type - Transfer Control**.**

Op code - Interrupt Instruction.

SCSI Phase - These bits reflect the actual values of the SCSI phase lines. The bit values are defined below.

Phase	Message	Command / Data	Input / Output
DATA_OUT	0	0	0
DATA_IN	0	0	1
COMMAND	0	1	0
STATUS	0	1	1
RES4	1	0	0
RES5	1	0	1
MESSAGE_OUT	1	1	0
MESSAGE_IN	1	1	1

* Note: 0 - False, negated; 1 - True, asserted. For these phases, SEL is negated and BSY is asserted.

* Res4 and Res5 are reserved SCSI phases. These combinations should not be used for standard SCSI implementations

Carry Test - When this bit is set, true/false comparisons may be made based on the ALU Carry bit. Carry comparisons cannot be made at the same time as data and phase comparisons.

True - Transfer on TRUE/FALSE condition
0 - Transfer if condition is FALSE
1 - Transfer if condition is TRUE

Compare Data - Compare data byte to first byte of the received data.
0 - Do not compare data
1 - Perform comparison

Compare Phase - Compare current SCSI phase to SCSI phase field or SATN/. This bit is set whenever the Phase operand is used.
0 - Do not compare phase
1 - Perform comparison

Wait - Wait for valid phase. This bit is set by the WHEN operand in the instruction, and cleared by the IF operand.
0 - Perform comparison immediately
1 - Wait for valid phase (SREQ/ asserted by target)

Mask - 8-bit field that is used to mask the value in SFBR before the comparison with the data field in the instruction takes place. As a result of this operation, any bits that are set will cause the corresponding bit in the data byte to be ignored. If this field is not specified, a mask of 0x00 is used.

Data - 8-bit field that is compared with the incoming data after the mask operation with the mask byte takes place. Comparison indicates either an equal or not equal condition. If the Data field is not specified, the compare data bit is cleared and 0x00 is coded for both the mask and data bytes.

Int_Value - a 32-bit user defined value that is available to the external processor to identify the cause of the interrupt. If the interrupt conditions are met, the int_value will be available in the DSPS register for the processor to use to determine the cause of the interrupt.

Description:

The SCSI Interrupt instruction causes the chip to conditionally halt execution and post an interrupt request to the external processor. It is used if the SCSI phase, data, or attention condition compares true with the phase, data, or attention condition described in the instruction. The NOT qualifier is used for the comparison to determine a boolean true/false outcome of the comparison. If the comparison is false, the SCRIPTS processor will not post the interrupt but will instead fetch the next in-line instruction and continue execution.

When the optional *data* field is used, it is compared to the first byte of the SFBR. This contains the most recent byte of any kind of data that has been moved into the SFBR register. The user's SCSI SCRIPTS program can determine which routine to execute next based on actual data values received. Using a series of these compares, the algorithm can process complex sequences with no intervention required by the external processor.

When the optional *MASK* keyword and its associated value are specified the SCRIPTS processor allows selective comparisons of bits within the data byte. This comparison is illustrated in Figure 3-1. During the comparison, any bits that are set in the mask byte will cause the corresponding bit in the data byte to be ignored for the comparison.

Notes:

Legal Forms:

```
INT int_value
INT int_value, IF ATN
INT int_value, IF Phase
INT int_value, IF CARRY
INT int_value, IF data
INT int_value, IF data AND MASK data
INT int_value, IF ATN AND data
INT int_value, IF ATN AND data AND MASK data
INT int_value, IF Phase AND data
INT int_value, IF Phase AND data AND MASK data
INT int_value, WHEN Phase
INT int_value, WHEN CARRY
INT int_value, WHEN data
INT int_value, WHEN data AND MASK data
INT int_value, WHEN Phase AND data
INT int_value, WHEN Phase AND data AND MASK data
INT int_value, IF NOT ATN
INT int_value, IF NOT Phase
INT int_value, IF NOT CARRY
INT int_value, IF NOT data
INT int_value, IF NOT data AND MASK data
INT int_value, IF NOT ATN OR data
INT int_value, IF NOT ATN OR data AND MASK data
INT int_value, IF NOT Phase OR data
INT int_value, IF NOT Phase OR data AND MASK data
INT int_value, WHEN NOT Phase
INT int_value, WHEN NOT CARRY
INT int_value, WHEN NOT data
INT int_value, WHEN NOT data AND MASK data
INT int_value, WHEN NOT Phase OR data
INT int_value, WHEN NOT Phase OR data AND MASK data
```

INTFLY

```
INTFLY [int_value] [, {IF | WHEN}[NOT][ATN | Phase] [AND | OR]
[data[AND MASK data]]]
INTFLY [int_value] [, {IF | WHEN}[NOT] CARRY]
```

Supported by: All SYM53C8XX PCI-SCSI I/O Processors

Definition: Generate Interrupt and Continue SCRIPTS Execution

Operands: **int_value** is a user defined 32-bit value that is written to the DSPS register at the time of the interrupt. However, as stated in the Note below, since the processor continues to execute, the value is immediately overwritten with the next instruction fetch. Refer to the Note at the end of this section for more information.

WHEN forces the SCRIPTS engine to wait for a valid SCSI bus phase before continuing. A valid phase is indicated by assertion of the SREQ/ signal.

IF causes the SYM53C8XX to check immediately for a valid SCSI bus phase without waiting. IF should not be used when comparing for a phase, as this could yield unpredictable results. The only exception is if a WHEN conditional was used just prior to the IF conditional, for any given sequence of phase checks.

NOT negates the comparison. It clears the True bit if present, otherwise the True bit is set.

Phase is used to specify the Message, Command/Data, and Input/Output bit values that identify the SCSI phase in the instruction. The desired phase value is compared with the actual values of the SCSI phase lines before the SYM53C8XX performs the instruction. This field is only valid for initiator mode and should not be used in target mode.

ATN indicates that an interrupt should take place based on the state of the initiator SATN/ signal. This field is valid only for target mode and should not be used in initiator mode.

data represents an 8-bit value that is stored in the data field of the instruction. In addition the Compare Data bit is set.

MASK represents an 8-bit value that is stored in the mask field of the instruction. Any bit that is set in the mask causes the corresponding bit in the data byte to be ignored at the time of the comparison.

CARRY indicates that a jump should take place based on the value of the carry bit in the ALU. Carry comparisons cannot be made in the same instruction as data or phase comparisons.

Example:

```
INTFLY 0x00000001, WHEN NOT COMMAND
INTFLY 0x200010F7, IF 0xF8 AND MASK 0x07
```

Format:

DCMD Register			DBC Register										DSPS Register
31 30 29 27		26 24	23	22	21 20		19	18	17	16	15 8	7 0	31 0
10	011	XXX	0	0	0	1	X	X	X	X	X...X	X..X	X...X
Instr Type	Op code	SCSI Phase	RES	RES	Carry Test	Int on Fly	True	Comp Data	Comp Phase	Wait	Mask	Data	Int_ Value

Fields:

Instruction Type - Transfer Control

Op code - Interrupt on the Fly Instruction

SCSI Phase - These bits reflect the actual values of the SCSI phase lines. The bit values are defined below.

Phase	Message	Command / Data	Input / Output
DATA_OUT	0	0	0
DATA_IN	0	0	1
COMMAND	0	1	0
STATUS	0	1	1
RES4	1	0	0
RES5	1	0	1
MESSAGE_OUT	1	1	0
MESSAGE_IN	1	1	1

* Note: 0 - False, negated; 1 - True, asserted. For these phases, SEL is negated and BSY is asserted.

* Res4 and Res5 are reserved SCSI phases. These combinations should not be used for standard SCSI implementations

Carry Test- When this bit is set, true/false comparisons may be made based on the ALU Carry bit. Carry comparisons cannot be made in the same instruction as data or phase comparisons.

Int on Fly - When this bit is set, the Interrupt instruction will not halt the SCRIPTS processor.

True - Transfer on TRUE/FALSE condition
0 - Transfer if condition is FALSE
1 - Transfer if condition is TRUE

Compare Data - Compare data byte to first byte of the received data.
0 - Do not compare data
1 - Perform comparison

Compare Phase - Compare current SCSI phase to SCSI phase field or SATN. This bit is set whenever the Phase operand is used.
0 - Do not compare phase
1 - Perform comparison

Wait - Wait for valid phase. This bit is set by the WHEN operand in the instruction, and cleared by the IF operand.
0 - Perform comparison immediately
1 - Wait for valid phase (SREQ/ asserted by target)

Mask - 8-bit field that is used to mask the value in SFBR before the comparison with the data field in the instruction takes place. As a result of this operation, any bits that are set will cause the corresponding bit in the data byte to be ignored. If this field is not specified, a mask of 0x00 is used.

Data - 8-bit field that is compared with the incoming data after the mask operation with the mask byte takes place. Comparison indicates either an equal or not equal condition. If the Data field is not specified, the compare data bit is cleared and 0x00 is coded for both the mask and data bytes.

Int_Value - a 32-bit user defined value that identifies the cause of the interrupt. Even though the int_value is stored, since the processor continues to execute, it is immediately overwritten with the next instruction fetch. Refer to the Note at the end of this section for more information.

Description:

The SCSI Interrupt on-the-Fly instruction causes the chip to conditionally set the INTFLY bit in the ISTAT register and post an interrupt request to the external processor. It is invoked if the SCSI phase, data, or attention condition compares true with the phase, data, or attention condition described in the instruction.

The NOT qualifier is used to indicate a boolean true/false desired outcome of the comparison. If the comparison is false, the SCRIPTS processor will not post the interrupt but will instead fetch the next instruction and continue SCRIPTS execution.

When the optional *data* field is used, it is compared to the first byte of the SFBR. This contains the most recent byte of any kind of data that has been moved into the SFBR register. The user's SCSI SCRIPTS program can determine which routine to execute next based on actual data values received. Using a series of these compares, the algorithm can process complex sequences with no intervention required by the external processor.

When the optional *MASK* keyword and its associated value are specified the SCRIPTS processor allows selective comparisons of bits within the data byte. This comparison is illustrated in Figure 3-1. During the comparison, any bits that are set in the mask field will cause the corresponding bit in the data byte to be ignored for the comparison.

Notes:

Unlike the INT instruction, the INTFLY does not allow a driver program to make an inquiry to the chip for the `int_value`. Even though the `int_value` is stored, since the processor continues to

execute, it is immediately overwritten with the next instruction fetch. Users who want an accessible interrupt value can use the move memory instruction to store a user defined value to a known memory location before executing the INTFLY instruction.

Legal Forms:

```
INTFLY
INTFLY, IF ATN
INTFLY, IF Phase
INTFLY, IF CARRY
INTFLY, IF data
INTFLY, IF data AND MASK data
INTFLY, IF ATN AND data
INTFLY, IF ATN AND data AND MASK data
INTFLY, IF Phase AND data
INTFLY, IF Phase AND data AND MASK data
INTFLY, WHEN Phase
INTFLY, WHEN CARRY
INTFLY, WHEN data
INTFLY, WHEN data AND MASK data
INTFLY, WHEN Phase AND data
INTFLY, WHEN Phase AND data AND MASK data
INTFLY, IF NOT ATN
INTFLY, IF NOT Phase
INTFLY, IF NOT CARRY
INTFLY, IF NOT data
INTFLY, IF NOT data AND MASK data
INTFLY, IF NOT ATN OR data
INTFLY, IF NOT ATN OR data AND MASK data
INTFLY, IF NOT Phase OR data
INTFLY, IF NOT Phase OR data AND MASK data
INTFLY, WHEN NOT Phase
INTFLY, WHEN NOT CARRY
INTFLY, WHEN NOT data
INTFLY, WHEN NOT data AND MASK data
INTFLY, WHEN NOT Phase OR data
INTFLY, WHEN NOT Phase OR data AND MASK data
INTFLY int_value
INTFLY int_value, IF ATN
INTFLY int_value, IF Phase
INTFLY int_value, IF CARRY
INTFLY int_value, IF data
INTFLY int_value, IF data AND MASK data
INTFLY int_value, IF ATN AND data
INTFLY int_value, IF ATN AND data AND MASK data
INTFLY int_value, IF Phase AND data
INTFLY int_value, IF Phase AND data AND MASK data
INTFLY int_value, WHEN Phase
INTFLY int_value, WHEN CARRY
INTFLY int_value, WHEN data
```

```
INTFLY int_value, WHEN data AND MASK data
INTFLY int_value, WHEN Phase AND data
INTFLY int_value, WHEN Phase AND data AND MASK data
INTFLY int_value, IF NOT ATN
INTFLY int_value, IF NOT Phase
INTFLY int_value, IF NOT CARRY
INTFLY int_value, IF NOT data
INTFLY int_value, IF NOT data AND MASK data
INTFLY int_value, IF NOT ATN OR data
INTFLY int_value, IF NOT ATN OR data AND MASK data
INTFLY int_value, IF NOT Phase OR data
INTFLY int_value, IF NOT Phase OR data AND MASK data
INTFLY int_value, WHEN NOT Phase
INTFLY int_value, WHEN NOT CARRY
INTFLY int_value, WHEN NOT data
INTFLY int_value, WHEN NOT data AND MASK data
INTFLY int_value, WHEN NOT Phase OR data
INTFLY int_value, WHEN NOT Phase OR data AND MASK data
```

JUMP

```
JUMP {REL(Address) | Address} [,{IF | WHEN}[NOT][ATN | Phase] AND |
OR] [data[AND MASK data]]]
JUMP {[REL] (Address) | Address} [, {IF | WHEN}[NOT] CARRY]
```

Supported by: All Symbios Logic PCI-SCSI I/O Processors

Definition: SCSI Transfer Control - Jump

Operands: **REL** indicates the use of relative addressing.

Address is the location to which execution will be transferred if the subroutine is called. If REL is used, Address is the offset from the current DSP value.

WHEN forces the SCRIPTS engine to wait for a valid SCSI bus phase before continuing. A valid phase is indicated by assertion of the SREQ/ signal.

IF causes the SYM53C8XX to check immediately for a valid SCSI bus phase without waiting. IF should not be used when comparing for a phase, as this could yield unpredictable results. The only exception is if a WHEN conditional was used just prior to the IF conditional, for any given sequence of phase checks.

NOT negates the comparison. It clears the True bit if present, otherwise the True bit is set.

Phase is used to specify the Message, Command/Data, and Input/Output bit values that identify the SCSI phase in the instruction. The desired phase value is compared with the actual values of the SCSI phase lines before the SYM53C8XX performs the instruction. This field is only valid for initiator mode and should not be used in target mode.

ATN is used to indicate that a jump should take place based on the state of the initiator SATN signal. This field is valid only for target mode and should not be used in initiator mode.

data represents an 8-bit value that is stored in the data field of the instruction. In addition, this keyword indicates that the Compare Data bit is set.

MASK represents an 8-bit value that is stored in the mask field of the instruction. Any bit that is set in the mask causes the corresponding bit in the data byte to be ignored at the time of the comparison.

CARRY indicates that a jump should take place based on the value of the carry bit in the ALU.

Example:

```
JUMP Do_Next_Command WHEN COMMAND
JUMP Data_Check, IF DATA_IN AND 0x80 MASK 0x7F
```

Format:

DCMD Register			DBC Register										DSPS Register
31 30 29 27		26 24	23	22	21 20		19	18	17	16	15 8	7 0	31 0
10	000	XXX	X	0	0	0	X	X	X	X	X...X	X...X	X...X
Instr Type	Opcode	SCSI Phase	Rel Addr	RES	Carry Test	RES	True	Comp Data	Comp Phase	Wait	Mask	Data	Dest Addr

Fields:

Instruction Type - Transfer Control

Op code - Jump instruction

SCSI Phase - These bits reflect the actual values of the SCSI phase lines. The bit values are defined below.

Phase	Message	Command / Data	Input / Output
DATA_OUT	0	0	0
DATA_IN	0	0	1
COMMAND	0	1	0
STATUS	0	1	1
RES4	1	0	0
RES5	1	0	1
MESSAGE_OUT	1	1	0
MESSAGE_IN	1	1	1

* Note: 0 - False, negated; 1 - True, asserted. For these phases, SEL is negated and BSY is asserted.

* Res4 and Res5 are reserved SCSI phases. These combinations should not be used for standard SCSI implementations

Relative Addr - Relative Addressing Mode indicates that the 24-bit address value in the instruction is to be used as an offset from the current DSP address (which is pointing to the next instruction, not the one currently executing).

Carry Test - When this bit is set, true/false comparisons may be made based on the ALU Carry bit. Comparisons to the state of the Carry flag may not be made in conjunction with other comparisons.

True - Transfer on TRUE/FALSE condition
0 - Transfer if condition is FALSE
1 - Transfer if condition is TRUE

Compare Data - Compare data byte to first byte of the received data.
0 - Do not compare data
1 - Perform comparison

Compare Phase - Compare current SCSI phase to SCSI phase field or SATN/. This bit is set whenever the Phase operand is used.
0 - Do not compare phase
1 - Perform comparison

Wait - Wait for valid phase. This bit is set by the WHEN operand in the instruction, and cleared by the IF operand.
0 - Perform comparison immediately
1 - Wait for valid phase (SREQ/ asserted by target)

Mask - 8-bit field that is used to mask the value in SFBR before the comparison with the data field in the instruction takes place. As a result of this operation, any bits that are set will cause the corresponding bit in the data byte to be ignored. If this field is not specified, a mask of 0x00 is used.

Data - 8-bit field that is compared with the incoming data after the mask operation with the mask byte takes place. Comparison indicates either an equal or not equal condition. If the Data field is not specified, the Compare Data bit is cleared and 0x00 is coded for both the mask and data bytes.

Dest Addr - a 32-bit address (or 24-bit offset) where execution will continue if the jump is executed.

Description:

The SCSI Jump instruction is a conditional jump to the destination address, if the SCSI phase, data, or attention condition compares true with the phase, data, or attention condition described in the instruction. If the comparison is false, the SCRIPTS processor will not branch to the destination but will instead fetch the next instruction and continue execution.

When the optional *data* field is used, it is compared to the SFBR. This contains the most recent byte of any kind of data that has been moved into the SFBR register. The user's SCSI SCRIPTS program can determine which routine to execute next based on actual data values received. Using a series of these compares, the algorithm can process complex sequences with no intervention required by the external processor.

When the optional *MASK* keyword and its associated value are specified, the SCRIPTS processor allows selective comparisons of bits within the data byte. During the compare, any mask bits that are set will cause the corresponding bit in the data byte to be ignored for the comparison.

Notes:

Jump instructions are used to control the flow of the SCRIPTS routines. They are used to avoid phase mismatch interrupts in situations where multiple phase sequences are possible.

The REL keyword, which indicates relative addressing, is unrelated to the declarative keyword RELATIVE that establishes relative buffers.

Legal Forms:

```
JUMP address
JUMP address, IF ATN
JUMP address, IF Phase
JUMP address, IF CARRY
JUMP address, IF data
JUMP address, IF data AND MASK data
JUMP address, IF ATN AND data
JUMP address, IF ATN AND data AND MASK data
JUMP address, IF Phase AND data
JUMP address, IF Phase AND data AND MASK data
JUMP address, WHEN Phase
JUMP address, WHEN CARRY
JUMP address, WHEN data
JUMP address, WHEN data AND MASK data
JUMP address, WHEN Phase AND data
JUMP address, WHEN Phase AND data AND MASK data
JUMP address, IF NOT ATN
JUMP address, IF NOT Phase
JUMP address, IF NOT CARRY
JUMP address, IF NOT data
JUMP address, IF NOT data AND MASK data
JUMP address, IF NOT ATN OR data
JUMP address, IF NOT ATN OR data AND MASK data
JUMP address, IF NOT Phase OR data
JUMP address, IF NOT Phase OR data AND MASK data
JUMP address, WHEN NOT Phase
JUMP address, WHEN NOT CARRY
JUMP address, WHEN NOT data
JUMP address, WHEN NOT data AND MASK data
JUMP address, WHEN NOT Phase OR data
JUMP address, WHEN NOT Phase OR data AND MASK data
JUMP REL(address)
JUMP REL(address), IF ATN
JUMP REL(address), IF Phase
JUMP REL(address), IF CARRY
JUMP REL(address), IF data
JUMP REL(address), IF data AND MASK data
JUMP REL(address), IF ATN AND data
JUMP REL(address), IF ATN AND data AND MASK data
JUMP REL(address), IF Phase AND data
JUMP REL(address), IF Phase AND data AND MASK data
JUMP REL(address), WHEN Phase
JUMP REL(address), WHEN CARRY
JUMP REL(address), WHEN data
JUMP REL(address), WHEN data AND MASK data
JUMP REL(address), WHEN Phase AND data
JUMP REL(address), WHEN Phase AND data AND MASK data
JUMP REL(address), IF NOT ATN
JUMP REL(address), IF NOT Phase
```

```
JUMP REL(address), IF NOT CARRY
JUMP REL(address), IF NOT data
JUMP REL(address), IF NOT data AND MASK data
JUMP REL(address), IF NOT ATN OR data
JUMP REL(address), IF NOT ATN OR data AND MASK data
JUMP REL(address), IF NOT Phase OR data
JUMP REL(address), IF NOT Phase OR data AND MASK data
JUMP REL(address), WHEN NOT Phase
JUMP REL(address), WHEN NOT CARRY
JUMP REL(address), WHEN NOT data
JUMP REL(address), WHEN NOT data AND MASK data
JUMP REL(address), WHEN NOT Phase OR data
JUMP REL(address), WHEN NOT Phase OR data AND MASK data
```

LOAD

```
LOAD register, byte_count, [DSAREL(]source_address[)]
```

Supported by: SYM53C810A, SYM53C860, SYM53C825A, SYM53C875, SYM53C876, SYM53C885, SYM53C895

Definition: Load data from memory to an internal register of the SYM53C8XX.

Operands: **register** is one of the register names in the SYM53C8XX operating register set.

byte_count is the number of bytes (1-4) to be transferred from the source_address.

DSAREL indicates that the source_address is an offset and should be added to the DSA register to obtain the physical address (DSA relative).

source_address is the physical address or offset from the DSA to obtain the physical address of the data to be loaded into the register.

Example:

```
LOAD SCRATCHA0, 4, data_buf
LOAD SCRATCHA3, 2, DSAREL (0x02)
```

Format:

DCMD Register				**DBC Register**				**DSPS register**
31 ...29	28	27....25	24	23	22....16	15.....3	2 0	31 0
111	X	000	1	0	X.. X	00..00	XXX	XX... XX
Instr type	DSA Relative	RES	Load/ Store	RES	Reg Addr	RES	Byte Count	Source Addr/DSA Offset

Fields: **Instruction Type** - Load/Store

DSA Relative- indicates source address location
0 - DSPS contains actual address of data to load
1 - DSPS contains a 24-bit offset value that is added to the DSA to determine the source address.

Load/Store - This field defines whether the instruction will be executed as a Load or a Store.

0 - Store instruction
1 - Load instruction

Reg Addr- These bits select the register to load within the SYM53C8XX operating register set.

Byte Count - Indicates the number of bytes to transfer. Valid values are 1, 2, 3, or 4.

Source Addr - Actual address (or offset from the DSA) of the data to load into the SYM53C8XX register.

Description:

The Load instruction is a more efficient means than the Move Memory instruction of moving data from a memory location to an internal register of the SYM53C8XX. It is a two-dword instruction, compared to three dwords for a Memory Move. This instruction may be used to move up to 4 bytes. The number of bytes to load is indicated by the low order bits in the first dword of the instruction. The maximum number of bytes to load is defined by the Register Address field, as illustrated in the following table:

DBC Bits 17-16 (Register Address bits A1-A0)	Number of Bytes to Load
00	1, 2, 3, or 4
01	1, 2, or 3
10	1 or 2
11	1

Notes:

The register address and memory address must have the same byte alignment, and the byte count set so that it does not cross dword boundaries. The memory address may not map back to the SYM53C8XX operating registers, although it may map back to a location in the SCRIPTS RAM. If these conditions are violated, a PCI illegal read/write cycle will occur and the chip will issue an Interrupt (Illegal Instruction Detected) immediately following, because the intended operation did not happen.

Legal Forms:

```
LOAD register, byte_count, source_address
LOAD register, byte_count, DSAREL(source_address)
```

MOVE

```
MOVE {FROM | count,} [PTR] address, {WITH | WHEN}phase
```

Supported by: All Symbios Logic PCI-SCSI I/O Processors

Definition: SCSI Block Move

Operands: **FROM** indicates table indirect addressing mode.

Note: FROM and PTR must not be used in the same instruction.

count is a 24-bit number of bytes to transfer across the SCSI bus.

PTR sets the indirect bit if present, it is clear otherwise.

Note: FROM and PTR must not be used in the same instruction

address is the 32-bit starting address of the data in memory.

WITH/WHEN sets the mode for the device; WITH for target mode and WHEN for initiator mode.

Phase is used to specify the Message, Command/Data, and Input/Output bit values that identify the SCSI phase in the instruction. The desired phase value is compared with the actual values of the SCSI phase lines before the SYM53C8XX performs the instruction. This field is only valid for initiator mode and should not be used in target mode.

Example:

```
MOVE FROM dev_1, WITH MSG_IN
MOVE 6, cmd_buf, WHEN CMD
```

Format:

DCMD Register					**DBC Register**	**DSPS register**
31 30	29	28	27	26 24	23 0	31 0
00	X	X	X	XXX	XX... XX	XX... XX
Instr type	Indirect	Table Indirect	Op code	SCSI Phase	Byte Count	Dest Addr

Fields: **Instruction Type** - Block Move

Indirect - Indirect Addressing Mode
0 - Use destination field as an address
1 - Use destination field as a pointer to an address

Table Indirect - Table Indirect Addressing Mode
0 - Use Absolute addressing mode
1 - Use destination address as offset from the value of DSA register.

Op code - This field defines whether the instruction will be executed as a Block Move or a Chained Block Move.

	Target	Initiator
MOVE	Op code = 0	Op code = 1
CHMOV	Op code = 1	Op code = 0

SCSI Phase - These bits reflect the actual values of the SCSI phase lines. The bit values are defined below.

Phase	Message	Command / Data	Input / Output
DATA_OUT	0	0	0
DATA_IN	0	0	1
COMMAND	0	1	0
STATUS	0	1	1
RES4	1	0	0
RES5	1	0	1
MESSAGE_OUT	1	1	0
MESSAGE_IN	1	1	1

* Note: 0 - False, negated; 1 - True, asserted. For these phases, SEL is negated and BSY is asserted.

* Res4 and Res5 are reserved SCSI phases. These combinations should not be used for standard SCSI implementations

Byte Count - 24-bit number indicating the number of bytes to transfer.

Dest Addr - Address to perform data transfer on.

Description:

There are various forms of the Block Move instruction. The “address” and ”count” specify the address and byte count fields of the instruction. If the optional keyword “PTR” is present, then the Indirect bit will be set. If the optional keyword FROM is present the Table Indirect bit will be set (For more information on Table Indirect addressing, refer to Chapter 9). PTR and FROM may not be used in the same instruction. “Phase” specifies the phase field of the instruction. WITH or WHEN are used to specify the Block Move function codes. WITH is used to signal the target role which sets the phase values, and WHEN is the initiator “test for phase” feature.

The SYM53C8XX waits for a valid phase (initiator) or drives the phase lines (target). In the initiator role, it performs a comparison looking for a match between the phase specified in the SCRIPT and the actual value on the bus. If the phases do not match, a phase mismatch interrupt occurs. If the phases match, data is transferred in or out according to the phase lines. After the last byte is transferred to its final destination, the SYM53C8XX fetches the next SCRIPTS instruction. If the target changes phase in the middle of a block move, a phase mismatch interrupt will occur.

Notes:

In target mode, a MOVE instruction with a byte count of zero can be used during Command phase. The SYM53C8XX will determine the number of bytes to move from the command group code in the first byte of the command.

If the command code is vendor unique, the SYM53C8XX uses the byte count from the instruction. If this byte count is zero, the chip issues an Illegal Instruction interrupt.

SYM53C825A, SYM53C875, SYM53C876, SYM53C885, SYM53C895 Only
If the SCSI group code is either Group 0, 1, 2, or 5 and if the Vendor Unique Enhancement bit 1 (VUE1) bit (SCNTL2 bit 1) is clear, the SYM53C8XX overwrites the DBC register with the length of the Command Descriptor Block: 6, 10, or 12 bytes. If the Vendor Unique Enhancement 1 (VUE1) bit (SCNTL2 bit 1) is clear and the SCSI group code is a vendor unique code, the chip receives the number of bytes in the count. If the VUE1 bit is set, the chip receives the number of bytes in the byte count regardless of the group code.

Legal Forms:

```
MOVE count, address, WITH phase
MOVE count, address, WHEN phase
MOVE count, PTR address, WITH phase
MOVE count, PTR address, WHEN phase
MOVE FROM address, WITH phase
MOVE FROM address, WHEN phase
```

MOVE MEMORY

```
MOVE MEMORY[NO FLUSH]count, source_address, destination_address
```

Supported by: All Symbios Logic PCI-SCSI I/O Processors; No Flush option is available with SYM53C810A, SYM53C860, SYM53C825A, SYM53C875, SYM53C876, SYM53C885, and SYM53C895 only.

Definition: Memory to Memory Move (DMA)

Operands: **NO FLUSH** allows the SYM53C8XX to perform the Move Memory without flushing the prefetch buffer.

count is a 24-bit expression which indicates the number of bytes to transfer.

source_address is the absolute 32-bit starting address of the data in memory.

destination_address is the absolute 32-bit destination address of where to move the data.

Example:

```
MOVE MEMORY 1024, Source_Buffer, Dest_Buffer
```

Format:

DCMD Register			DBC Register	DSPS register	TEMP Register
31 29	28 25 24		23 0	31 0	31 0
11 0	0000	X	XX... XX	XX...XX	XX...XX
Inst Type	RES	No Flush	Byte Count	Source Addr	Dest Addr

Fields: **Instruction Type** - Memory to Memory Move

No Flush - When this bit is set, the SYM53C8XX performs the Move Memory without flushing the prefetch buffer. When this bit is clear, the instruction automatically flushes the prefetch buffer. The No Flush option should be used if the source and destination are not within four instructions of the current Move Memory instruction. Note: this bit has no effect unless instruction Prefetching is enabled, by setting the Pre-fetch Enable bit in the DCNTL register.

Byte Count - 24-bit number indicating the number of bytes to transfer.

Source Addr- the absolute 32-bit starting address of the data in memory.

Dest Addr - the absolute 32-bit destination address of where to move the data.

Description

The Move Memory instruction is able to transfer data from one 32-bit location to another. A 24-bit counter allows large moves to occur with no intervention required by the processor.

If both addresses are in system memory, then the SYM53C8XX functions as a high-speed DMA controller, able to move data at speeds of (up to) 47 MB/s without using the processor or its cache memory.

If just the destination address is in system memory and the source is within the SYM53C8XX address space, then the instruction performs a register store to external memory.

If just the source address is in system memory and the destination is within the SYM53C8XX address space, then the instruction performs a register load from external memory.

Notes:

The Indirect Mode is not allowed for the Move Memory instruction.

If cache line bursting is not enabled, the source and destination addresses must be on the same byte boundary. If cache line bursting is enabled and the byte count is larger than 32, the lower four bits of the source and destination addresses must be identical. If these conditions are not met, an illegal instruction interrupt is generated.

If the SYM53C8XX is only I/O mapped, it cannot do memory-to-register or register-to-memory moves.

Legal Forms:

```
MOVE MEMORY count, src_address, dest_address
```

MOVE REGISTER

```
MOVE {register | {data8} | register operator data8} TO
register [WITH CARRY]
```

Supported by: All Symbios Logic PCI-SCSI I/O Processors; additional functionality supported by SYM53C825A, SYM53C875, SYM53C876, SYM53C885, SYM53C895.

Definition: Register to Register Move

Operands:

register is one of the registers listed in the SYM53C8XX register set section in Chapter 6 of this manual. Either the register address or register name may be used in this instruction.

data8 is an expression or value that evaluates to an 8-bit unsigned number. In the SYM53C825A/53C875/53C876/53C885/53C895, SFBR may be substituted for data8 to add two register values. Bit 23 of the first dword of the instruction indicates that the SFBR is to be used instead of a data8 value.

operator is one of the following operators: '|' (OR), '&' (AND), SHL (Shift Left), SHR (Shift Right), XOR, '+' (Add), '-' (Subtract). The enhanced Move Register instruction does not support the SHL or SHR operators. See the appropriate product data manual for detailed information on the supported operations.

WITH CARRY adds in the current value of the CARRY bit from the ALU during a "+" or "-"' operation. It is not allowed for any other operations.

Example:

```
MOVE 0xFF TO SFBR
MOVE SCNTL1 & 0x01 TO SCNTL1
```

SYM53C825A, SYM53C875, SYM53C876, SYM53C895, and SYM53C885 only:

```
MOVE SCRATCHA + SFBR to SFBR
MOVE SCRATCHA XOR SFBR to SFBR
```

Subtraction (SFBR - SCRATCHA)

```
MOVE SCRATCHA XOR 0xFF to SCRATCHA
MOVE SCRATCHA + 1 to SCRATCHA
MOVE SCRATCHA + SFBR to SFBR
```

Format:

DCMD Register			**DBC Register**				**DSPS register**
31 30	29 27	26 24	23	22 16	15 8	7 0	31 0
01	XXX	XXX	X	XXXXXXX	X...X	00...0 0	00...00
Inst Type	Function	Operator	Use data8/ SFBR	Register Address	Immediate Data	RES	RES

Fields:

Instruction Type - Read/Write

Function - in either the target or initiator role, the function bits select the desired register operation.

101 - Move the SCSI First Byte Received register (SFBR) to the specified destination register.

110 - Move the specified register to the SCSI First Byte Received register (SFBR).

111 - Read a specified register, modify it, and write the result back into the **same** register.

Operator - specifies which logical or arithmetic operation will be performed.

000 - move, no modification performed
001*- Shift source left one bit, store result in destination
010 - OR immediate data with source, store result in destination
011 - XOR immediate data with source, store result in destination
100 - AND immediate data with source, store result in destination
101*- Shift source right one bit, store result in destination
110 - ADD immediate data to source, store result in destination
111 - add in immediate data plus Carry bit to source; store result in destination
*Data is shifted through the Carry bit and the Carry bit is shifted into the data byte.

Use data8/SFBR (SYM53C825A/53C875/53C876/53C885/ 53C895 only) - When this bit is set, SFBR will be used instead of the data8 value during a Read/Write instruction. This allows the user to add two register values.

Register Address - a 7-bit value that specifies which register to use as the source register for the instruction.

Immediate Data - an 8-bit value that will be used as the second operand in the logical and arithmetic functions. For the move function, the specified data is stored in the destination register.

Description:

The Move Register instruction allows a register read-modify-write, or a move to/from a register from/to the SCSI First Byte Received register (SFBR).

The SYM53C8XX does not provide a true move from any source register to any destination register. To accomplish this, two register move instructions must be used. First move the source register to the SFBR register, then move the SFBR register to the desired destination register. The two register names in each line must be

identical, or one must be SFBR. The two registers must be byte-aligned. If the 32-bit absolute addresses of the source and destination registers are known, then a register to register move can also be accomplished by using the memory to memory move instruction. However, a SCRIPTS instruction written in this manner will be less portable to other machines than if the previous method is used.

Caution must be exercised when this instruction is used, because writing to certain registers could have adverse effects on the SCSI bus or the operation of the chip. When a register is written or read, side effects may occur; the degree and possibility of these effects must be clearly understood. The SYM53C8XX data manuals contain detailed descriptions of individual register and bit operations.

The Add and Subtract operators can be used for loop counters in SCRIPTS programming. To subtract one value from another, first XOR the value to subtract (subtrahend) with 0XFF, and add 1 to the resulting value. This creates a 2's compliment of the subtrahend. The two values can then be added to obtain the difference.

SYM53C825A, SYM53C875, SYM53C876, SYM53C895, and SYM53C885 only
These chips allow use of the SFBR register for easier addition, subtraction, and comparison of two separate values within the chip. The instruction can perform the specified operation on the specified register and the SFBR, then store the result back to the specified register or the SFBR. The SFBR is used in place of the data8 value in the Read/Write operation. Subtraction cannot be used when the SFBR is used instead of a data8 value, because the SFBR value is not known at compile time.

Notes:

The mathematical operation is performed by the chip during execution, not by the assembler when the SCRIPTS routine is being assembled.

Legal Forms:

In the following, where the word **register** appears twice for an instruction the register name must be the same name for both the source and destination, not two different register names.

```
Move register to register
Move data8 to REGISTER
Move REGISTER SHL REGISTER
Move REGISTER | data8 to REGISTER
Move REGISTER XOR data8 to REGISTER
Move REGISTER & data8 to REGISTER
Move REGISTER SHR REGISTER
Move REGISTER + data8 to REGISTER
Move REGISTER + data8 to REGISTER with Carry
Move REGISTER - data8 to REGISTER
Move data8 to SFBR
Move REGISTER to SFBR
Move REGISTER SHL SFBR
Move REGISTER | data8 to SFBR
```

```
Move REGISTER XOR data8 to SFBR
Move REGISTER & data8 to SFBR
Move REGISTER SHR SFBR
Move REGISTER + data8 to SFBR
Move REGISTER - data8 to SFBR
Move REGISTER + data8 to SFBR with Carry
Move SFBR SHL REGISTER
Move SFBR | data8 to REGISTER
Move SFBR XOR data8 to REGISTER
Move SFBR & data8 to REGISTER
Move SFBR SHR REGISTER
Move SFBR + data8 to REGISTER
Move SFBR - data8 to REGISTER
Move SFBR + data8 to REGISTER with Carry
```

Additional Forms for SYM53C825A/53C875/53C876/ 53C885/53C895

```
Move SFBR to REGISTER
Move REGISTER | SFBR to REGISTER
Move REGISTER XOR SFBR to REGISTER
Move REGISTER & SFBR to REGISTER
Move REGISTER + SFBR to REGISTER
Move REGISTER + SFBR to REGISTER with Carry
Move REGISTER | SFBR to SFBR
Move REGISTER XOR SFBR to SFBR
Move REGISTER & SFBR to SFBR
Move REGISTER + SFBR to SFBR
Move REGISTER - SFBR to SFBR
Move REGISTER + SFBR to SFBR with Carry
Move SFBR to REGISTER
Move SFBR | SFBR to REGISTER
Move SFBR XOR SFBR to REGISTER
Move SFBR & SFBR to REGISTER
Move SFBR + SFBR to REGISTER
Move SFBR - SFBR to REGISTER
Move SFBR + SFBR to REGISTER with Carry
```

NOP

```
NOP
```

Supported by: All Symbios Logic PCI-SCSI I/O Processors

Definition: No operation

Operands: None

Example:
```
NOP
```

Format:

DCMD Register	DBC Register	DSPS Register
31 24	23 0	31 0
10000000	000 000	00...00
Op code	RES	RES

Fields: **Op code**—No Operation

Description: This instruction has no operation assignment and can be used as a delay function, or to reserve SCSI SCRIPTS patch areas.

Notes:

Legal Forms:
```
NOP
```

RESELECT

```
RESELECT {FROM Address | ID}, {REL(Address) | Address}
```

Supported by: All Symbios Logic PCI-SCSI I/O Processors

Definition: Reselect SCSI initiator device

Operands: **FROM Address** indicates table indirect mode.

ID is ID Number of the SCSI initiator that is to be selected.

REL indicates the use of indirect addressing.

Address is a 32-bit address that represents the address of the next instruction to fetch when the chip is selected or reselected.

Example:

```
RESELECT host_1, rsel_addr
RESELECT FROM entry_2, REL rsel_addr
```

Format:

DCMD Register					**DBC Register**			**DSPS Register**
31 30	29 27	26	25	24	23 20	19 16	15..... 0	31 0
01	000	0	0	0	00....00	XXXX	00 ..00	X.. X
Instr Type	Op code	Relative	Table Indirect	RES	RES	SCSI ID	RES	Alt Addr

Fields: **Instruction Type** - I/O

Op code - Reselect instruction

Relative Mode - Indicates that the 24-bit address is an offset from the current program counter.

Table Indirect Mode - Indicates that the SCSI ID, synchronous, and wide parameters should be loaded offset from the Data Structure Address.

SCSI ID - identifies the SCSI initiator to be reselected. This 4-bit field specifies the encoded destination ID. This field is part of the address if table indirect mode is used.

Alternate Address - specifies the memory address to fetch the next instruction if the SYM53C8XX is selected or reselected.

Description: The chip waits for Bus Free, arbitrates for the SCSI bus, then performs a reselection. If the chip loses arbitration it will wait again for Bus Free and continue trying until it is successful, unless there is a bus initiated interrupt. Once arbitration is won, the SYM53C8XX will continue to execute instructions until an interrupt or any

instruction related to the SCSI bus is issued. If arbitration terminates because of a bus initiated selection or reselection, the chip will use the 32-bit jump address value to fetch the next instruction and begin execution at that address. When the instruction completes then the next sequential instruction is fetched and executed. The Reselection process is illustrated in Figure 3-2.

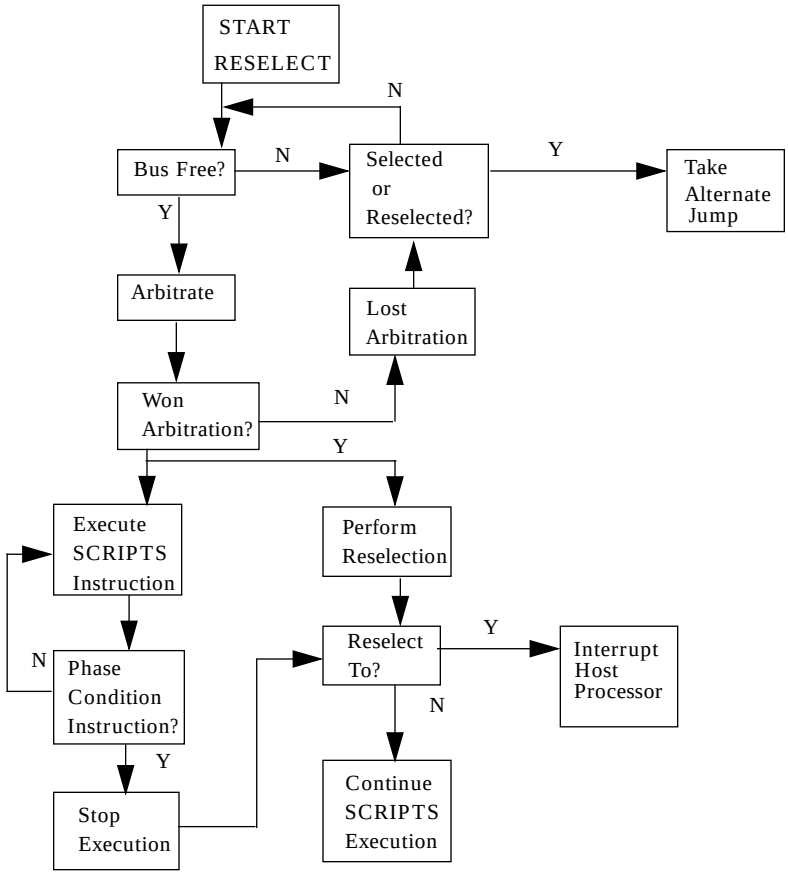


Figure 3-2
Reselection Instruction

Notes:

The REL keyword, which indicates relative addressing, is unrelated to the declarative keyword RELATIVE that establishes relative buffers.

Legal Forms:

```
RESELECT scsi_id, address
RESELECT FROM table_entry, address
RESELECT scsi_id, REL(address)
RESELECT FROM table_entry, REL(address)
```

RETURN

```
RETURN [, {IF | WHEN}[NOT][ATN | Phase] [AND | OR] [data[, AND MASK
data]]]
RETURN [, {IF | WHEN}[NOT] CARRY]
```

Supported by: All Symbios Logic PCI-SCSI I/O Processors

Definition: SCSI Transfer Control - Return from a Subroutine

Operands:

WHEN forces the SCRIPTS engine to wait for a valid SCSI bus phase before continuing. A valid phase is indicated by assertion of the SREQ/ signal.

IF causes the SYM53C8XX to check immediately for a valid SCSI bus phase without waiting. IF should not be used when comparing for a phase, as this could yield unpredictable results. The only exception is if a WHEN conditional was used just prior to the IF conditional, for any given sequence of phase checks.

NOT negates the comparison. It clears the True bit if present, otherwise the True bit is set.

Phase is used to specify the Message, Command/Data, and Input/Output bit values that identify the SCSI phase in the instruction. The desired phase value is compared with the actual values of the SCSI phase lines before the SYM53C8XX performs the instruction. This field is only valid for initiator mode and should not be used in target mode.

ATN indicates that a return should take place based on the state of the initiator SATN/ signal. This field is valid only for target mode and should not be used in initiator mode.

data represents an 8-bit value that is stored in the data field of the instruction. In addition the Compare Data bit is set.

MASK represents an 8-bit value that is stored in the mask field of the instruction. Any bit that is set in the mask causes the corresponding bit in the data byte to be ignored at the time of the comparison.

CARRY indicates that a return should take place based on the value of the Carry bit in the ALU.

Example:

```
RETURN
RETURN WHEN DATA_OUT
```

Format:

DCMD Register			DBC Register								DSPS Register
31 30 29 27		26 24	23,22	21 20		19	18	17	16	15 8	7 0	31 0
10	010	XXX	00	0	0	X	X	X	X	X...X	X...X	0...0
Inst Type	Op code	SCSI Phase	RES	Carry Test	RES	True	Comp Data	Comp Phase	Wait	Mask	Data	RES

Fields:

Op code - Transfer Control, Return instruction

SCSI Phase - These bits reflect the actual values of the SCSI phase lines. The bit values are defined below

Phase	Message	Command / Data	Input / Output
DATA_OUT	0	0	0
DATA_IN	0	0	1
COMMAND	0	1	0
STATUS	0	1	1
RES4	1	0	0
RES5	1	0	1
MESSAGE_OUT	1	1	0
MESSAGE_IN	1	1	1

* Note: 0 - False, negated; 1 - True, asserted. For these phases, SEL is negated and BSY is asserted.

* Res4 and Res5 are reserved SCSI phases. These combinations should not be used for standard SCSI implementations

Carry Test - When this bit is set, true/false comparisons may be made based on the ALU Carry bit. The Carry test may not be combined with other types of comparisons.

True - Transfer on TRUE/FALSE condition
0 - Transfer if condition is FALSE
1 - Transfer if condition is TRUE

Compare Data - Compare data byte to the SFBR register.
0 - Do not compare data
1 - Perform comparison

Compare Phase - Compare current SCSI phase to SCSI phase field or SATN/. This bit is set whenever the Phase operand is used.
0 - Do not compare phase
1 - Perform comparison

Wait - Wait for valid phase. This bit is set by the WHEN operand in the instruction, and cleared by the IF operand.
0 - Perform comparison immediately
1 - Wait for valid phase (SREQ/ asserted by target)

Mask - 8-bit field that is used to mask the value in SFBR before the comparison with the data field in the instruction takes place. As a

result of this operation, any bits that are set will cause the corresponding bit in the data byte to be ignored. If this field is not specified, a mask of 0x00 is used.

Data - 8-bit field that is compared with the incoming data after the mask operation with the mask byte takes place. Comparison indicates either an equal or not equal condition. If the Data field is not specified, the compare data bit is cleared and 0x00 is coded for both the mask and data bytes.

Description:

The SCSI RETURN instruction is a conditional return from a subroutine to the effective address, stored in the chip's TEMP register, if the SCSI phase, data, or attention condition compares true with the condition specified in the instruction.

When the optional *data* field is used, it is compared to the SFBR. This contains the most recent byte of any kind of data that has been moved into the SFBR register. The user's SCSI SCRIPTS program can determine which routine to execute next based on actual data values received. Using a series of these comparisons, the algorithm can process complex sequences with no intervention required by the external processor.

When the optional *MASK* keyword and its associated value are specified the SCRIPTS processor allows selective comparisons of bits within the data byte. During the comparison, any bits that are set in the mask byte will cause the corresponding bit in the data byte to be ignored for the comparison.

Notes:

If a RETURN instruction is executed without any previous CALL instruction, then there is no proper return address in the chip's TEMP register. This may cause the chip to generate an illegal op code after the return.

Legal Forms:

```
RETURN
RETURN, IF ATN
RETURN, IF Phase
RETURN, IF CARRY
RETURN, IF data
RETURN, IF data AND MASK data
RETURN, IF ATN AND data
RETURN, IF ATN AND data AND MASK data
RETURN, IF Phase AND data
RETURN, IF Phase AND data AND MASK data
RETURN, WHEN Phase
RETURN, WHEN CARRY
RETURN, WHEN data
RETURN, WHEN data AND MASK data
RETURN, WHEN Phase AND data
RETURN, WHEN Phase AND data AND MASK data
RETURN, IF NOT ATN
```

```
RETURN, IF NOT Phase
RETURN, IF NOT CARRY
RETURN, IF NOT data
RETURN, IF NOT data AND MASK data
RETURN, IF NOT ATN OR data
RETURN, IF NOT ATN OR data AND MASK data
RETURN, IF NOT Phase OR data
RETURN, IF NOT Phase OR data AND MASK data
RETURN, WHEN NOT Phase
RETURN, WHEN NOT CARRY
RETURN, WHEN NOT data
RETURN, WHEN NOT data AND MASK data
RETURN, WHEN NOT Phase OR data
RETURN, WHEN NOT Phase OR data AND MASK data
```

SELECT

```
SELECT [ATN] {FROM Address | ID}, {REL(Address) | Address}
```

Supported by: All Symbios Logic PCI-SCSI I/O Processors

Definition: Select SCSI target device.

Operands: **FROM Address** indicates table indirect mode.

ID is the ID Number of the SCSI target that is to be selected.

REL indicates the use of relative addressing.

Address - a 32-bit address (or 24-bit offset) that represents the address of the next instruction to fetch if the chip is selected or reselected by another device.

Example:

```
SELECT host_1, sel_addr
SELECT FROM entry_2, sel_addr
```

Format:

DCMD Register					**DBC Register**					**DSPS Register**	
31 30	29 27	26	25	24	23 20		19 16		15 0	31	0
01	000	0	0	X	0000		XXXX		00...00	XX...XX	
Instr Type	Op code	Relative	Table Indirect	Select with ATN	RES		SCSI ID		RES	Dest Addr	

Fields: **Instruction Type** - I/O

Op code - Select instruction

Relative Mode - Indicates that the 24-bit address is an offset from the current program counter.

Table Indirect Mode - Indicates that the SCSI ID and synchronous and wide parameters should be loaded offset from the Data Structure Address.

Select with ATN - indicates whether or not the SCSI ATN signal should be asserted.

SCSI ID - identifies the SCSI target to be selected. This 4-bit field specifies the encoded destination ID. This field is reserved if table indirect mode is used.

Destination Address - specifies the memory address to fetch the next instruction if the chip is selected or reselected during the selection.

Description:

The chip waits for Bus Free, arbitrates for the SCSI bus, then performs a selection. If the chip loses arbitration it will wait again for Bus Free and continue trying until it is successful, unless there is a bus initiated interrupt. Once arbitration is won, the SYM53C8XX will continue to execute instructions until an interrupt or any instruction related to the SCSI bus is issued. If arbitration terminates because of a bus initiated selection or reselection, the chip will use the 32-bit jump address value to fetch the next instruction and begin execution at that address. When the instruction is completed then the next sequential instruction is fetched and executed.

Notes:

The REL keyword, which indicates relative addressing, is unrelated to the declarative keyword RELATIVE that establishes relative buffers.

Legal Forms:

```
SELECT scsi_id, address
SELECT FROM table_entry, address
SELECT ATN scsi_id, address
SELECT ATN FROM table_entry, address
SELECT scsi_id, REL(address)
SELECT FROM table_entry, REL(address)
SELECT ATN scsi_id, REL(address)
SELECT ATN FROM table_entry, REL(address)
```

SET

```
SET {ACK|ATN|TARGET|CARRY}[and {ACK | ATN | TARGET | CARRY} ...]
```

Supported by: All Symbios Logic PCI-SCSI I/O Processors

Definition: Asserts SCSI ACK or ATN, or sets internal flags

Operands:
ACK sets the Assert SCSI ACK bit.
ATN sets the Assert SCSI ATN bit.
TARGET sets the Set Target role bit.
CARRY sets the CARRY bit in the ALU.

Example:
```
SET TARGET
SET ACK and TARGET
```

Format:

DCMD Register			**DBC Register**								**DSPS Register**
31 30 29 25		24	23 ..11	10	9	8 7	6	5 4	3	2 0	31 0
01	01100	0	00 ..00	X	X	000	X	00	X	000	00...00
Instr Type	Op code	RES	RES	Set Clear Carry	Set/ Clear Target Mode	RES	Set/ Clear SACK/	RES	Set/ Clear SATN/	RES	RES

Fields:

Op code - I/O, Set instruction

Set/Clear Carry
1 - sets the Carry bit in the ALU
0 - has no effect

Set/Clear Target Mode
1 - places the chip into target mode
0 - has no effect

Assert SCSI ACK
1 - asserts the SCSI acknowledge signal
0 - has no effect

Assert SCSI ATN
1 - asserts the SCSI attention
0 - has no effect

Description: The chip asserts the SCSI bus bits requested in the flags field. Currently four bits are defined, allowing the SCSI ACK, target role, and ATN bits to be set, as well as the Carry bit in the ALU. Bit 10 is for Carry, bit 9 is for target, bit 6 is for Acknowledge, and bit 3 is for Attention.

Notes:

Legal Forms:

```
SET ACK
SET ATN
SET TARGET
SET CARRY
SET ACK and ATN
SET ACK and TARGET
SET ACK and CARRY
SET ATN and TARGET
SET ATN and CARRY
SET TARGET and CARRY
SET ACK and ATN and TARGET
SET ACK and ATN and CARRY
SET ACK and ATN and TARGET and CARRY
```

STORE

```
STORE [NOFLUSH] register, byte_count,
[DSAREL(]destination_address[)]
```

Supported by: SYM53C810A, SYM53C860, SYM53C825A, SYM53C875, SYM53C876, SYM53C895, SYM53C885

Definition: Store data from an internal SYM53C8XX register to memory.

Operands:

NOFLUSH indicates that the prefetch buffer should not be flushed when the instruction executes

register is one of the register names in the SYM53C8XX operating register set.

byte_count is the number of bytes (1-4) to be transferred from the source_address.

DSAREL indicates that the source_address is an offset and should be added to the DSA register to obtain the physical address (DSA relative).

Note: the FROM keyword can still be used to indicate DSA relative addressing, but it is being phased out in favor of DSAREL.

destination_address is the physical address or offset from the DSA to obtain the physical address of the destination.

Example:

```
STORE SCRATCHA0, 4, data_buf
STORE SCRATCHA3, 2, DSAREL (0x02)
STORE NOFLUSH SCRATCHA0, 4, data_buf
```

Format:

DCMD Register					**DBC Register**				**DSPS register**
31 ...29	28	27, 26	25	24	23	22....16	15....3	2 0	31 0
111	X	00	X	0	0	X.. X	00..00	XXX	XX... XX
Instr type	DSA Relative	RES	No Flush	Load/ Store	RES	Reg Addr	RES	Byte Count	Destination Addr/DSA Offset

Fields:

Instruction Type - Load/Store

DSA Relative- indicates source address location
0 - DSPS contains actual address of data to load
1 - DSPS contains a 24-bit offset value that is added to the DSA to determine the source address.

No Flush - When this bit is clear, the prefetch buffer will be flushed during the Store instruction. When set, the prefetch buffer will not be flushed automatically on a Store instruction.

Load/Store - This field defines whether the instruction will be executed as a Load or a Store.

0 - Store instruction
1 - Load instruction

Reg Addr- These bits select the register to load within the SYM53C8XX operating register set.

Byte Count - 3-bit number indicating the number of bytes to transfer.

Destination Addr - Actual address (or offset from the DSA) of the destination address.

Description:

The Store instruction is a more efficient means than the Move Memory instruction of moving data from an internal register of the SYM53C8XX to memory. It is a two-dword instruction. This instruction may be used to move up to 4 bytes. The number of bytes to store is indicated by the low order bits in the first dword of the instruction, as illustrated in the following table:

Notes:

DBC Bits 17-16 (Register Address bits A1-A0)	Number of Bytes to Store
00	1, 2, 3, or 4
01	1, 2, or 3
10	1 or 2
11	1

The register address and memory address must have the same byte alignment, and the byte count set so that it does not cross dword boundaries. The memory address may not map back to the SYM53C8XX operating registers, although it may map back to a location in the SCRIPTS RAM. If these conditions are violated, a PCI illegal read/write cycle will occur and the chip will issue an Interrupt (Illegal Instruction Detected) immediately following, because the intended operation did not happen.

Legal Forms:

```
STORE register, byte_count, destination_address
STORE register, byte_count, DSAREL (destination_address)
STORE NOFLUSH register, byte_count, destination_address
```

WAIT DISCONNECT

```
WAIT DISCONNECT
```

Supported by: All Symbios Logic PCI-SCSI I/O Processors

Definition: Wait for SCSI bus disconnect

Operands: None

Example:
```
WAIT DISCONNECT
```

Format:

DCMD Register			DBC Register	DSPS Register
31 30	29 25	24	23.....0	31 0
01	00100	0	00 .. 00	00...00
Instr Type	Op code	RES	RES	RES

Fields:

Instruction Type - I/O

Op code - Wait Disconnect

Description: The initiator waits for a disconnect from the SCSI bus. A legal disconnect is defined as a loss of busy and select for the specified bus free time, following a DISCONNECT message or a COMMAND COMPLETE message. If the SCSI Disconnect Unexpected (SDU) bit (SCNTL2, bit 7)is clear and a disconnect occurs, the next SCSI SCRIPTS instruction will be executed. If the SDU bit is set and a disconnect occurs, an Unexpected Disconnect interrupt will occur.

Notes:

Legal Forms:
```
WAIT DISCONNECT
```

WAIT RESELECT

```
WAIT RESELECT {REL(Address) | Address}
```

Definition: Wait for reselection from target

Supported by: All Symbios Logic PCI-SCSI I/O Processors

Operands: **REL** indicates the use of relative addressing.

Address is a 32-bit address (or 24-bit offset) that represents the address of the next instruction to fetch if the chip is selected, or if the SIGP bit in the ISTAT register is set.

Example:

```
WAIT RESELECT alt_addr
WAIT RESELECT REL(alt_addr)
```

Format:

DCMD Register				DBC Register	DSPS Register
31 30 29 27	26	25 24		23 0	31 0
01	010	X	00	00 00	XX...XX
Inst Type	Op code	Relative	RES	RES	Dest Addr

Fields:

Instruction Type - I/O

Op code - I/O instruction type, Wait Reselect

Relative Mode - Indicates that the 24-bit address is an offset from the current program counter.

Dest Addr - Specifies the memory address to fetch the next instruction if a reselection occurs or the SIGP bit is set by the host processor.

Description: The initiator waits to be reselected by a previously selected target device. If the chip is responding to a previous reselection, it will fetch and execute the next instruction. If the chip has already responded to reselection, it will immediately fetch the next instruction. If the operation completes as expected, the next instruction is fetched and executed by the SYM53C8XX. However, if the chip is selected, then the alternate jump address should contain the address of an algorithm for a selection. Include in the address a wait for selection (target role) instruction. That instruction's alternate address is the error recovery algorithm (for initiator role—reselect). The chip can determine exactly what happened and transfer control to the appropriate SCSI SCRIPTS algorithm. If the SIGP bit in the ISTAT register is set by the host processor, the chip will also fetch the instruction at the alternate address. This allows the driver program to

schedule another I/O instead of waiting for the reselection to complete.This driver code activity is illustrated in Figure 3-3.

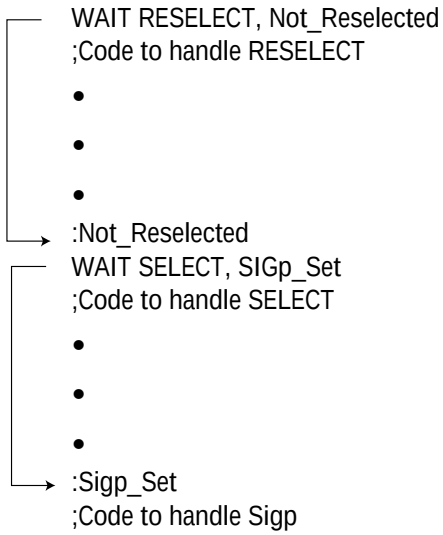


Figure 3-3
WAIT RESELECT and the SIGP bit

Notes:

With the SYM53C8XX byte compare capability of the transfer control instruction, the SCSI SCRIPTS algorithm can determine which target reselected the initiator and can jump to the correct algorithm for that particular target. The SYM53C8XX checks the SIGP bit before checking to see whether it has been reselected. SCSI SCRIPTS can be tuned for the various types of available target devices and executed with no external processor intervention.

Legal Forms:

```
WAIT RESELECT Address
WAIT RESELECT REL(address)
```

WAIT SELECT

```
WAIT SELECT {REL(Address) | Address}
```

Definition: Wait for selection from initiator

Operands: **REL** indicates the use of relative addressing.

Address is a 32-bit address (or 24-bit offset) that represents the address of the next instruction to fetch if the chip is selected, or if the SIGP bit in the ISTAT register is set.

Example:

```
WAIT SELECT alt_addr
WAIT SELECT REL(alt_addr)
```

Format:

DCMD Register				DBC Register			DSPS Register
31 27		26	25 24	23 10	9	8 0	31 0
01	010	X	00	00..00	1	00...00	X...X
Inst Type	Opcode	Relative Mode	RES	RES	Set Target Role	RES	Dest Addr

Fields:

Instruction Type - I/O

Op code - Wait Select instruction

Relative Mode - Indicates that the 24-bit address is an offset from the current program counter.

Set Target Role
1 - places the chip into target mode
0 - places the chip into initiator mode

Destination Address - specifies the memory address to fetch the next instruction if the device is reselected during the selection attempt, or if the SIGP bit is set.

Description: The chip waits for a SCSI selection by another device on the SCSI bus. If the chip is already selected, then the next SCSI SCRIPTS is fetched and executed. When a bus initiated interrupt or reselect occurs, the chip changes to the initiator role and fetches the next instruction from the address pointed to by the 32-bit jump address, and continues execution. If the SIGP bit in the ISTAT register is set by the host processor, the chip will also fetch the instruction at the alternate address. The SYM53C8XX checks the SIGP bit before checking to see whether it has been reselected.

Notes:

Legal Forms:

```
WAIT SELECT Address
WAIT SELECT REL(address)
```

Instruction Examples

This section illustrates the operation of the five SCSI instruction types supported by the SYM53C8XX. In each diagram, the SCSI SCRIPTS Source Code version shows how the operation would be expressed in the SCRIPTS language. This high-level textual format is translated by NASM into a hexadecimal format that is put inside a "C" language data declaration. After this intermediate form is compiled, the instruction exists in a binary form that can be loaded into host memory and fetched and executed by the SCRIPTS processor.

I/O Instruction Example

In this example, the processor is selecting the SCSI device with SCSI ID 01. The instruction is a Select With Attention, as indicated by the ATN keyword.

The SELECT instruction and ATN flag generates a value of 41h for the high order byte of the instruction, translating to a binary 01 for I/O Instruction type, 000 for the op code, and a 1 in the ATN flag bit. The SCSI target identity (01) is encoded in the next byte. The rest of the bits are reserved and should remain cleared. The alternate

address in the original SCRIPTS instruction is loaded into the DSPS register.

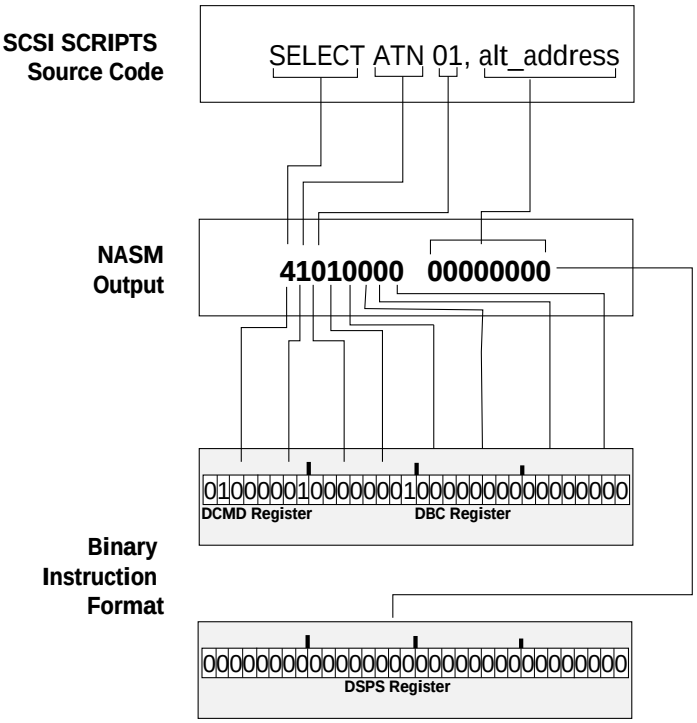


Figure 3-4
I/O Instruction Type

Memory Move Instruction Example

In this example, the processor is attempting to move eight bytes (as defined by `command_length`) from the source address (as defined by `command_buffer`) to the destination address relative to the source (as defined by `scratch_buffer`).

The MOVE MEMORY instruction generates an op code of C0 for the high order byte of the instruction. The remaining bits of the DCMD register are reserved and must be set to zero. The DBC register contains a value of eight as directed by the translation of the command_length of 0x08. Figure 3-5 shows the original SCRIPTS

language form of the instruction, the SCRIPTS compiler output, and the binary form of the first 32-bit word of the instruction.

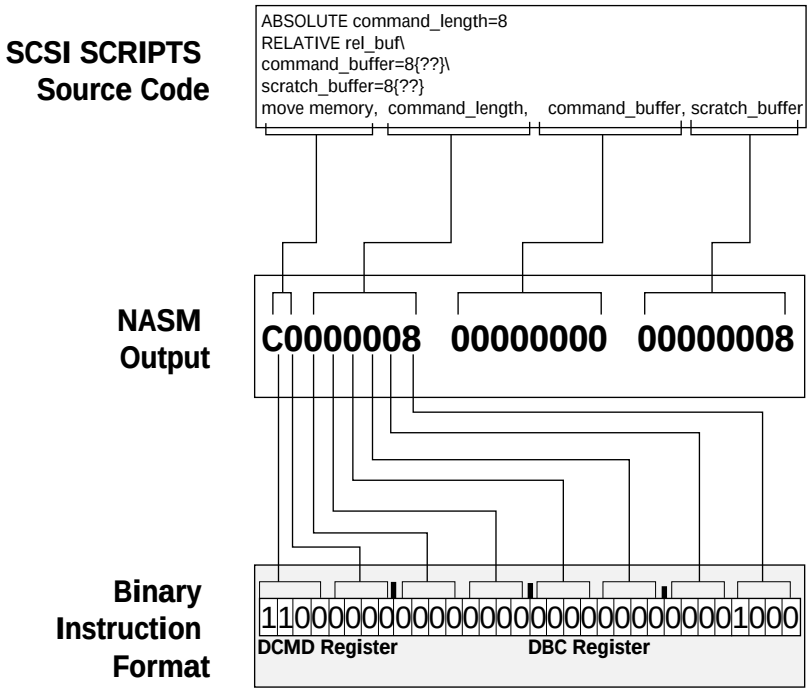


Figure 3-5
Memory Move Instruction Part 1

Figure 3-6 shows the Assembler output and the binary form of the second and third 32-bit words of the Memory Move instruction.

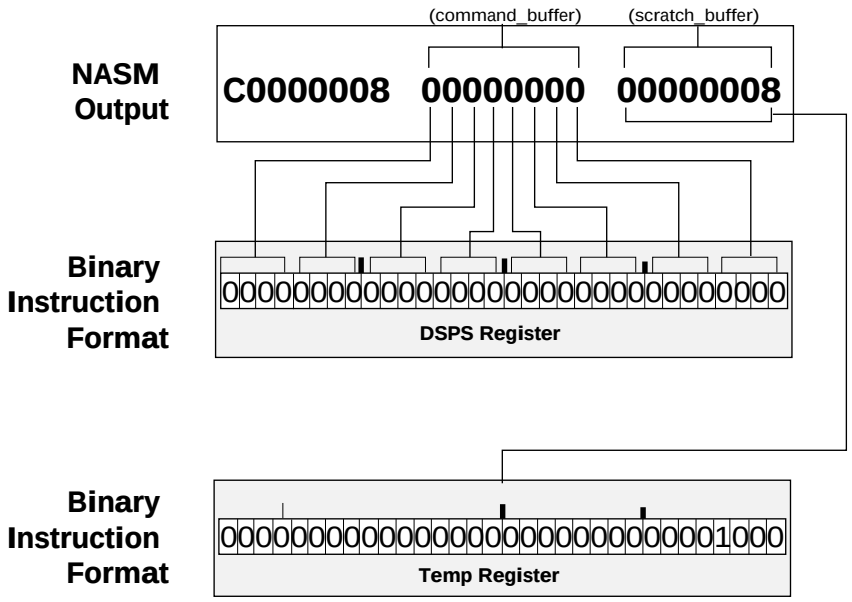


Figure 3-6
Memory Move Instruction Part 2

Transfer Control Instruction Example

In this example, the processor is performing an interrupt with a vector of 0xACB. The first version shows how the operation would be expressed in the SCRIPTS language. NASM translates the operation into the hexadecimal format shown. The hexadecimal format is then compiled producing the instruction in a binary form that can be loaded into host memory and put inside a “C” language data declaration. The INT instruction generates a hexadecimal value of 0x98 for the high order byte of the instruction, translating in binary to 10 for Transfer Control, and 011 for the op code for Interrupt.

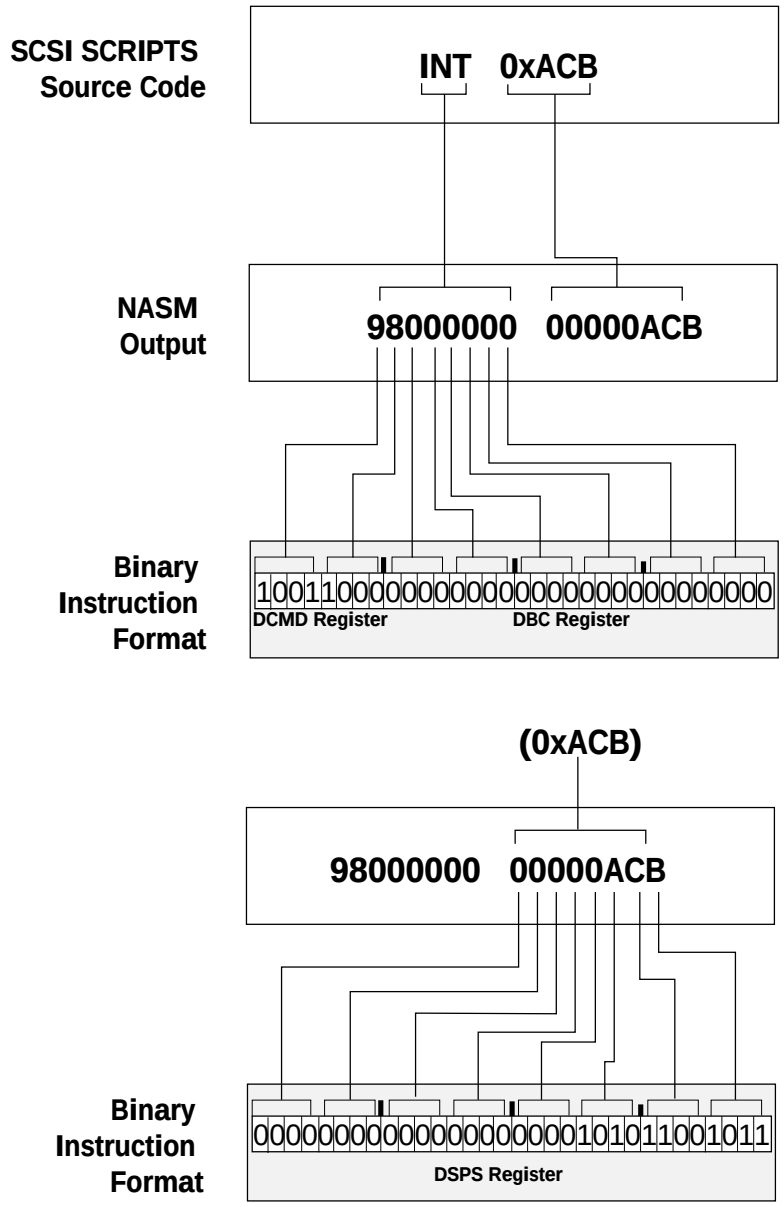


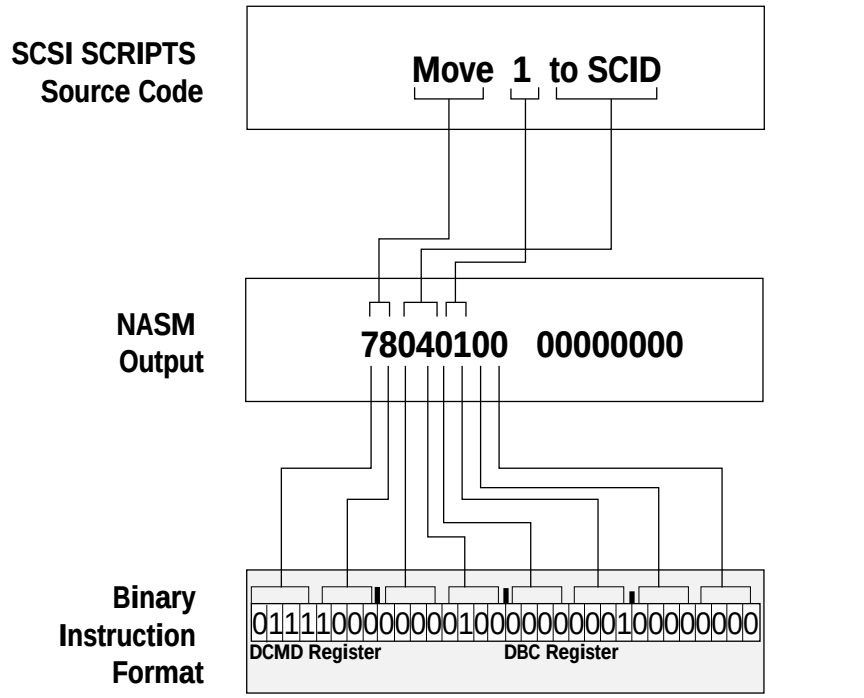
Figure 3-7
Transfer Control Instruction

Read/Write Instruction Example

This example writes 01 into the SCID register. This is illustrated by the translation of the hexadecimal compiler output into binary format.

The MOVE instruction is 78 in hexadecimal, translating into 01 for Read/Write; 111, the op code for the Read/Modify/Write function; and 00 in the operator field to indicate that the instruction will operate on the immediate data and write to the destination register.

The address of register SCID is 04 in hexadecimal, translating to a binary format for the Register Address bits of the DBC register.



Note: all bits in the DSPS register are reserved, and must remain cleared.

Figure 3-8
Read/Write Instruction

Block Move Instruction Example

In this example, the processor waits for a valid phase (indicated by SREQ/ being asserted) and compares it to CMD phase. If the phase matches, the processor then transfers the command descriptor block from the address represented by the `command_buffer`. In the hexadecimal version of the first 32-bit word of the instruction, Move is represented by 0A, which translates into binary as an op code of 00, indicating a Block Move instruction type. The 00 indicates that neither type of indirect addressing bits are on, 1 indicates that the processing is in the Initiator role, and 010 (Command) is the

expected value of the SCSI phase lines. The command length is six bytes, indicated by 06. This length is loaded into the DBC register.

The bottom portion of the illustration shows the second 32-bit word of the instruction, defined by `command_buffer`, the address from which the Block Move instruction will start transferring data. It is loaded into the DSPS register.

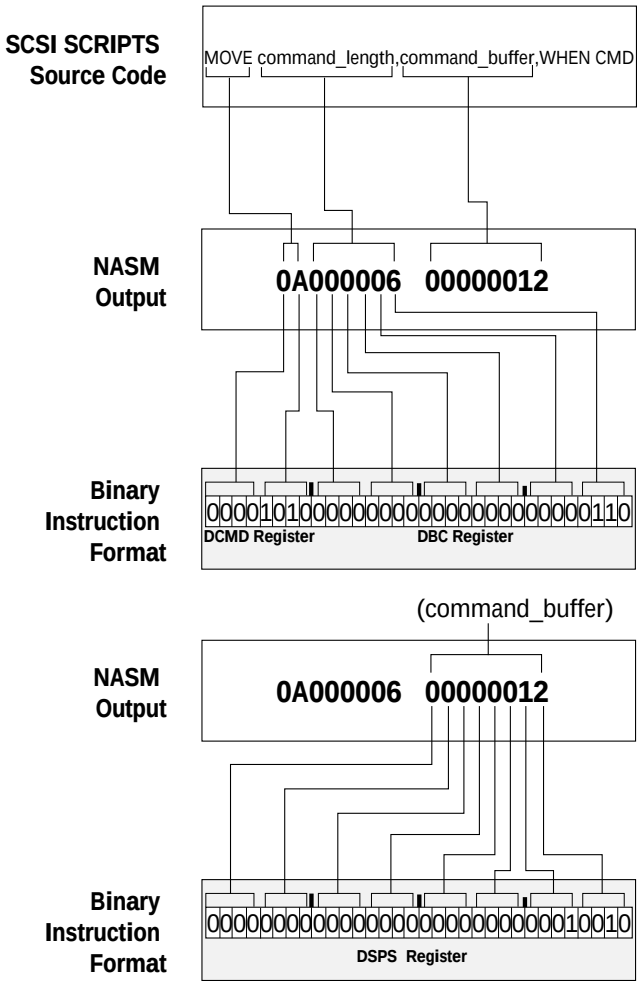


Figure 3-9
Block Move Instruction

Load/Store Instruction Example

In this example, the processor waits for a valid phase (indicated by SREQ/ being asserted) and compares it to CMD phase. If the phase matches, the processor then transfers the command descriptor block from the address represented by the command_buffer. In the hexadecimal version of the first 32-bit word of the instruction, STORE with the No Flush option is represented by E2, which translates into binary as an op code of 111, indicating a Load/Store instruction type. The 0 indicates that the DSPS value is the actual address to STORE from, and 0010 indicates that the prefetch buffer will not be flushed during the STORE, and that the SYM53C8XX is

performing a STORE rather than a LOAD instruction. The data will be stored to the SCRATCHA register; one, two, three, or four bytes may be stored.

The bottom portion of the illustration shows the second 32-bit word of the instruction, defined by command_buffer, the address to which the STORE instruction will start transferring data. It is loaded into the DSPS register.

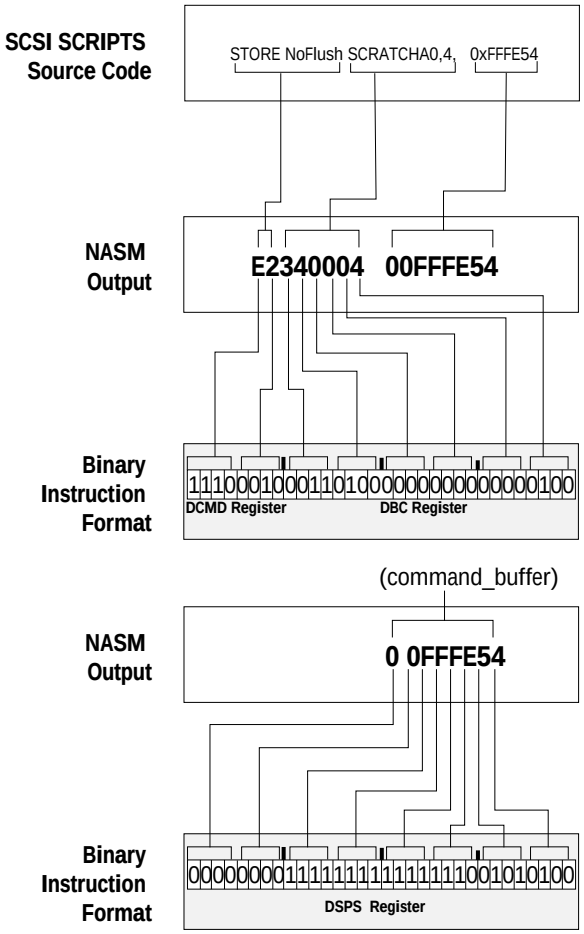


Figure 3-10
Load/Store Instruction

Chapter 4

Using the Symbios Logic Assembler

Overview

The Symbios Logic Assembler (NASM) is a DOS command line-driven assembler that supports the Symbios Logic SCSI I/O Processor family, including the SYM53C8XX. NASM creates a “C” header file from the SCSI SCRIPTS source file. It assembles SCSI SCRIPTS for inclusion into SCSI device driver software.

Inputs to the assembler are command line switches, and input and output file names. The assembler produces comprehensive error messages, cross referenced list files, and “C” include files. The source file may be created using any standard text editor that creates an ASCII file as output.

To assure portability, NASM does not provide support for directory paths. The resulting output file and the optional listing file will be placed in the directory where NASM is executed. Since the assembler is written in “C”, it can easily be ported to any non-DOS-based development environment that offers a “C” compiler.

Starting NASM

To run the assembler, copy the assembler executable file directly into the directory from which the assembly will be performed.

The NASM command line recognizes DOS wild card characters(“*”, “?”) in the file names. Entering `NASM` on the command line with no arguments produces a short description of all the valid switches.

Usage: `NASM filename [options]`

where:

`filename` is the name of the file to assemble. Files should be specified in the standard DOS format:`[d:] [path] name.ext` The file name is the root file name of the `.ss` file unless otherwise indicated.

`[options]` - are one or more of the following preceded by the hyphen (“-”) character

`a [arch]`	Specify SCSI architecture (default is SYM53C700)
`b`	Generate binary cross reference values
`e [filename[.err]]`	Save error messages (filename optional)
`l [filename[.lis]]`	Generate cross reference (filename optional)
`o [filename[out]]`	Generate “C” source output (filename optional)
`p [filename[.out]]`	Generate partial “C” header (filename optional)
`s [filename[.bin]]`	Generate .bin format output (filename optional)
`u`	Exclude module termination record
`v`	Verbose messages
`x`	List patch offsets in cross reference listing

Command Line Options

A [arch] - Specify processor for code generation

The `-A` option allows the programmer to specify the Symbios Logic chip for which code will be generated. The currently supported chips are listed in the table below, along with the corresponding number to enter to choose the architecture. An `ARCH` statement at the beginning of a SCRIPTS source file overrides any options typed in on command line. If the source file does not have an `ARCH` statement and no architecture is specified in the command line, NASM will use the default architecture (SYM53C700).

Product Name	ARCH Command line entry
SYM53C700 or SYM53C700-66	`-A 700`
SYM53C710	`-A 710`
SYM53C720	`-A 720`
SYM53C770	`-A 770`
SYM53C810	`-A 810`
SYM53C810A	`-A 810A`
SYM53C825	`-A 825`
SYM53C815	`-A 815`
SYM53C825A (all package variations)	`-A 825A`
SYM53C875 (all package variations)	`-A 875`
SYM53C876	`-A 876`
SYM53C895	`-A 895`
SYM53C885	`-A 885`

B - Binary Cross Reference Values

The `-B` option causes the `filename.lis` file to generate binary as well as hexadecimal opcodes in the listing file.

E - Creates an error listing file

This option allows an error message file to be created if errors occur during NASM assembly. If no file name is given, the `-e` option will create a file with the same root name as the source file, with a `.err` extension.

L - Creates a listing file

This option creates an assembly listing (`.LIS`) file. When invoked, the `-L` option creates a file with the same root name as the source file and a `.LIS` extension, unless otherwise specified.

O - Generate output file

The `-O` option creates a "C" style output (.OUT) file. When invoked, the `-O` option creates a file with the same root name as the source file and a `.OUT` extension, unless otherwise specified.

P - Generate Partial "C" Source

This option, which is mutually exclusive with the -o option, creates a partial "C" style output file with a `.out` extension, but no patch information is listed. If the `-o` and `-p` options are both specified, the `-p` option always takes precedence. The portions of the SCRIPTS outfile that are eliminated by the `-P` option are listed below. For more detail on the SCRIPTS output file, refer to Chapter 5.

- the `#define Ext_Count...` which is a count of external variables
- the `char *External_Names[Ext_Count]...` array of external variable names
- the `#define E_buf_name...` definition of the external buffer offset because it will always be zero
- the `#define Rel_Count...` which is a count of relative buffers
- the `ULONG Rel Patches [Rel_Count]...` array of relative patches
- the `#define R_buf_name...` define of the relative buffer offsets
- the `#define Abs_Count...` which is a count of Absolute variables
- the `char *Absolute_Names[Abs_Count]...` which is an array of absolute names
- the `ULONG A_absolute_Used[]...` array of locations where absolute variables are used
- the termination record is removed (as in the -U option)
- a `#define` instruction `0x????????` is added, which is the instruction count

S - Generate .BIN Output

This option generates a file with a `.bin` extension.

U - Omit Termination Record

This option instructs the assembler to omit the INSTRUCTIONS and PATCHES information from the output file. It must be used with the `-o` or `-p` option.

V - Verbose Messages

This option instructs the assembler to generate more comprehensive status messages.

X - Patch Offsets

Using this option produces an assembly level output file, including a list of patch addresses for each symbol. These addresses indicate where to patch each individual symbol value.

Example Assembler Command Lines

The following command lines are typical examples of how to use the various options.

1. `NASM demoPCI.ss`

   This command line produces no output files, but allows a quick syntax check on the SCRIPTS instructions in the file named *DEMOPCI.SS*

2. `NASM demoPCI.ss -a 875 -l -o -e errors.txt`

   This command line requests that NASM check the syntax and generate code for the SYM53C875 chip. The listing, error log, and standard C header will be generated. Since no filenames were specified for the listing and C header files, they will take the name of the input file, but with .LIS, and .OUT as the file extensions, respectively. The error log will be sent to the file named ERRORS.TXT

How NASM Parses SCRIPTS Files

SCSI SCRIPTS programs contain a series of lines. Blank lines, lines containing only white space, and anything after a semi-colon on a line are ignored.

The assembler is token oriented. It reads the source file and splits it up into tokens. White space and anything from a semicolon to the end of the line is not part of any token, and is ignored by the first pass of the assembler.

There are two types of tokens. Any string of consecutive letters, numbers, dollar signs, and underscores is a token. The second type of token consists of characters that are not part of other tokens. Anything that is not a letter, a digit, an underscore, or a dollar sign, will become a token. For example, the string “xxx = 0x123; assign value to xxx” contains three tokens. “xxx” is a token, “=” is a token, and “0x123” is a token.

Numeric values may be specified in decimal, hexadecimal, octal, or binary format. Decimal numbers are specified by a string of digits that does not begin with a zero. Octal numbers are specified by a string of digits that begins with zero. Hex numbers are specified by a string consisting of “0x” or “0X” and the hex digits of the number. Both upper and lower case are allowed. A binary number is similar to a hex number, except that “0b” or “0B” is used instead of “0x” or “0X”.

Assembler Declarative Keywords

To do its job efficiently, the assembler needs to recognize a set of commands that are different from the processor instructions. These commands, called declarative keywords, control the different aspects of code generation and are intended for the assembler's use. In most cases, the declarative keywords will not produce executable code by themselves, but must be combined with processor instructions to generate assembled code.

The declarative keywords are grouped functionally in Table 4-1, Table 4-2, and Table 4-3. They are listed alphabetically and defined in the remainder of this section.

Table 4-1
Data Definition and Storage Keywords

Function	Keyword
Equates	ABSOLUTE
Storage Definition	RELATIVE, EXTERNAL
Table Addressing	TABLE

Table 4-2
Code Generation Keywords

Function	Keyword
Code Generation	ARCH

Table 4-3
Miscellaneous Keywords

Function	Keyword
Module Definition	PROC
Code Entry Labels	ENTRY

ABSOLUTE

Equate Value with Symbol

Purpose

Use ABSOLUTE to define the symbol name by assigning it a numeric value. Once a name has been declared using ABSOLUTE, NASM will substitute this numeric value in each instruction where the name is used.

Syntax

```
ABSOLUTE name = expression
```

Fields

Example

```
ABSOLUTE bytes = 2048    ; A sector is 2048
                         bytes
ABSOLUTE sectors = 4     ; A cluster has 4
                         sectors
ABSOLUTE cluster = bytes ; cluster size
ABSOLUTE bytecnt = bytes ; bytecnt is an indexing
                         ; variable
```

Description

The ABSOLUTE keyword supplies a list of names, or labels, solely for the use of the assembler. NASM can refer to this list when it is actually assembling the program.

Notes

ARCH

Specify Target Architecture

Purpose

Use ARCH to direct the assembler to generate instructions that are specific to a chip architecture.

Syntax

```
ARCH chip_number
```

Fields

chip_number

Example

```
ARCH 810A
ARCH 875
```

Description

Notes

If used, this keyword should be placed before any executable statements so that the assembler knows which chip to generate code for. The chip architecture may also be specified on the assembler command line for the assembler using the `-A chip_number` option. ARCH takes precedence over the `-A` option in the NASM command line. The chip number entries should use the last three digits of the product number, as indicated in the example above.

ENTRY

Declare External Entry Point

Purpose

Use the ENTRY keyword to inform the driver program of the starting location of callable routines contained in a given SCRIPTS instruction. ENTRY allows the declaration of variables as entry points into the SCSI SCRIPTS instruction array. It defines the names and values of the variables, making them also available to the host development system.

Syntax

```
ENTRY label [, label ...]
```

Fields

Example

```
ENTRY start, Data_Out_Entry
```

Description

The ENTRY keyword indicates which SCRIPTS entry points should be made visible to the driver code. Only those entry points named in the ENTRY keyword will generate information in the assembler output file.

Notes

All entries must be used as a label somewhere in the SCRIPTS code, otherwise an error message will be reported.

EXTERN

Declare External Symbol

Purpose

Use the EXTERN keyword to inform the assembler that a symbol should be resolved at link time. This keyword allows the declaration of variables that are defined external to the SCRIPTS program. EXTERN causes the assembler to keep an array of offsets into the SCRIPTS array that the driver can use to patch SCRIPTS instructions into the driver program.

Syntax

```
EXTERN label [, label ...] or
EXTERN label = data_specifier [, label =
data_specifier...]
a data specifier is:
{byte_val[, byte_val]} or count{byte_val | ??}
```

Fields

A count is any valid constant with a value between 0 and 64K

Example

```
EXTERN buffer; a buffer in the driver
EXTERN buffer=1024{??}; same buffer, but now
         ; the debugger will have
         ; information about space
         ;  requirements
```

Description

The first form of the EXTERN syntax is only provided for compatibility with older versions of the SCRIPTS compiler. The second form (with space requirements information for the debugger) should be used in all new programs. Declarative instructions never allocate memory, but give the debugger or driver code the information required to allocate the memory.

PASS

Transfer an Element, Unaltered, to the output file

Purpose

Allows the programmer to pass a "C" element unaltered to the SCRIPTS output file and on to the "C" compiler. Using the Pass option avoids the need for runtime patching of the addresses of SCRIPTS objects. PASS is typically used for two types of "C" elements; either an include statement or a literal string.

Syntax

```
PASS(element)
```

Fields

Examples

Include statement: `PASS(include"SCRIPTS.h")`
Literal string: `Wait Reselect PASS(&alt_addr)`

Description

PASS tells NASM to pass everything between the left and right parentheses on to the output file, literally. Therefore, the passed statements can be read by the "C" compiler.

Notes

PROC

Define an output module

Purpose

PROC is used in the SCRIPTS code to build output arrays with names other than the generic array name `SCRIPT` that NASM normally assigns to SCRIPTS opcode arrays. This is useful when more than one SCRIPTS file is used in a driver program. It also allows several output arrays to be created with specific code segments in each one. When SCRIPTS storage space is limited, code can be divided into different sections where one section would fit in a limited space (such as SCRIPTS RAM) and the remaining code can be stored elsewhere.

Syntax

```
PROC label:
```

Fields

`label` is the name assigned to the SCRIPTS output array.

Examples

```
PROC Start:
```

Description

When a PROC keyword is used, the SCRIPTS output array in the `.out` file is given the name specified in `label`, overriding the default name `SCRIPT`. If additional PROC statements are used in the same SCRIPTS source file, NASM will create additional output arrays in the `.out` file with the name specified in `label` for each PROC statement.

Notes

RELATIVE

Define Contiguous Data Structure

Purpose

Use RELATIVE to begin the definition of a data structure named `baselabel` with offsets into the buffer specified by the labels. Allows the declaration of buffers to be positioned relative to one another. The expression used will be the offset from the start of the relative data area where the buffer variable is located.

Syntax

```
RELATIVE label = expression [, label = expression...]
or
RELATIVE baselabel \
label = data_specifier [, label = data_specifier...]


a data specifier is:
{byte_val [, byte_val]}
or
count{byte_val |??}
```

Fields

A byte_val is any valid constant with a value between 0 and 255. For example: 0x10 and 16 both represent a byte value of 16. Also, the special data value '??' can be used to indicate that a byte should be reserved, but that it should not be initialized to a specific value. The SCRIPTS program does not allocate memory; this is done by "C" code in the SCRIPTS debugger or in the driver code.
A count is any valid constant with a value between 0 and 64K.

Example

This example shows the typical use of the RELATIVE keyword. NASM syntax requires that no SCRIPTS statements span more than one line; however, in the case of the RELATIVE, this would result in a very unreadable source code file. The following example demonstrates the use of the logical line continuation character '\'. When this character is used, the assembler appends the next line to the end of the current line.

```
RELATIVE data_buffer\
identify_msg_buf =1{??}, \
synch_msgo_buf = {1,2,3,4,5},\
synch_msgi_buf = 5{??},\
cmd_buf = 12{??},\
W_cmd_buf = 12{??},\
stat_buf = 1{??},\
msg_in_buf = 1{??},\
disc_msg_in_buf = 2{??},\
read_cap_buf = {1,2,3,4,\
5,6,7,8},\
inquiry_buf = 36{??},\
request_sense_buf = 18{??},\
data_buf = 16384{??}
```

Description The RELATIVE keyword defines a template for a collection of data elements of the same or varying types, each of which can be accessed by a descriptive name, but **no storage is allocated**. It is up to the programmer to use the RELATIVE information that is placed in the output file to declare space in the driver program that the RELATIVE maps to.

Notes The first form of the RELATIVE syntax example is only provided for compatibility with older versions of the SCRIPTS compiler. The second form (with baselabel definition) should be used for all new programs.
Since the SCRIPTS array will have only offsets from the base address of the buffer, the SCRIPTS elements containing references to relative buffers will need to be patched by the driver program after the buffer space is allocated.

TABLE

Define Data Structure for Table Indirect Addressing

Purpose Use TABLE to describe a data structure that will be used with the table indirect addressing feature of the 53C8XX. The starting location for the buffer is defined by the data structure address written to the DSA register. The expression specifies the offset into the buffer and is added to the starting address of the buffer (DSA register) to form the absolute address. This feature allows SCRIPTS to be programmed into a ROM.

Syntax

```
TABLE tablelabel \
         label = data_specifier \
         [, label = data_specifier...]
a data specifier is:
         {byte_val [, byte_val]} or
         count{byte_val |??} or
         ID{byte_val | ??}
```

Fields A byte_val is any valid constant with a value between 0 and 255. For example, 0x10 and 16 both represent a byte value of 16. Also, the special data value ‘??’ can be used to indicate that a byte should be reserved, but that it should not be initialized to a specific value.
A count is any valid constant with a value between 0 and 64K.

Example This example shows the typical use of the TABLE keyword NASM does not generate any output based on the TABLE keyword. This example is a template for a data structure that will be used in the driver program or in the SCRIPTS debugger. NASM assembler syntax requires SCRIPTS statements to span no more than one line; however in the case of the TABLE, this would result in a very unreadable source code file. The following example demonstrates the use of the logical line end character (/).When this character is used, NASM appends the next line to the end of the current line.

```
TABLE table_indirect\
stat_buf = {??},\ ;stat_buf = 1 byte
msg_in_buf= {??},\ ;msg_in_buf = 1 byte
data_buf = 512{??},\
R_data_buf = 512{??},\ ; read data buffer
W_data_buf = 512{0xaa},\ ; write data buffer
W_cmd_buf = {0x0A, 0x00, 0x00, 0x00, 0x01, 0x00}, \
R_cmd_buf= {0x08, 0x00, 0x00, 0x00, 0x01, 0x00}, \
dum_buf  = 512{??},\
scsi_id  = ID{??},\
select_id = ID{0x33, 0x00, 0x00, 0x00}
```

Description Table indirect addressing allows a SCRIPTS program to be placed in ROM and still allows the driver program to dynamically specify different parameters for the BLOCK MOVE, SELECT, or RESELECT instructions.

Notes The TABLE keyword defines the table entries, each of which can be accessed by a descriptive name, but no storage is allocated. It is up to the programmer to provide the data definition and allocation for the SCRIPTS table in the driver program and load the DSA prior to execution of SCRIPTS routines.

Currently, only one TABLE keyword per SCRIPTS routine is allowed. An error message will be generated if multiple TABLEs are used.

The ID parameter in the data specifier allows initialization of the table entries for use with the FROM keyword of the SELECT and RESELECT instructions on the SYM53C8XX chips.

Conditional Keywords

Conditional keywords are used to test for conditions such as an expected phase or data byte.

If

The IF keyword indicates that a comparison is to be done immediately.

Example: `JUMP address, IF phase`

When

The WHEN keyword causes the chip (as an initiator) to wait for a phase to become valid. A valid phase is indicated by REQ/ being asserted on the SCSI bus. Since WHEN waits for the SCSI REQ/ signal when making a comparison, it may not work when comparing for conditions that are not related to the SCSI bus.

Example: `CALL address, WHEN data`

Logical Keywords

Logical keywords are used in conjunction with conditional keywords to add detail or additional comparisons to the conditions being tested.)

NOT

NOT negates (logically inverts) the conditions specified by the qualifiers that follow. For example, an instruction that reads `RETURN if NOT data` compares data to the contents of the SFBR register. If they are not identical, the operation will execute.

Example: `JUMP address, if NOT data`

AND

AND is used to compound the condition being tested. All conditions that are added with the AND keyword must be true for the operation to execute.

Example: `RETURN, WHEN data AND MASK DATA`

OR

OR specifies a list of conditions, one of which must be true for the operation to execute.

Example: `CALL REL (address), IF NOT ATN OR data`

Flag Fields

The Flag Fields keywords are used to signify that a flag field bit has been set. The flag field bits are controlled with the SET and CLEAR instructions

ACK

The target checks to see if the SCSI ACK/ signal is asserted.

Example: `CLEAR ACK`

ATN

The target checks to see if the initiator has set the SCSI ATN/ signal.
Example: `JUMP address, IF NOT ATN`

TARGET

By setting or clearing this bit, the SYM53C8XX is placed in target or initiator role. This must be done before the chip can execute target- or initiator-specific operations, such as reselection.

Example: `SET TARGET`

CARRY

This keyword checks the ALU Carry bit in the SYM53C8XX to determine which SCRIPTS routine to execute next. CARRY is not valid if phase or data clauses are used in the same instruction. Register Move (arithmetic) operations also affect the CARRY flag.

Example: `JUMP address, IF CARRY`

Qualifier Keywords

Qualifier keywords are used in conjunction with action keywords to add details about the instructions to be performed.

DSAREL

This keyword is only available in the Symbios Logic devices that support Load and Store instructions. It is used in Load/Store instructions to indicate that the data to be loaded or stored is relative to the DSA register. This keyword replaces the RELATIVE keyword, although NASM still supports RELATIVE as well.

Example: `STORE NOFLUSH SCRATCHA0, 4 DSAREL (address)`

FROM

This signifies that table indirect addressing is used. It can be used with Block Move or Select operations.

Example: `MOVE FROM address, WITH phase`

MASK

This keyword allows selective comparison of specified bits with the SCSI First Byte Received (SFBR) register. Any bits that are set in the mask byte eliminate the corresponding bits in the SFBR register.

Example: `RETURN WHEN data AND MASK DATA`

MEMORY

The MEMORY keyword is used in conjunction with an action keyword to signify a Memory to Memory Move instruction.

Example: `MOVE MEMORY 512, data_buf, data_buf1`

PTR

PTR causes the Indirect bit to be set in a Block Move instruction.

Example: `MOVE count, PTR address, WITH phase`

REG

This keyword allows access to register by register number instead of register name. The register number must be in parenthesis.

Example: `MOVE REG(10) + 0x01 TO REG(10)`

REL

This keyword indicates that relative addressing is used.

Example: `SELECT ID, REL(address)`

TO

This keyword indicates the destination of a Register Move operation.

Example: `MOVE data TO register`

WITH

The WITH keyword allows the target to drive the phase on the SCSI bus. This keyword is used for Target Move operations.

Example: `CHMOV count, address, WITH phase`

NOFLUSH

This keyword is used in the SYM53C8XX products that support instruction prefetching, in conjunction with Move Memory and Store instructions that affect the prefetch buffer. Its purpose is to preserve the contents of the prefetch buffer when one of these operations is performed.

Example: `STORE NOFLUSH SCRATCHA0,4 DSAREL (address)`

Other Keywords

Action Keywords

These words are used to execute SCSI SCRIPTS instructions. They are described in detail in Chapter 3.

SCSI Phases

These words are used to describe the phases of the SCSI bus. One of these keywords should be used in place of the word “phase” when it appears in programming examples in this manual. The SCSI phase keywords are CMD, COMMAND, DATA_IN, DATA_OUT, MSG_IN, MSG_OUT, STATUS, RES4, RES5.

Register Names

All register names are reserved keywords. A full list of register names and brief descriptions appears in Appendix B.

Chapter 5

The NASM Output File

Overview

The NASM assembler produces an output file with all the necessary data structures and information that a programmer writing a driver program needs to be able to load and run a SCSI SCRIPTS program. The assembler produces data structures compatible with ANSI “C”. The file can be included in a “C” program and compiled without any modifications.

Three command line parameters determine whether certain structures will be produced in the output file. The `-o` option will cause NASM to generate all of the structures described in this chapter. The `-p` option will cause only some of the structures to be generated; please see each section for the effects of the `-p` option. Finally, the `-u` option only affects the Termination Record which is detailed later in this chapter. The `-o` and `-p` options are mutually exclusive (i.e. only one or the other can be used); if they are used together in the command line, the `-p` option takes precedence. The `-u` option must be used in conjunction with either the `-o` or the `-p` option.

The example SCRIPTS program in Figure 5-1 shows the various types of structures produced by the NASM assembler.

Figure 5-1
Structures in a SCRIPTS Program

```
ARCH 825
ABSOLUTE Got_Selected= 0xA5
ABSOLUTE Not_Msg_Out= 0x11
ABSOLUTE Select_ID= 2
ABSOLUTE Command_Complete= 0x01
EXTERN ex_buf1
EXTERN ex_buf2
RELATIVE rel_buffer \
        rel_buf1= ??, \
        rel_buf2= 6{??}, \
        rel_buf3= ??
TABLE tbl_buffer \
        tbl_buf1= ??, \
        tbl_buf2= ??, \
        tbl_buf3= ??
ENTRY Start
ENTRY Send_CMD
ENTRY Send_DATA
Start:
        SELECT ATN Select_ID, REL(Interrupt)
        INT Not_Msg_Out, WHEN NOT MSG_OUT
        MOVE 1, rel_buf1, WHEN MSG_OUT
Send_CMD:
        MOVE 6, rel_buf2, WHEN CMD
Send_DATA:
        MOVE FROM tbl_buf1, WHEN DATA_OUT
        MOVE FROM tbl_buf2, WHEN DATA_OUT
        MOVE FROM tbl_buf3, WHEN DATA_OUT
        MOVE 1, ex_buf1, WHEN STATUS
        MOVE 1, ex_buf1, WHEN MSG_IN
        MOVE SCNTL2 & 0x7F to SCNTL2
        CLEAR ACK
        WAIT DISCONNECT
        JUMP All_done
Interrupt:
        INT Got_Selected
All_done:
        INT Command_Complete
```

NASM Output File Sections

The parts of the `.out` file discussed in this section correspond to the example SCRIPTS program in Figure 5-1.

SCRIPTS Array

The SCRIPTS array is an array of unsigned long values that is the actual contiguous machine code (opcodes) produced by the assembler. Each line of the array contains (from left to right) one instruction, and one or two address fields depending on the instruction. If a PROC directive is used in the source program, there may be more than one SCRIPTS array. For each PROC, a new array will be declared with the name specified with the PROC directive.

For example if the above code started with:

```
PROC SCSI_READ:
Start:
        SELECT ATN Select_ID, REL(Interrupt)
              .
              .
              .
```

Then the SCRIPTS array would have started:

```
typedef unsigned long ULONG;
ULONG SCSI_READ[] = {
   0x45020000L, 0x00000060L,
```

The default array name without the PROC statement is SCRIPT. The SCRIPTS array is not affected by NASM command line options.

Example of SCRIPTS array:

```
typedef unsigned long ULONG;
ULONG SCRIPT[] = {
   0x45020000L,0x00000060L,
   0x9E030000L,0x00000011L,
   0x0E000001L,0x00000000L,
   0x0A000006L,0x00000001L,
   0x18000000L,0x00000000L,
   0x18000000L,0x00000008L,
   0x18000000L,0x00000010L,
```

```
        0x0B000001L,0x00000000L,
        0x0F000001L,0x00000000L,
        0x7C027F00L,0x00000000L,
        0x60000040L,0x00000000L,
        0x48000000L,0x00000000L,
        0x80080000L,0x00000070L,
        0x98080000L,0x000000A5L,
        0x98080000L,0x00000001L

};
```

Entry and PROC

A PROC label generates separate arrays of SCRIPTS instructions for each PROC occurrence. An Entry specification generates a "C" language #define (pronounced "pound define") equal to the number of bytes between this entry and the beginning of the first code array. The #define offset is not relative to the array in which it appears, but is relative to the first code array created. In the example shown in Table 5-1, the first SCRIPTS instruction for INC_A is located 40 (hex) bytes after the location of MAIN[ ].

Table 5-1
Relationship Between Entry and PROC Statements and Output File

Source	Output File
	typedef unsigned long ULONG;
Entry MAIN	#define ENT_MAIN 0x00000000L
Entry CLEAR_A	#define ENT_CLEAR_A 0x00000018L
Entry INC_A	#define ENT_INC_A 0x00000040L
PROC MAIN:	ULONG MAIN[] = {
call CLEAR_A	0x88080000L, 0x00000018L,
call INC_A	0x88080000L, 0x00000040L,
call INC_A	0x88080000L, 0x00000040L,
	};
PROC CLEAR_A:	ULONG CLEAR_A[] = {
move SCRATCHA0 & 00 to SCRATCHA0	0x7C340000L, 0x00000000L,
move SCRATCHA1 & 00 to SCRATCHA1	0x7C350000L, 0x00000000L,
move SCRATCHA2 & 00 to SCRATCHA2	0x7C360000L, 0x00000000L,
move SCRATCHA3 & 00 to SCRATCHA3	0x7C370000L, 0x00000000L,
return;	0x90080000L, 0x00000000L
INC_A:	
move SCRATCHA0 + 1 to SCRATCHA0	0x7E340100L, 0x00000000L,
return, if NOT Carry;	0x90200000L, 0x00000000L,
move SCRATCHA1 + 1to SCRATCHA1	0x7E350100L, 0x00000000L,
return, if NOT Carry;	0x90200000L, 0x00000000L,
move SCRATCHA2 + 1 to SCRATCHA2	0x7E360100L, 0x00000000L,
return, if NOT Carry;	0x90200000L, 0x00000000L,
move SCRATCHA3 + 1 to SCRATCHA3	0x7E370100L, 0x00000000L,
return;	0x90080000L, 0x00000000L
	};

External

The External section contains the external variable records, if any were declared. First, is the External Header Record which contains:

```
#define Ext_count count
```

where count is defined to be the number of external variables. Second is a character array of all external names used:

```
char *External_Names[Ext_Count] = {
   “dsa_storage”,
   “in_offset”,
   “out_offset”
   };
```

Third is a list of External Contents Records:

```
#define E_name offset
```

where name is the name of the variable and offset is defined to be the byte offset from the beginning of the data area (this is always zero for externals).

Following this is an array of unsigned longs named by appending “_Used” to the variable name. This array is a list of dword offsets from the beginning of the SCRIPTS array where the variable is used and should be patched.

```
#define E_name_Used offset
```

The last two sections (External Contents Record and Offset Array) of the External record are repeated for every External defined in the SCRIPT.

Effect of Command Line Switches

If the -o compiler option is used then all items mentioned above are included in the output file. If the -p (partial ‘C’ output) option is used then the External Header Record and Character Array are omitted from the output file. An example of the output generated using each compiler option is listed below.

Example:

Using -o assembler option:

```
#define Ext_Count  2
char *External_Names[Ext_Count] = {
   “ex_buf2”,
   “ex_buf1”
};
```

```
#define E_ex_buf1 0x00000000L
ULONG E_ex_buf1_Used[] = {
   0x0000000FL,
   0x00000011L
};
```

Using `-p` assembler option:

```
ULONG E_ex_buf1_Used[] = {
   0x0000000FL,
   0x00000011L
};
```

Relative

The Relative section contains the relative buffer records, if any were declared. The first part is the Relative Header Record, which contains:

```
#define Rel_Count count
```

where `count` is a total count of all the uses of all the Relative buffers in the SCRIPTS program. For example, in the SCRIPTS example above, rel_buf1 and rel_buf2 are each used once so `Rel_Count` is #defined to 2, indicating that there were two uses of Relative buffers in the SCRIPTS code.

The second part of the Relative record is the Relative Patch Array which contains:

```
ULONG Rel_Patches[Rel_Count] = {
   Rel_Offset1,
   Rel_Offset2,
   Rel_Offset3,
   .
   .
   Rel_Offsetn
};
```

where Rel_Offsetx is an offset into the SCRIPTS array where a Relative buffer is used. This array, along with the Relative Header Record, can be used to patch all Relative buffers in a SCRIPTS program. Please see the section on patching SCRIPTS instructions for more information on how to do this.

The third part of the Relative record is the Relative Buffer Record, which contains:

```
#define R_name offset
```

where name is the name of the Relative buffer (i.e. rel_buf1) and offset is the relative offset of this buffer from the beginning of the entire Relative buffer. For example, in the above SCRIPTS example rel_buf2 has an offset of 0x00000001L, indicating that it starts one byte from the beginning of the Relative buffer.

The final part of the Relative record is the offset array which lists the dword offsets in the SCRIPTS array where each individual relative buffer is used. It is the same as the offset array used for External buffers, except that the array names are of the format R_name_Used where name is the name of the individual relative buffer.

```
#define R_name_Used offset
```

The last two sections (Relative Buffer Record and Offset Array) of the Relative record are repeated for every Relative defined in the SCRIPTS program.

Effect of Command Line Switches

If the -o compiler option is used then all items mentioned above are included in the output file. If the -p (partial 'C' output) option is used, the Relative Header Record and Relative Patch Array are omitted from the output file. An example of the output generated using each compiler option is listed below.

Example:

Using -o assembler option:

```
#define Rel_Count 2
ULONG Rel_Patches[Rel_Count] = {
   0x00000007L,
   0x00000005L
};
#define R_rel_buf1 0x00000000L
ULONG R_rel_buf1_Used[] = {
   0x00000005L
};

#define R_rel_buf2 0x00000001L
ULONG R_rel_buf2_Used[] = {
   0x00000007L
};
```

Using -p assembler option:

```
#define R_rel_buf1 0x00000000L
ULONG R_rel_buf1_Used[] = {
    0x00000005L
};
#define R_rel_buf2 0x00000001L
ULONG R_rel_buf2_Used[] = {
    0x00000007L
};
```

Entry

The ENTRY section contains the entry records, if any were declared. An entry record is a #define of the entry name prefixed with Ent_, defined to be a byte offset into the SCRIPTS array.

Example:

Using -o or -p assembler option:

```
#define Ent_Send_CMD          0x00000018L
#define Ent_Send_DATA         0x00000020L
#define Ent_Start             0x00000000L
```

The labels defined as entries are the only ones that will be available to the driver code. The "C" code examples in Chapter 7 are examples of how the driver can use this information to start SCRIPTS routines at any location defined as an entry. The ENTRY section is not affected by NASM command line options.

Label Patches

The Label Patches section contains the label patch records. A label patch record is an array of locations that are referred to by an absolute Transfer Control instruction. These locations are the dword offsets into the SCRIPTS array. The offsets are used to patch in the physical addresses at run time. Please see the section on patching SCRIPTS in Chapter 7 for more information on how to patch absolute jump instructions. The Label Patches section is not affected by NASM command line options.

Example:

Using -o or -p assembler option:

```
ULONG LABELPATCHES[] = {
    0x00000019L
                        };
```

Absolute

The Absolute section contains the Absolute records, if any were declared. First is the Absolute Header Record, which contains:

```
#define Abs_Count count
```

where count is the number of Absolutes defined in the SCRIPTS program.

The second section is the Character Array of all ABSOLUTE names used, it contains:

```
char *Absolute_Names[Abs_Count] = {
    Abs_String1,
    Abs_String1,
    .
    .
    Abs_Stringn
};
```

where Abs_Stringx is the name of the Absolute being defined.

Third is the Absolute Value Definition, which contains:

```
#define A_name value
```

where name is the name of the Absolute and value is the value assigned to this Absolute in the SCRIPTS program.

The final part of the ABSOLUTE record is the Offset Array, which lists the offsets in the SCRIPTS array where each ABSOLUTE is used. It is the same as the offset array used for External buffers, except that the array names are of the format A_name_Used where name is the name of the ABSOLUTE.

The last two sections (Absolute Value Definition and Offset Array) of the ABSOLUTE record are repeated for every ABSOLUTE defined in the SCRIPTS program.

Effect of Command Line Switches

If the -o compiler option is used, then all items mentioned above are included in the output file. If the -p (partial 'C' output) option is used, then the Offset Array is omitted from the output file. An example of the output generated using each compiler option is listed below.

Using -o assembler option:

```
#define Abs_Count 4
char *Absolute_Names[Abs_Count] = {
    "Command_Complete",
    "Got_Selected",
    "Not_Msg_Out",
```

```
    "Select_ID"
};
#define A_Command_Complete 0x00000001L
ULONG A_Command_Complete_Used[] = {
    0x0000001DL
};
#define A_Select_ID 0x00000002L
ULONG A_Select_ID_Used[] = {
    0x00000000L
};

#define A_Not_Msg_Out 0x00000011L
ULONG A_Not_Msg_Out_Used[] = {
    0x00000003L
};
#define A_Got_Selected 0x000000A5L
ULONG A_Got_Selected_Used[] = {
    0x0000001BL
};
```

Using `-p` assembler option:

```
#define A_Command_Complete 0x00000001L
#define A_Select_ID 0x00000002L
#define A_Not_Msg_Out 0x00000011L
#define A_Got_Selected 0x000000A5L
```

Termination Record

The module termination record declares two variables, INSTRUCTIONS and PATCHES. INSTRUCTIONS is assigned the number of instructions found in the SCRIPTS program, and PATCHES is assigned the number of label patches. If the `-o` compiler option is used, then all items mentioned above are included in the output file. If the `-p` (partial 'C' output) option is used, then the Patches variable is omitted from the output file. If the `-u` (exclude module termination record) is used, then both variables are omitted from the output file. An example of the output generated using each compiler option is listed below.

Using `-o` assembler option:

```
ULONG INSTRUCTIONS= 0x0000000EL;
ULONG PATCHES= 0x00000000L;
```

Using `-p` assembler option:

```
ULONG INSTRUCTIONS= 0x0000000EL;
```

Chapter 6

Using the Registers to Control Chip Operations

Overview

The SYM53C8XX is initialized by setting and clearing bits in the operating registers. This chapter lists the SYM53C8XX registers, grouped by function. The register descriptions provide an overview of the aspects of chip operation that are controlled in each register. Appendix B lists all of the operating registers and bits in the SYM53C8XX processors by hexadecimal address. The SYM53C8XX also has a set of PCI Configuration registers, but they are not described in this document since they are initialized by the system, not by the SCSI driver program. Full definitions of these registers, as well as the individual bits in the operating registers, can be found in the SYM53C8XX data manuals.

SCSI Registers

The SCSI registers are used for the following functions:

- performing SCSI operations by low-level, register-oriented programming
- obtaining data for debugging, such as checking the signal status of the SBCL and SBDL registers to determine exactly what is on the SCSI bus at the time the registers are read
- obtaining SCSI interrupt status, which is contained in the SIST0, and SIST1 registers
- initialization of the SCSI interface, for example, parity generation and checking on the SCSI bus
- enabling or masking SCSI interrupts in the SIEN registers

Table 6-1
SYM53C8XX SCSI Registers

Name	Definition	Functions
SIEN1	SCSI Interrupt Enable 1	interrupt mask bits for selection/reselection time-out, general purpose time-out, handshake to handshake time-out
SIEN0	SCSI Interrupt Enable 0	interrupt mask bits for phase mismatch, SATN/, function complete, selection/reselection, gross error, unexpected disconnect, SCSI reset, parity error
SDID	SCSI Destination ID	encoded destination SCSI ID
SCNTL3	SCSI Control 3	clock conversion factor bits, enable wide SCSI, enable Ultra SCSI or Ultra2 SCSI
SCNTL2	SCSI Control 2	wide SCSI control bits, vendor unique enhancements; DIFFSENS mismatch indicator (53C895 only)
SCNTL1	SCSI Control 1	add an extra clock cycle of setup to each SCSI data transfer; disable halt on parity error; Connected bit; parity bits; Immediate Arbitration bit
SCNTL0	SCSI Control 0	arbitration mode bits; enable parity checking
SOCL	SCSI Output Control Latch	testing SCSI control lines
SSID	SCSI Selector ID	the ID of the device that selected or reselected the SYM53C8XX
SODL	SCSI Output Data Latch	data flows through this register when sending data in any mode
SXFER	SCSI Transfer	define synchronous transfer period and synchronous offset
SCID	SCSI Chip ID	enable response to selection/reselection, set SCSI ID for SYM53C8XX
SBCL	SCSI Bus Control Lines	used to return SCSI control line status
SBDL	SCSI Bus Data Lines	contains SCSI data bus status
SIDL	SCSI Input Data Latch	contains latched data from the SCSI bus
SFBR	SCSI First Byte Received	contains the first byte received in any asynchronous information transfer phase
SSTAT2	SCSI Status 2	reports SIDL, SODR, SODL most significant byte full; parity detection, disconnect detection
SSTAT1	SCSI Status 1	FIFO flags; latched SCSI parity signal; latched SCSI phase status bits

Table 6-1 (Continued)
SYM53C8XX SCSI Registers

Name	Definition	Functions
SSTAT0	SCSI Status 0	SIDL, SODR, SODL least significant byte full; arbitration reporting bits; status of RST/ and SDP0/ signals
SLPAR	SCSI Longitudinal Parity	performs a bytewise longitudinal parity check on all SCSI data
SWIDE (wide SCSI products only)	SCSI Wide Residue Data	contains a residual data byte that was never sent across the DMA bus after wide SCSI operation
STIME0	SCSI Timer 0	selects handshake to handshake time-out period
STIME1	SCSI Timer 1	selects general purpose time-out period
RESPID0	Response ID 0	contains IDs the SYM53C8XX will respond to when it is selected or reselected
RESPID1 (wide SCSI products only)	Response ID 1	contains IDs the SYM53C8XX will respond to when it is selected or reselected
STEST4 (SYM53C895 only)	SCSI Test 4	contains DIFFSENS pin values that indicate the type of SCSI device connected to the bus; frequency lock bit for clock quadrupler
STEST3	SCSI Test 3	active negation enable; SCSI FIFO test read/write; Halt SCSI clock; Clear SCSI FIFO
STEST2	SCSI Test 2	clear synchronous offset; enable differential mode; wide SCSI; extend SREQ/-SACK/ filtering; low level mode enable
STEST1	SCSI Test 1	disable the external SCLK pin and use the PCI clock as the internal SCSI clock; enable the SCSI Clock doubler (SYM53C825A/875/ 876/885 only) or SCSI clock quadupler (SYM53C895 only)
STEST0	SCSI Test 0	these bits are used for low level operation and manufacturer testing, SCSI selected as ID

DMA Registers

The DMA registers are used for the following functions:

- setting up the host interface
- obtaining DMA interrupt status information contained in the DSTAT register
- obtaining DMA FIFO information, such as the number of bytes it contains
- enabling or masking DMA interrupts with the DIEN registers

Table 6-2
SYM53C8XX DMA Registers

Name	Definition	Functions
TEMP	Temporary Register	stores pointer to next SCRIPTS instruction to be executed when returning from a subroutine
DFIFO	DMA FIFO	may be used to determine the number of bytes in the DMA FIFO when an interrupt occurs, when used in conjunction with DBC
DCMD	DMA Command	identifies the instruction that the SYM53C8XX will execute
DBC	DMA Byte Counter	determines the number of bytes to be transferred in a Block Move instruction
DNAD	DMA Next Address	contains the general purpose address pointer
DSP	DMA SCRIPTS Pointer	contains the address of the next SCRIPTS instruction to be fetched. Placing an address in this register starts SCRIPTS
DSPS	DMA SCRIPTS Pointer Save	contains the second dword of a SCRIPTS instruction
DMODE	DMA Mode	defines burst length; near or far memory access; enables PCI read line command; manual start mode bit to prevent automatic execution of SCRIPTS
DCNTL	DMA Control	enable single step mode; SYM53C700 compatibility bit; enable PCI Cache Line Size register; enable instruction prefetching
DIEN	DMA Interrupt Enable	contains interrupt mask bits corresponding to master data parity error, bus fault, aborted, single step interrupt, SCRIPT interrupt instruction received, illegal instruction detected

SCRIPTS Registers

The SCRIPTS registers are used to hold the SCRIPTS instruction information which is fetched from host memory at run time by the SYM53C8XX.

Table 6-3
SYM53C8XX SCRIPTS Registers

Name	Definition	Functions
DCMD	DMA Command	identifies the instruction that the SYM53C8XX will execute
DBC	DMA Byte Counter	determines the number of bytes to be transferred in a Block Move instruction
DNAD	DMA Next Address	contains the general purpose address pointer
DSP	DMA SCRIPTS Pointer	contains the address of the next SCRIPTS instruction to be fetched; placing an address in this register starts SCRIPTS
DSPS	DMA SCRIPTS Pointer Save	contains the second dword of a SCRIPTS instruction
DSA	Data Structure Address	contains base address used for all table indirect calculations

Interrupt Registers

Interrupt registers contain interrupt status information. The DSTAT contains the DMA interrupt status information. The SIST0 and SIST1 contain SCSI interrupt status bits. The remaining registers contain interrupt enable bits. The ISTAT register can be polled for interrupts. It is the only register that can be accessed while SCRIPTS is running. Refer to Chapter 9 for more information on handling interrupts.

Table 6-4
SYM53C8XX Interrupt Registers

Name	Definition	Functions
ISTAT	Interrupt Status	interrupt polling; determine whether a SCSI or DMA interrupt has occurred; check for stacked interrupts; abort an operation; software reset; Signal Process bit; semaphore bit;interrupt on the fly bit;indicate SCSI interrupt pending (SYM53C885 only); SCSI bus mode change (53C895 only)
SIEN1	SCSI Interrupt Enable 1	interrupt mask bits for selection/reselection time-out, general purpose time-out, handshake to handshake time-out; wakeup (SYM53C885 only)SCSI bus mode change (53C895 only)
SIEN0	SCSI Interrupt Enable 0	interrupt mask bits for phase mismatch, SATN/, function complete, selection/reselection, gross error, unexpected disconnect, SCSI reset, parity error
SIST1	SCSI Interrupt Status 1	returns the status of the following interrupt conditions: selection/reselection time-out, general purpose timer expired, handshake to handshake timer expired; wakeup (SYM53C885 only)
SIST0	SCSI Interrupt Status 0	returns the status of the following interrupt conditions: phase mismatch (SATN/ active), function complete, selection/reselection, SCSI gross error, unexpected disconnect, SCSI RST/ received, parity error.
DIEN	DMA Interrupt Enable	contains interrupt mask bits corresponding to master data parity error, bus fault, aborted operation, single step interrupt, SCRIPTS interrupt instruction received, illegal instruction detected
DSTAT	DMA Status	reports sources of DMA interrupts: DMA FIFO empty, Master data parity error, bus fault, aborted, single step interrupt, SCRIPTS interrupt instruction received, illegal instruction detected

Test and Miscellaneous Registers

The test registers are used to test the DMA and SCSI FIFOs and perform other miscellaneous functions. The test registers can be used to decrement the byte count or increment the address count in the FIFOs.

Table 6-5
SYM53C8XX Test Registers

Name	Definition	Functions
CTEST3	Chip Test 3	revision level bits, flush/clear DMA FIFO,
CTEST2	Chip Test 2	data transfer direction; I/O or memory configuration; request/acknowledge status
CTEST1	Chip Test 1	DMA FIFO bits full or empty
CTEST6	Chip Test 6	writes data to the DMA FIFO
CTEST5	Chip Test 5	clock address incrementor; clock byte counter; DMA direction; control of set or reset pulses
CTEST4	Chip Test 4	burst disable; master parity error enable; DMA FIFO byte control
ADDER	Adder Sum Output	contains output of internal adder
CTEST0	Chip Test 0	used to enable power management modes in SYM53C885

General Purpose Registers

Table 6-6
SYM53C8XX General Purpose Registers

Name	Definition
GPREG	General Purpose
DWT/SBR	DMA Watchdog Timer/Scratch Byte Register
CTEST0	Chip Test 0
SCRATCHA	General Purpose Scratchpad A
SCRATCHB	General Purpose Scratchpad B
SCRATCHC-J	General Purpose Scratchpad C-J (SYM53C825A/53C875/53C876/ 53C885/53C895 only)
GPCNTL	General Purpose Control
MACNTL	Memory Access Control

Register Initialization

The startup register values are determined by a “C” program, written by the software developer, that can be loaded automatically by the device driver. The appropriate startup values for the register bits depend on the design of the individual system, so a single start-up algorithm will not support every application. The hardware default values for each bit are provided in the Register Summary, Appendix B; these default values are suitable for most applications.

This section lists the important register bits to consider when writing a startup program for a specific system. Although the startup program does not have to initialize all bits in the chip if the default values are acceptable, the bits in these lists affect features that should be enabled or disabled, or other decisions that should be made, when initializing the chip. For complete register and bit descriptions, refer to the SYM53C8XX data manuals. In addition, Chapter 2, “Functional Description,” in the product data manuals contains a section on the bits and registers that affect parity checking and generation. All reserved bits should be left cleared by the startup program.

Table 6-7
53C815/53C810A/53C860
Startup Bits

Register Address	Register Name	Bits	Remarks
00	SCNTL0	7-6, 3, 1-0	Bits 7-6: Arbitration Mode Bit 3: Enable Parity Checking Bit 1: Assert SATN/ on Parity Error Bit 0: Target Mode. Bit 0 may be set either at initialization or during SCRIPTS operation. Set it at startup if the chip will operate as a target only. If it will switch between target and initiator roles, use SCRIPTS to control this bit.
01	SCNTL1	7, 5	Bit 7: Extra Clock Cycle of Data Setup Bit 5: Disable Halt on Parity Error or SATN/ (for target mode only)
03	SCNTL3	7-4, 2-0	Bit 7: Ultra Enable (53C860 only) Bits 6-4: Synchronous Clock Conversion Factor Bits 2-0: Clock Conversion Factor
04	SCID	6-5, 2-0	Bit 6: Enable Response to Reselection Bit 5: Enable Response to Selection Bits 2-0: Encoded Chip SCSI ID
05	SXFER	all	Since the default operation for SCSI is asynchronous transfers, these bits should probably not be set until synchronous parameters are established between the initiator and target. Bits 7-5: Synchronous Transfer Period Bits 3-0: Max SCSI Synchronous Offset
10-13	DSA	all	must be initialized if you are using table indirect mode
1B	CTEST3	1-0	Bit 1: Fetch Pin Mode Bit 0: Write and Invalidate Enable (53C810A/53C860 only)
21	CTEST4	7, 3	Bit 7: Burst Disable Bit 3: Master Parity Error Enable
2C-2F	DSP	all	At the end of the initialization program, write the address of the first SCRIPTS instruction to this register to begin SCRIPTS execution.

Table 6-7
53C815/53C810A/53C860
Startup Bits (Continued)

Register Address	Register Name	Bits	Remarks
38	DMODE	7-2	Bits 7-6: Burst Length Bit 5: Source I/O-Memory Enable Bit 4: Destination I/O-Memory Enable Bit 3: Enable Read Line Bit 2: Enable Read Multiple (53C810A/53C860 only)
39	DIEN	6-2, 0	Bit 6: Master Data Parity Error Bit 5: Bus Fault Bit 4: Aborted Bit 3: Single Step Interrupt Bit 2: SCRIPTS Interrupt Instruction Received Bit 0: Illegal Instruction Detected
3B	DCNTL	7, 5-3, 0	Bit 7: Cache Line Size Enable Bit 5: Pre-fetch Enable (53C810A/53C860 only) Bit 4: Single-Step Mode Bit 3: IRQ Mode Bit 0: SYM53C700 Compatibility
40	SIEN0	all	Interrupt mask bits for: Bit 7: Phase Mismatch or SATN/ Bit 6: Function Complete Bit 5: Selected Bit 4: Reselected Bit 3: SCSI Gross Error Bit 2: Unexpected Disconnect Bit 1: SCSI Reset Condition Bit 0: SCSI Parity Error
41	SIEN1	2-0	Interrupt mask bits for: Bit 2: Selection or Reselection Time-Out Bit 1: General Purpose Timer Expired Bit 0: Handshake to Handshake Timer Expired
46	MACNTL	3-0	Initialize these when using the MAC_TSTOUT pin. These bits determine local or far access for the following operations: Bit 3: Data write Bit 2: Data read Bit 1: SCRIPTS pointers Bit 0: SCRIPTS fetches
48	STIME0	all	Bits 7-4: Handshake to Handshake Timer Period Bits 3-0: Selection Time-Out
49	STIME1	3-0	Bits 3-0: General Purpose Timer Period

Table 6-7
53C815/53C810A/53C860
Startup Bits (Continued)

Register Address	Register Name	Bits	Remarks
4A	RESPID	all	
4D	STEST1	7	Bit 7: SCLK
4E	STEST2	1	Bit 1: Extend SREQ/SACK Filtering
4F	STEST3	7	Bit 7: TolerANT Enable

Table 6-8
SYM53C825A/875/876/885/895
Startup Bits

Register Address	Register Name	Bits	Remarks
00	SCNTL0	7-6, 3, 1-0	Bits 7-6: Arbitration Mode Bit 3: Enable Parity Checking Bit 1: Assert SATN/ on Parity Error Bit 0: Target Mode. Bit 0 may be set either at initialization or during SCRIPTS operation. Set it at startup if the chip will operate as a target only. If it will switch between target and initiator roles, use SCRIPTS to control this bit.
01	SCNTL1	7, 5	Bit 7: Extra Clock Cycle of Data Setup Bit 5: Disable Halt on Parity Error or SATN/ (for target mode only)
03	SCNTL3	all	Bit 7: Ultra Enable (53C875/876/885/895 only) Bits 6-4: Synchronous Clock Conversion Factor Bits 2-0: Clock Conversion Factor
04	SCID	6-5, 3-0	Bit 6: Enable Response to Reselection Bit 5: Enable Response to Selection Bit 3: Enable Wide SCSI Bits 2-0: Encoded Chip SCSI ID
05	SXFER	all	Since the default operation for SCSI is asynchronous transfers, these bits should probably not be set until synchronous parameters are established between the initiator and target. Bits 7-5: Synchronous Transfer Period Bits 3-0: Max SCSI Synchronous Offset

Table 6-8
SYM53C825A/875/876/885/895
Startup Bits (Continued)

Register Address	Register Name	Bits	Remarks
10-13	DSA	all	must be initialized if you are using table indirect mode
1B	CTEST3	1-0	Bit 1: Fetch Pin Mode Bit 0: Write and Invalidate Enable
21	CTEST4	7, 3	Bit 7: Burst Disable Bit 3: Master Parity Error Enable
22	CTEST2	3	SCRATCHA/B operation (when SCRIPTS RAM is enabled)
18	CTEST0	2-0	Set the priority level for gaining access to the PCI bus (SYM53C885 only)
2C-2F	DSP	all	At the end of the initialization program, write the address of the first SCRIPTS instruction to this register to begin SCRIPTS execution.
38	DMODE	7-2	Bits 7-6: Burst Length Bit 5: Source I/O-Memory Enable Bit 4: Destination I/O-Memory Enable Bit 3: Enable Read Line Bit 2: Enable Read Multiple
39	DIEN	4-2, 0	Bit 4: Aborted Bit 3: Single Step Interrupt Bit 2: SCRIPTS Interrupt Instruction Received Bit 0: Illegal Instruction Detected
3B	DCNTL	7, 5-3, 0	Bit 7: Cache Line Size Enable Bit 5: Pre-fetch Enable Bit 4: Single-Step Mode Bit 3: IRQ Mode Bit 0: SYM53C700 Compatibility
40	SIEN0	all	Interrupt mask bits for: Bit 7: Phase Mismatch or SATN/ Bit 6: Function Complete Bit 5: Selected Bit 4: Reselected Bit 3: SCSI Gross Error Bit 2: Unexpected Disconnect Bit 1: SCSI Reset Condition Bit 0: SCSI Parity Error

Table 6-8
SYM53C825A/875/876/885/895
Startup Bits (Continued)

Register Address	Register Name	Bits	Remarks
41	SIEN1	4,2-0	Interrupt mask bits for: Bit 4: SCSI Bus Mode Change (53C895 only) Bit 2: Selection or Reselection Time-Out Bit 1: General Purpose Timer Expired Bit 0: Handshake to Handshake Timer Expired
46	MACNTL	3-0	Initialize these when using the MAC_TESTOUT pin. These bits determine local or far access for the following operations: Bit 3: Data write Bit 2: Data read Bit 1: SCRIPTS pointers Bit 0: SCRIPTS fetch
48	STIME0	all	Bits 7-4: Handshake to Handshake Timer Period Bits 3-0: Selection Time-Out
49	STIME1	3-0	Bits 3-0: General Purpose Timer Period
4A	RESPID0	all	
4B	RESPID1	all	
4D	STEST1	7, 3-2	Bit 7: SCLK Bits 3-2: SCSI Clock Doubler 1-0 (53C875 only)
4E	STEST2	5, 1	Bit 5: SCSI Differential Mode Bit 1: Extend REQ/ACK Filtering
4F	STEST3	7	Bit 7: TolerANT Enable

Chapter 7

Integrating SCRIPTS Programs Into “C” Language Drivers

Overview

This chapter demonstrates how assembled SCRIPTS programs are included in SCSI device drivers written in “C” language. The chapter provides examples of the “C” code used to execute many of the types of algorithms used by SCSI SCRIPTS. The chapter ends with an entire SCRIPTS source file, `general.ss`, in Figure 7-1. Figure 7-2 is the output file from the Symbios Logic Assembler, called `general.out`. The relationship between the source file and the output file is described in more detail in Chapter 5.

Initializing the SYM53C8XX

The following “C” code example shows how the SYM53C8XX accesses the operating registers at initialization. The SYM53C8XX can be memory- or I/O-mapped or both. The example functions in this section access the I/O mapped registers of the SYM53C8XX.

```
/*****************************************************

 Function: IORead8

   Purpose: To read a byte from an io port
   Input: IO address of byte to be read
   Output: byte read from io port
   Assumptions: That the IO port actually exists
   Restrictions: Although IO_Addr is defined as
                 a ULONG it must not exceed 16 bits
                 in length as this is the maximum
              IO address the X86 architecture can produce
   Other functions called: inportb to read the io port
*****************************************************/

UBYTE IORead8(ULONG IO_Addr)

{

   return (inportb((UINT) IO_Addr));

}
```

```
/*****************************************************

 Function: IOWrite8

   Purpose: To write a byte out to an IO port
   Input: Value to be written and IO port address
   Output: None
   Assumptions: That the IO port actually exists
   Restrictions: Although IO_Addr is defined as a
                    ULONG it must not exceed 16 bits
                    in length as this is the maximum IO
                 address the X86 architecture can produce
   Other functions called: outportb to write to the io
                              port

*****************************************************/

void IOWrite8(ULONG IO_Addr, UBYTE value)

{

   outportb((UINT) IO_Addr, value);

}

/*****************************************************

 Function: IORead32

   Purpose: To read a dword (32 bits) from an io port
   Input: IO address of dword to be read
   Output: dword read from io port
   Assumptions: That the IO port actually exists
   Restrictions: Although IO_Addr is defined as a
                    ULONG it must not exceed 16 bits in
                    length as this is the maximum IO
                 address the X86 architecture can produce
   Other functions called: none

*****************************************************/

ULONG IORead32(ULONG IO_Addr)

{

   ULONG result;

   asm

   {

      .386

      mov dx, [IO_Addr]

      in eax, dx

      mov [result], eax

   }

   return(result);

}
```

```
/*****************************************************

 Function: IOWrite32

   Purpose: To write a dword (32 bits) out to an IO port
   Input: Value to be written and IO port address
   Output: None
   Assumptions: That the IO port actually exists
   Restrictions: Although IO_Addr is defined as a
                    ULONG it must not exceed 16 bits in
                    length as this is the maximum IO
                 address the X86 architecture can produce
   Other functions called: none

*****************************************************/

void IOWrite32(ULONG IO_Addr, ULONG value)

{

   asm

   {

       .386

       mov dx, [IO_Addr]

       mov eax, [value]

       out dx, eax

   }

}
```

Resetting The SYM53C8XX

This example shows how to reset the SYM53C8XX by setting, then clearing, the Software Reset (SRST) bit in the ISTAT register. It executes a Read-Modify-Write for each register whose default value will be changed at reset.

```
* sets SRST(bit 6) */
IOWrite8(ISTAT, (IORead8(ISTAT) | 0x40));/

* clears SRST(bit 6) */
IOWrite8(ISTAT, (IORead8(ISTAT) & 0xBF))/
```

Table Indirect Operations

More information on Table Indirect operation and on creating a table is provided in Chapter 9.

Initializing a Table

The following example is a SCRIPTS declaration of a table. Although NASM does not actually generate any output based on the table declaration, it does place offsets into the SCRIPTS array based on the order of the buffers in the table declaration. The actual byte values and byte counts in the SCRIPTS instruction are not used at this stage, because NASM does not generate any output from the table declaration.

```
TABLE dsa_table \
    sendmsg = ??, \
    rcvmsg = ??, \
    cmd_adr = ??, \
    device = ID{??}, \
    status_adr = ??, \
    ext_buf = ??, \
    sync_in = ??, \
    data_adr = ??
```

Create Table Indirect Entry Offsets

The following “C” code example sets up a table that can be used with the SYM53C8XX table indirect addressing mode. Each entry in the table is a pair of 32-bit values. These entries reference the same buffers as the SCRIPTS code examples above. For more illustration on the relationship between these pieces of code, refer to the Table Indirect Addressing section in Chapter 8. For this SCRIPTS program to work correctly, the table must start on a dword boundary and the offset labels must be in the same order as in the SCRIPTS table declaration.

```
/* The following definition sets up a table that can be used
with the SYM53C8XX table indirect addressing mode. Each entry
in the table is a pair of 32-bit values. For the SCRIPTS
routine to work correctly the table MUST start on a word boundary
and the offset labels must be in the same order in the SCRIPTS
table declaration. */

enum offsets {
    SENDMSG = 0,
    RCVMSG,
    CMD_ADR,
    DEVICE,
    STATUS_ADR,
    EXT_BUF,
    SYNC_IN,
    DATA_ADR, /* DATA_ADR must be last buffer.
};
```

Defining the Table Structure

The following code defines a data structure with two fields, a count and an address, which correspond to one element in the DSA table. A type is then defined and a pointer to a variable of this type is also defined. This pointer and the enumerated offsets defined above will then be used to access specific elements of the table. This example defines the table structure, but no space has been allocated yet in memory.

Example of data structure and type definition:

```
struct_table {
    uquad   count;
    uquad   address;
};
typedef struct_table table:
```

Declaring a Pointer to the Table

```
extern table   *buffer_table;
```

Allocating Memory for the Table

```
int init_table(void)
{
    UBYTE *buf_ptr;   /* temp ptr to ti tables */
    /* allocate space for table */
    buf_ptr =  (UBYTE far *) malloc((TABLE_SIZE
                                    sizeof(ti_entry))+ 4);
    /* did we get the memory */
    if (buf_ptr == NULL) return(COMMANDFAILED);

    /* dword align the table buffer, ByteAlignBuffer does
       this */
    dsa_table = (ti_entry *) ByteAlignBuffer(buf_ptr, 0);

   /* This initializes the DSA register to point to the buffer
      table that was allocated above*/
   IOWrite32(PCIDeviceIOBase+DSA, getPhysAddr(dsa_table));
    return(GOOD);
}
```

Using a Table

The following example creates two buffers (`identify_msg` and `test_unit_ready_cmd`). The byte counts and addresses for these buffers are then loaded into the `CMD_ADR` and `SENDMSG` elements of the `DSA_table` array. These examples define a message and a command buffer in the desired table, and loads the bytes into the table. The enumerated types are used in the Test Unit Ready example to index into the table.

```
static ubyte identify_msg[] = {
    0xc0           /* 0xc0 = allow disconnect, 0x80 = no
                   ** disconnect */
};
static ubyte test_unit_ready_cmd[] = {
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00
};
/* drive is the destination ID for the I/O*/
DSA_table[DEVICE].count=(ULONG)drive<< 16;
DSA_table[CMD_ADR].count = sizeof(test_unit_ready_cmd);
DSA_table[CMD_ADR].address=getPhyAddr(test_unit_ready_cmd;

DSA_table[SENDMSG].count = 1;
DSA_table[SENDMSG].address = getPhysAddr(identify_msg);

DSA_table[STATUS_ADR].count = 1;
DSA_table[STATUS_ADR].address = getPhysAddr(status);

DSA_table[RCVMSG].count = 1;
DSA_table[RCVMSG].address = getPhysAddr(msg_in);
```

Patching

Sometimes it is necessary for the “C” code to modify some elements of the SCRIPTS array once buffers have been allocated. This is called patching. Patching is required when relative transfer control instructions or table indirect addressing are not used. However, most applications will take advantage of these features, so patching is not often required. When patching is necessary, the general format of the patch in “C” is `SCRIPT[patch_offset] = patch_value;`

When only part of the 32-bit value in the SCRIPTS array must be modified, a Read-Modify-Write can be used. The format for this type of operation is `SCRIPT[patch_offset]|= patch_value;`. Any arithmetic or logical operator can be used in place of the logical or (|) symbol to make the desired modification.

The `patch_offset` is an index into the SCRIPTS array where the patch must be made. This value is usually obtained from one of the sections of the NASM output file. Please see Chapter 5 for more information on the NASM output file and the patch offsets it contains.

The `patch_value` is usually either a buffer physical address or a byte count, but could be anything that modifies the part of the SCRIPTS program.

The remainder of this section contains patching techniques for various instructions and buffer types that require modification at run time. Please note that this chapter only describes the most common types of patches. Other types of patching can generally be used to modify any part of a SCRIPTS instruction by using the ENTRY point patching method described in this section.

EXTERN Buffers

1. Create a buffer in ‘C’ statically or dynamically if necessary

   Example: `UCHAR msgin_buf[4];`

2. Patch the SCRIPT wherever this buffer is used, with the patch array generated by NASM

   Example: `SCRIPT[E_ex_buf1_Used[1]] = VirttoPhys(msgin_buf);`

See Chapter 5 for more information on the `_Used` patch array.

RELATIVE Buffers

RELATIVE buffers are essentially the same as External buffers. The SCRIPTS output file contains some additional information to aid in patching the SCRIPTS instructions. The individual relative buffer offset is encoded into the SCRIPTS instruction

Procedure 1

1. Create a buffer to hold all the individual relative buffers

   Example: `UCHAR rel_buffer[8]`

2. Patch the SCRIPTS array using the Patch array generated by NASM

   Example: `SCRIPT[R_rel_buf2_Used[0]] += VirttoPhys(rel_buffer)`

Procedure 2

1. Create a buffer to hold all the individual relative buffers

   Example: `UCHAR rel_buffer[8]`

2. All buffers can be patched in one loop if the main Patch array is accessed and the Header record is used. The -o assembler option must be used for this procedure to work.

   Example: `for(i=0; i<Rel_Count; i++) {`

```
SCRIPT[Rel_Patches[i]] += VirttoPhys(rel_buffer);
}
```

See Chapter 5 for more information on the structures created for patching relative buffers.

ABSOLUTE Values

ABSOLUTE values are patched exactly like EXTERN buffers. The `-o` compiler option must be used to patch Absolutes. See Chapter 5 for more information on ABSOLUTE values

Buffer Addresses

Buffer addresses are usually patched into Block Move, Memory to Memory, or Load/Store instructions. They are usually defined as EXTERNS, RELATIVES, or ABSOLUTES. The general format of this type of patch is:

`SCRIPT[X_buffername_Used[n]] = VirttoPhys(c_buffer);`

Where `X` is either E (Extern), R (Relative), or A (Absolute) depending on the type of buffer used.

`n` is the nth occurrence of this buffer in the SCRIPTS program

`c_buffer` is a buffer/array defined in ‘C’

See Chapter 5 for more information on the `_Used` array.

Byte Counts

Byte counts are usually patched into Block Move, Memory to Memory, or Load/Store instructions. Since the byte count is usually encoded in the first dword along with the opcode, be sure to OR in the byte count instead of doing a straight assignment. Byte counts to be patched are usually defined as EXTERNS, RELATIVES, or ABSOLUTES.The general format of this type of patch is:

```
SCRIPT[X_bytecount_Used[n]] |= c_byte_count;
```

Where `X` is either E, R, or A

`n` is the nth occurrence of this byte count in the SCRIPTS program

`c_byte_count` is a variable/constant byte count value

See Chapter 5 for more information on the `_Used` array.

Absolute JUMP/CALL Addresses

Use the LABELPATCHES array to patch in absolute JUMP or CALL addresses. The absolute offset from the beginning of the SCRIPTS instruction is encoded in the JUMP instruction at assembly. All that needs to be added is the base physical address of the SCRIPTS array. The general format of this type of patch is:

```
SCRIPT[LABELPATCHES[n]] += VirttoPhys(SCRIPT);
```

Where `n` is the nth jump instruction to be patched.

This can be automated using a loop and the PATCHES values.

See Chapter 5 for more information on the LABELPATCHES array.

Entry Locations

Entry offsets are byte offsets, not dword offsets. Divide the Entry offset by 4 to get to a SCRIPTS instruction offset. This method can be used to modify any SCRIPTS instruction that normally does not need patched, but needs to be modified in a special circumstance. The general format of this type of patch is:

```
SCRIPT[Ent_entrylabel/4 + n] = value;
```

Where `n` is either 0, 1 or 2 depending on the particular dword of the instruction that needs to be accessed.

If the first dword of an instruction is being accessed, you may need to do a Read-Modify-Write instruction to maintain the opcode.

See Chapter 5 for more information on the `Ent_ offsets`.

Self Modifying SCRIPTS Code

It is sometimes necessary to create self modifying SCRIPTS code for various reasons. When creating self modifying SCRIPTS code it should be done in such a way that external patching is only necessary at initialization time. Self modifying code can be accomplished by using either a Memory to Memory Move instruction or a combination of LOAD and STORE instructions. The following SCRIPTS example shows a Memory to Memory Move modifying a

Move Register instruction such that an offset can be added to a base address for jumping into a table.

```
ENTRY Patch_label1
ENTRY Patch_label2

EXTERN SCRATCHA1_addr
EXTERN SCRATCHB_addr

MOVE MEMORY 4, Patch_label2+4, SCRATCHB_addr
MOVE MEMORY 1, SCRATCHA1_addr, Patch_label1+1

Patch_label1:
MOVE SCRATCHB0 + 0 to SCRATCHB0
MOVE SCRATCHB1 + 0 to SCRATCHB1 WITH CARRY
MOVE SCRATCHB2 + 0 to SCRATCHB2 WITH CARRY
MOVE SCRATCHB3 + 0 to SCRATCHB3 WITH CARRY

MOVE MEMORY 4, SCRATCHB_addr, Patch_label2+4

Patch_label2:
JUMP REL(Jump_Table)
.
.
Jump_Table:
```

Patches to the SCRIPTS Instruction

Patch the Labels in the memory to memory move instructions first:

```
for (i=0; i<PATCHES; i++) {
   SCRIPT[LABELPATCHES[i]] += VirttoPhys(SCRIPT);
}
```

Next patch Scratch register physical addresses:

```
SCRIPT[E_SCRATCHA1_addr_Used[0]] =
VirttoPhys(chip_reg[ScratchA]) + 1;

SCRIPT[E_SCRATCHB_addr_Used[0]] =
VirttoPhys(chip_reg[ScratchB]);

SCRIPT[E_SCRATCHB_addr_Used[1]] =
VirttoPhys(chip_reg[ScratchB]);
```

These are the only patches required. LOAD and STORE instructions could be used to replace the Memory to Memory Move instructions.

> Note: SCRATCHA1 is used instead of SCRATCHA0 due to the alignment requirements of the Memory to Memory Move instruction.

Running a SCRIPTS Program

The SCRIPTS program is ready to run after all Command, Data, and Message buffers have been set up for the I/O. Start the SCRIPTS program by writing the physical address of the program to the DSP register. The superscript numerals refer to portions of the sample code in Figure 7-1 and Figure 7-2.

The entry points named in this example are all different points where SCRIPTS instructions could start.

```
static uquad start_offset[] = {
   Ent_init_siop[3], Ent_start_up[4], Ent_switch[5]
    };
```

This example starts the SCRIPTS program:

```
IOWrite32(PCIDeviceIOBase+DSP, getPhysAddr(script) +
                                        start_offset[mode]);
```

In this example, `mode = 0` begins at `init_siop` label, `mode = 1` begins at `start_up`, and `mode = 2` begins at the `switch label`.

Figure 7-1
SCRIPTS Source File

```
; Single-threaded general purpose SCRIPTS routine

; Offset for counts and addresses in the table
TABLE dsa_table \
sendmsg = ??, \
rcvmsg = ??, \
cmd_adr = ??, \
device = ID{??}, \
status_adr = ??, \
ext_buf = ??, \
sync_in = ??, \
data_adr = ??

; The SCRIPTS routine has finished initializing the SIOP.
Absolute done_init = 0x01

ABSOLUTE ok = 0x00
ABSOLUTE err1 = 0x0ff01
ABSOLUTE err2 = 0x0ff02
ABSOLUTE err3 = 0x0ff03
ABSOLUTE err4 = 0x0ff04
ABSOLUTE err5 = 0x0ff05
ABSOLUTE err6 = 0x0ff06
ABSOLUTE err7 = 0x0ff07
ABSOLUTE err8 = 0x0ff08
ABSOLUTE err9 = 0x0ff09

EXTERN dsa_storage, out_offset, in_offset

; SCSI I/O entry point. This address must be loaded into the
; SIOP before initiating a SCSI I/O.
ENTRY init_siop
ENTRY start_up
ENTRY switch
ENTRY datain
```

```
   ENTRY dataout

3  init_siop:
   INT done_init

4  start_up:

   SELECT ATN FROM device, REL(resel)

   ; Every phase comes back to here.
5  switch:
      JUMP REL(msgin), WHEN MSG_IN
      JUMP REL(msgout), IF MSG_OUT
      JUMP REL(command_phase), IF CMD
      JUMP REL(dataout), IF DATA_OUT
      JUMP REL(datain), IF DATA_IN
      JUMP REL(end), IF STATUS

      INT err1

   msgin:
      MOVE FROM rcvmsg, WHEN MSG_IN
      JUMP REL(ext_msg), IF 0x01
      JUMP REL(disc), IF 0x04
      CLEAR ACK
      JUMP REL(switch), IF 0x02 ; ignore save data pointers
      JUMP REL(switch), IF 0x07 ; ignore message reject)
      JUMP REL(switch), IF 0x03 ; ignore restore data pointers
      INT err2

   ext_msg:
      CLEAR ACK
      MOVE FROM ext_buf, WHEN MSG_IN
      JUMP REL(sync_msg), IF 0x03

      INT err3
```

```
sync_msg:
   CLEAR ACK
   MOVE FROM sync_in, WHEN MSG_IN
   CLEAR ACK
   JUMP REL(switch)

disc:
   MOVE SCNTL2 & 0x7f to SCNTL2 ;expect disconnect
   CLEAR ACK
   WAIT DISCONNECT
   WAIT RESELECT REL(select_adr)

   INT err4, WHEN NOT MSG_IN
   MOVE FROM rcvmsg, WHEN MSG_IN
   CLEAR ACK
   INT err9
   JUMP REL(switch)

msgout:
   MOVE FROM sendmsg, WHEN MSG_OUT
   JUMP REL(switch)

command_phase:
   MOVE FROM cmd_adr, WHEN CMD
   JUMP REL(switch)

; After every data transfer add 8 to data_adr. This allows
; scatter/gather operations when the list of addresses to
; read or write is appended to the end of the buffer_table.

1  dataout:
   MOVE FROM data_adr, WHEN DATA_OUT
   MOVE MEMORY 4, out_offset, scratch_adr
   CALL REL(addscratch)
   MOVE MEMORY 4, scratch_adr, out_offset
   JUMP REL(switch)
```

```
2  datain:
       MOVE FROM data_adr, WHEN DATA_IN
       MOVE MEMORY 4, in_offset, scratch_adr
       CALL REL(addscratch)
       MOVE MEMORY 4, scratch_adr, in_offset
       JUMP REL(switch)

   addscratch:
       MOVE SCRATCHA0 + 8 to SCRATCHA0
       MOVE SCRATCHA0 to SFBR
       JUMP REL(ck_carry), IF 0x00
       RETURN

   ck_carry:
       MOVE SCRATCHA1 + 1 to SCRATCHA1
       RETURN

   end:
       MOVE FROM status_adr, WHEN STATUS
       INT err5, WHEN NOT MSG_IN
       MOVE FROM rcvmsg, WHEN MSG_IN
       MOVE SCNTL2 & 0x7f to SCNTL2 ;expect disconnect
       CLEAR ACK
       WAIT DISCONNECT
       INT ok

   resel:
       INT err6

   select_adr:
       INT err7
```

Figure 7-2
NASM Output File

```
typedef unsigned long ULONG;

ULONG SCRIPT[] = {
    0x98080000L,0x00000001L,
    0x47000018L,0x000001E8L,
    0x878B0000L,0x00000030L,
    0x868A0000L,0x000000F0L,
    0x828A0000L,0x000000F8L,
    0x808A0000L,0x00000100L,
    0x818A0000L,0x00000128L,
    0x838A0000L,0x00000180L,
    0x98080000L,0x0000FF01L,
    0x1F000000L,0x00000008L,
    0x808C0001L,0x00000030L,
    0x808C0004L,0x00000068L,
    0x60000040L,0x00000000L,
    0x808C0002L,0xFFFFFFA0L,
    0x808C0007L,0xFFFFFF98L,
    0x808C0003L,0xFFFFFF90L,
    0x98080000L,0x0000FF02L,
    0x60000040L,0x00000000L,
    0x1F000000L,0x00000028L,
    0x808C0003L,0x00000008L,
    0x98080000L,0x0000FF03L,
    0x60000040L,0x00000000L,
    0x1F000000L,0x00000030L,
    0x60000040L,0x00000000L,
    0x80880000L,0xFFFFFF48L,
    0x7C027F00L,0x00000000L,
    0x60000040L,0x00000000L,
    0x48000000L,0x00000000L,
    0x54000000L,0x00000118L,
    0x9F030000L,0x0000FF04L,
    0x1F000000L,0x00000008L,
    0x60000040L,0x00000000L,
```

```
    0x98080000L,0x0000FF09L,
    0x80880000L,0xFFFFFF00L,
    0x1E000000L,0x00000000L,
    0x80880000L,0xFFFFFEF0L,
    0x1A000000L,0x00000010L,
    0x80880000L,0xFFFFFEE0L,
    0x18000000L,0x00000038L,
    0xC0000004L,0x00000000L,0x000DFE34L,
    0x88880000L,0x00000044L,
    0xC0000004L,0x000DFE34L,0x00000000L,
    0x80880000L,0xFFFFFEB0L,
    0x19000000L,0x00000038L,
    0xC0000004L,0x00000000L,0x000DFE34L,
    0x88880000L,0x00000014L,
    0xC0000004L,0x000DFE34L,0x00000000L,
    0x80880000L,0xFFFFFE80L,
    0x7E340800L,0x00000000L,
    0x72340000L,0x00000000L,
    0x808C0000L,0x00000008L,
    0x90080000L,0x00000000L,
    0x7E350100L,0x00000000L,
    0x90080000L,0x00000000L,
    0x1B000000L,0x00000020L,
    0x9F030000L,0x0000FF05L,
    0x1F000000L,0x00000008L,
    0x7C027F00L,0x00000000L,
    0x60000040L,0x00000000L,
    0x48000000L,0x00000000L,
    0x98080000L,0x00000000L,
    0x98080000L,0x0000FF06L,
    0x98080000L,0x0000FF07L

};
```

3

```
#define Ext_Count
char *External_Names[Ext_Count] = {
    “dsa_storage”,
    “in_offset”,
```

```
    “out_offset”
};

#define E_in_offset 0x00000000L
ULONG E_in_offset_Used[] = {
    0x0000005BL,
    0x00000061L
};

#define E_out_offset 0x00000000L
ULONG E_out_offset_Used[] = {
    0x0000004FL,
    0x00000055L
};

#define Abs_Count 11
char *Absolute_Names[Abs_Count] = {
    “done_init”,
    “err2”,
    “err1”,
    “err3”,
    “err4”,
    “err5”,
    “err6”,
    “err7”,
    “err9”,
    “ok”,
    “scratch_adr”
};

#define A_ok 0x00000000L
ULONG A_ok_Used[] = {
    0x0000007DL
};

#define A_done_init 0x00000001L
ULONG A_done_init_Used[] = {
    0x00000001L
};
```

```
#define A_err1 0x0000FF01L
ULONG A_err1_Used[] = {
    0x00000011L
};

#define A_err2 0x0000FF02L
ULONG A_err2_Used[] = {
    0x00000021L
};

#define A_err3 0x0000FF03L
ULONG A_err3_Used[] = {
    0x00000029L
};

#define A_err4 0x0000FF04L
ULONG A_err4_Used[] = {
    0x0000003BL
};

#define A_err5 0x0000FF05L
ULONG A_err5_Used[] = {
    0x00000073L
};

#define A_err6 0x0000FF06L
ULONG A_err6_Used[] = {
    0x0000007FL
};

#define A_err7 0x0000FF07L
ULONG A_err7_Used[] = {
    0x00000081L
};
```

```
#define A_err9 0x0000FF09L
ULONG A_err9_Used[] = {
   0x00000041L
};

#define A_scratch_adr 0x000DFE34L
ULONG A_scratch_adr_Used[] = {
   0x00000050L,
   0x00000054L,
   0x0000005CL,
   0x00000060L
};

2  #define Ent_datain 0x00000160L
1  #define Ent_dataout 0x00000130L
3  #define Ent_init_siop 0x00000000L
4  #define Ent_start_up 0x00000008L
5  #define Ent_switch 0x00000010L

ULONG INSTRUCTIONS= 0x0000003FL;
ULONG PATCHES= 0x00000000L;
```

Chapter 8

Writing Device Drivers With SCRIPTS

Overview

The architecture of a SCSI system may be viewed in layers, with each layer providing data to the layers immediately above and below. The device driver interfaces between the host operating system and the SYM53C8XX hardware and firmware. The device driver, host operating system, and all applications reside in the host computer's main memory. The SYM53C8XX is a separate hardware component, but has direct access to host memory. Figure 8-1 shows the relationship of the device driver to other parts of the SCSI system.

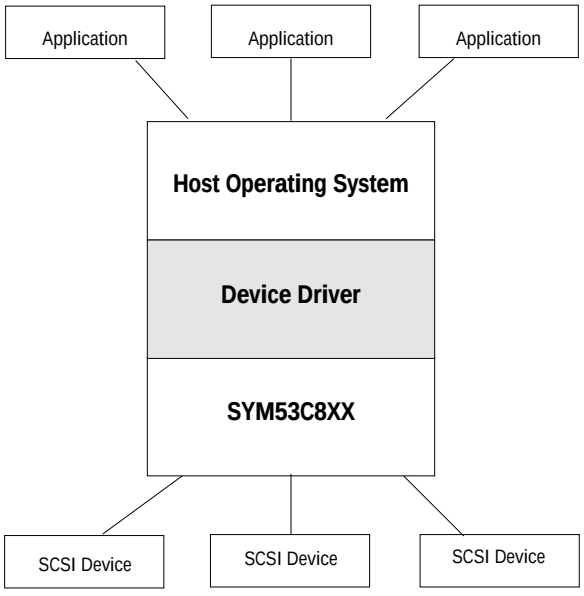


Figure 8-1
The Role of the SCSI Device Driver

The device driver itself contains two layers, illustrated in Figure 8-2. The top layer is the operating system interface. It accepts and interprets I/O requests from the host operating system. These requests may vary, depending on the type and vendor of the SCSI device. The formatted requests are passed to the hardware interface, or lower layer of the driver. The operating system interface must also schedule SCSI bus accesses when more than one device is active. It schedules the I/O requests, and tracks the completed and outstanding I/Os, based on status passed back from the hardware interface. The SCRIPTS program is compiled with the driver program, and is loaded into host memory when the device driver program starts.

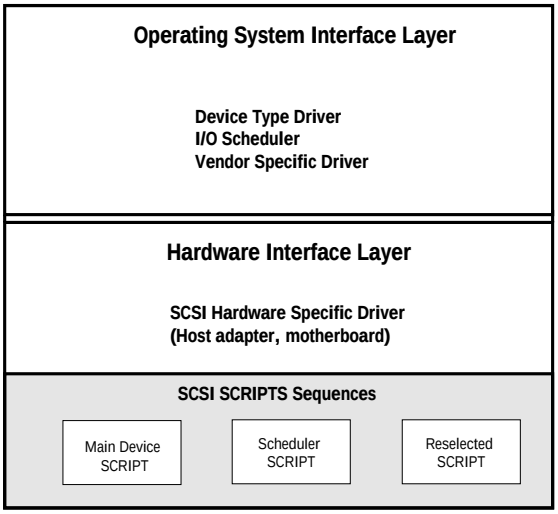


Figure 8-2
SCSI Device Driver Layers

The hardware interface layer interprets the operating system interface's formatted requests and prepares the SYM53C8XX by initializing the DMA, SCSI, and Interrupt registers and by loading the appropriate SCRIPTS into host memory. It then reserves memory for any data buffers that will be used by the SCRIPTS program. The hardware interface layer initializes data buffer addresses, byte counts, and SCSI IDs embedded in the SCRIPTS code. It then starts the execution of the SCRIPTS routine by loading the DSP register (2C-2Fh) with the address of the first SCRIPTS instruction. It waits for an interrupt to signal that the I/O is complete, then passes I/O status information back to the operating system interface.

Command Block

When the operating system interface layer of the SCSI device driver receives an I/O request, it creates a data structure in host memory. This data structure contains the information required by the hardware interface for that specific request. This information generally includes:

- the length of the array
- the SCSI ID for the target device
- the logical unit number (LUN)
- the length of the command block
- the SCSI command containing the beginning block and the number of blocks to be transferred
- a place for the hardware interface to write its completion status. The operating system interface reads the completion status and uses it to update the scheduler information.

Power Up Example

The hardware interface initializes the chip whenever the system is powered up or reset. In the DOS example in Figure 8-3, the system BIOS scans host memory for a ROM, signified by a 55AA code. It reads the third byte of the ROM, which contains a jump address. The following SYM53C8XX initialization information is located at that address:

- diagnostics to be run
- SCRIPTS to be loaded
- data buffer areas to be reserved

After performing these tasks, the hardware interface then scans for the hard disk and loads the operating system from it. The operating

system cannot be loaded from disk until the SCSI driver is active. This power on sequence of activities is illustrated in Figure 8-3.

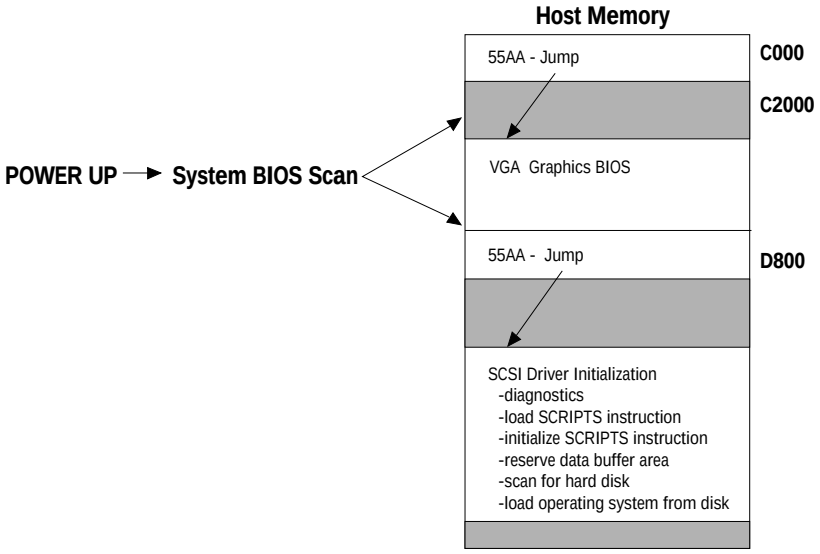


Figure 8-3
Power Up Example

I/O Request Process

Figure 8-4 illustrates a typical SCSI I/O operation. The I/O begins when the user application makes a request to the host operating system to access data on a SCSI device. The request is passed to the SCSI device driver's operating system interface where it is interpreted, scheduled, and formatted for the hardware interface. The operating system interface creates a data structure in host memory, which it passes to the hardware interface layer for execution. The hardware interface uses the information in the command block to determine which SCRIPTS routine to run, as well as where to place the data in memory.

The hardware interface sets up the data areas for the command and data, buffers (initialized table areas and buffers that are needed for the SCRIPTS to execute), and loads the SCRIPTS starting address into the DSP register of the SYM53C8XX chip. The SYM53C8XX executes the subroutine, accessing the drive with the SCSI device ID specified. When the I/O is complete, the hardware interface receives an interrupt and notifies the operating system interface. The operating system interface reads the completion status and uses it to

update the scheduler information. For more information on the scheduler, refer to Chapter 10.

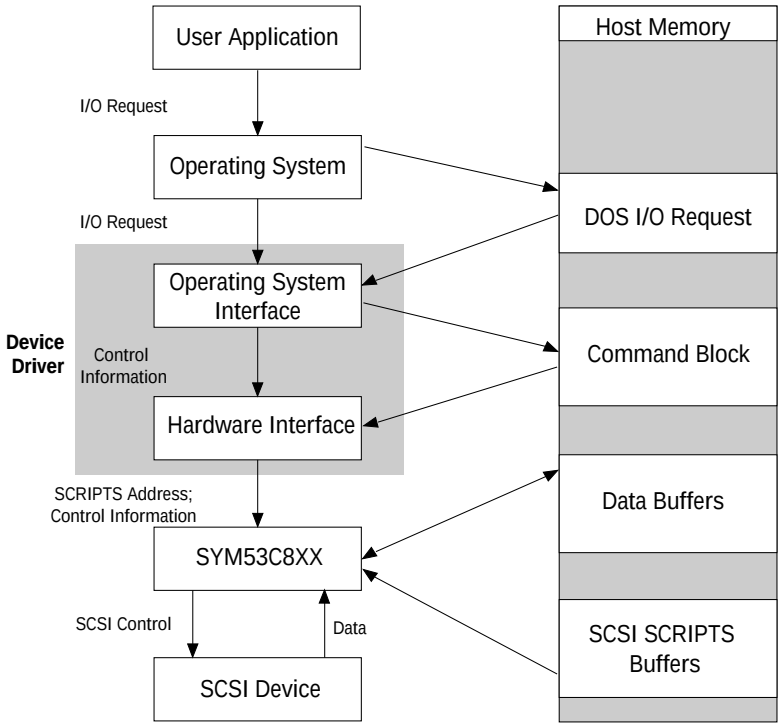


Figure 8-4
I/O Operation

How to Write a Device Driver With SCRIPTS

To develop an executable SCSI SCRIPTS program, first define the SCSI functions required. Identify which functions will be executed in SCRIPTS code and which ones must be contained in other parts of the driver code. Then, design the specific algorithms for the functions that will be executed in the SCSI SCRIPTS portion of the SCSI driver. A SCSI SCRIPTS program contains two areas: the definition area and the SCRIPTS area. The definition area contains variable and absolute values. These values may describe a variable location, variable byte count, or a fixed status byte value. The SCRIPTS area contains the SCSI instructions.

Use the SCRIPTS language to write instructions, then assemble them to create the SCRIPTS output file. The assembler output is a "C" include file that includes relocation information required to load the SCRIPTS object module into main memory, if any relocation is required. It can be directly included in firmware written in the "C" language.

When the SCRIPTS starting address is loaded, the SCRIPTS absolute jump addresses must be resolved. It is necessary to patch in the correct buffer addresses, byte counts, destination ID, and so forth, if table indirect addressing is not used.

Writing a logical I/O driver for the SYM53C8XX is easier than previous generation solutions. Because SCSI sequences are so simple to implement when written in SCSI SCRIPTS, you can rapidly prototype SCSI sequences for proof of concept and build on them to create more complete driver programs.

Table Indirect Addressing

Table indirect addressing simplifies SCRIPTS by separating addresses and device information from control information in Block Move and Select/Reselect instructions. One of the major advantages of table indirect addressing is that SCRIPTS can directly load operating system I/O data from the tables, which increases program efficiency and simplifies program structure. These tables eliminate the need for patching SCRIPTS at the beginning of an I/O. The table can begin on any dword boundary and can cross system segment boundaries. There are three restrictions on the placement of tables in memory:

1. The I/O data structure must lie within 8 MB above or below the base address.
2. An I/O table entry must have all 8 bytes contiguous in system memory.
3. The table must be a contiguous data structure of 8-byte entries.

Prior to the start of an I/O, the DSA register must be loaded with the base address of the table indirect data structure. The address must be on a dword boundary. At the start of a table indirect instruction, the DSA is added to the 24-bit signed offset value from the op code to generate the address of the table entry. Both positive and negative offsets are allowed. With table indirect addressing, it is not necessary to initialize the SCSI ID, byte counts, clock dividers, synchronous parameters, or data buffers within the SCRIPTS instruction. Instead, only the table in memory needs to be updated.

To use table indirect addressing, set up tables in memory similar to the one shown in Figure 8-5. These tables contain device IDs, synchronous period information, byte counts, and data addresses. The data in the table entry is fetched into the appropriate instruction, depending on whether it is a Block Move or a Select/Reselect.

Block Move Instructions

When table indirect mode is selected by using the FROM operator in a SCRIPTS Block Move instruction, the 32-bit start address is treated as a 24-bit signed value. After the instruction is moved into the SYM53C8XX, the 24 bits are added to the DSA register to form a 32-bit physical address. From this new address, the byte count (24 bits of count plus 8 bits of high-order zeros) and the data buffer address (32 bits) are fetched.

There are several programming implications of table indirect addressing. First, a standard SCSI data structure can be designed

with values at predefined offsets. The Block Move instruction does not require the actual 32-bit address or 24-bit count to be in the instruction itself. At the start of an I/O, once the actual data structure is built, no more firmware intervention is required except loading the data table base address into the DSA register. Second, the SCRIPTS instructions may be placed in a PROM because no dynamic alteration is required at the start of an I/O. Finally, only one copy of the main SCSI SCRIPTS program is needed for all I/O operations, with a fast context switch used to change to another I/O. Only the data structure is unique to each I/O, and the SCRIPTS instructions are reusable.

Dword 0	**Byte Lane 3**	**Byte Lane 2**	**Byte Lane 1**	**Byte Lane 0**
	Byte Count			
Dword 1	**Byte Lane 3**	**Byte Lane 2**	**Byte Lane 1**	**Byte Lane 0**
	Address			

Select/Reselect Instructions

During a Select/Reselect, when FROM is used to indicate table indirect addressing, the 24-bit signed value in the DBC register is an offset relative to the value of the DSA register. The table indirect feature allows the Synchronous Clock Conversion, Enable Wide SCSI, Clock Conversion Factor, SCSI Device ID, Synchronous Offset, and Synchronous Period bit values to be fetched from an I/O data structure that is built at the start of an I/O. Thus, an I/O can begin with no requirement to write the values into the chip or into the actual SCRIPTS instruction in memory. In the I/O data structure, the user must have written the following 8-byte value:

Dword 0	**Byte Lane 3**	**Byte Lane 2**	**Byte Lane 1**	**Byte Lane 0**
	Config (SCNTL3)	Device ID (SDID)	Period & Offset (SXFER)	Res
Dword 1	**Byte Lane 3**	**Byte Lane 2**	**Byte Lane 1**	**Byte Lane 0**
	Res	Res	Res	Res

The configuration information in byte lane 3 is mapped into the SCNTL3 register (03h). This includes the Synchronous Clock Conversion Factor, Enable Wide SCSI, Enable Ultra SCSI, and Clock Conversion Factor. The Encoded SCSI destination ID in byte lane 2 is mapped into the SDID register (06h), and the period and

offset information in byte lane 1 is mapped into the SXFER register (05h). The data must begin on a 4-byte boundary and must be located at the 24-bit signed offset from the address contained in the DSA register.

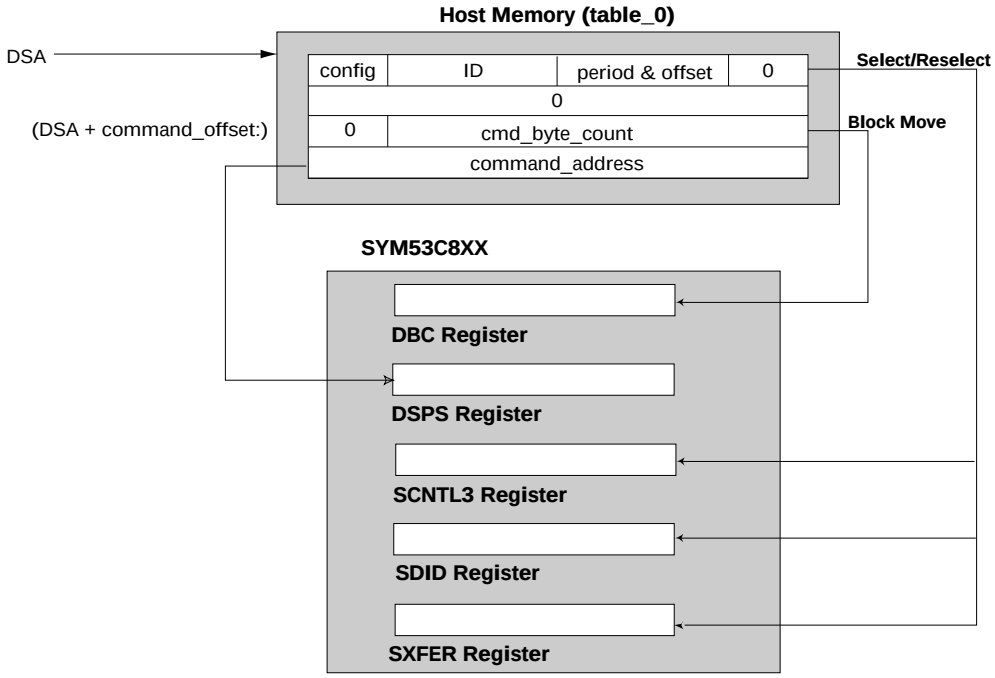


Figure 8-5
Table Indirect Addressing

Defining a Table

The first step in defining a table is to describe it in SCRIPTS code in terms of the order and size of table entries, or buffers. An example is shown in Figure 8-6.

```
; Table definition and use in SCRIPTS

Table dsa_table \

SCSI_ID = ID{0x33, 0x00, 0x00, 0x00}
Data_buf = 512{??} , \
ID_msg =  {0x80} , \
CMD_buf = {0x08, 0x00, 0x00, 0x00, 0x01, 0x00}

ENTRY table_use
```

SCRIPTS Code	Output File		Memory Definition
table_use:	ULONG SCRIPT [ ] = {		
SELECT ATN FROM SCSI_ID, REL (resel)	0x47000000	0x00000020,	Cmd (0x47) / Table Offset (0x00); Alternate Jump Address (0x20)
MOVE FROM ID_msg, WHEN MSG_OUT	0x1E000000	0x00000010,	Cmd (0x1E) / Not Used; Table Offset (0x10)
MOVE FROM CMD_buf, WHEN CMD	0x1A000000	0x00000018,	Cmd (0x1A) / Not Used; Table Offset (0x18)
MOVE FROM Data_buf, WHEN DATA_IN	0x19000000	0x00000008	Cmd (0x19) / Not Used; Table Offset (0x08)
INT 0x0A	0x98080000	0x0000000A,	
resel:			
int 0x0B	0x98080000	0x0900000B	
	};		
	#define ENT_table_use	0x00000000	
	ULONG INSTRUCTIONS	= 0x00000006;	
	ULONG PATCHES	= 0x00000000;	

Figure 8-6
Table Definition

Relative Addressing

In relative addressing mode, the 24-bit signed value in the DSPS register is used as a relative displacement from the current DMA SCRIPTS Pointer (DSP) register. Using this mode, the 32-bit physical address is formed at execution time, and there is no need to patch a SCRIPTS instruction at run time. Relative addressing may be used for jumps or calls, and requires no initialization of jump and call addresses. This feature may also be used with the alternate address field of Select, Reselect, Wait Select, and Wait Reselect instructions.

> Note: use the REL qualifier keyword in SCRIPTS instructions to specify relative addressing. RELATIVE is a declarative keyword, used by the SCRIPTS assembler, to establish relative buffers. These relative buffers are not used in connection with relative addressing.

Chapter 9

SCRIPTS Programming Topics

Overview

This chapter presents general information for some of the programming tasks that are often performed by SCRIPTS programs. For the most up-to-date example code for many of these operations, please contact the Symbios Logic electronic bulletin board.

Scatter/Gather Operations

Scatter/gather is used when data that is scattered throughout memory must be transferred across the SCSI bus together. Memory management units keep track of physical locations of user data that cannot be stored contiguously. During an I/O request for a SCSI device to fetch data, the memory management unit builds a gather table that provides the addresses of all of the desired data. There may be several entries, or pages, of data associated with a single transfer. Without scatter gather each entry is treated as an individual transfer, requiring a processor interrupt and DMA setup.

With SCSI SCRIPTS, it is possible to set up multiple data buffer areas and then fill them rapidly without interrupting the host processor. This allows faster and more efficient scatter/gather operations. Block move data may come from any memory address, so scatter/gather operations for user data are transparent to the chip and the host processor. With the technique illustrated below, a number of data buffers (pages, or gather table entries) are defined in advance and each is associated with a Block Move instruction. Any number of Block Moves can be hard-coded into the buffers. If the scatter/gather list requested has more entries than have been defined for the buffer, then an interrupt after the last entry in the series can inform the firmware it needs to set up the remaining scatter gather entries after the first group is complete.

```
RW_Offset_patch_do:
;Relative offset will be changed so that we jump into the
;proper place in the scatter gather list

JUMP REL(Data_Out_xfer); Data_Out_xfer:

CHMOV FROM data_buf1, WHEN DATA_OUT CHMOV FROM data_buf2, WHEN

; 16 moves to support Scatter Gather
DATA_OUT
CHMOV FROM data_buf3, WHEN DATA_OUT
CHMOV FROM data_buf4, WHEN DATA_OUT
CHMOV FROM data_buf5, WHEN DATA_OUT
CHMOV FROM data_buf6, WHEN DATA_OUT
CHMOV FROM data_buf7, WHEN DATA_OUT
CHMOV FROM data_buf8, WHEN DATA_OUT
CHMOV FROM data_buf9, WHEN DATA_OUT
CHMOV FROM data_buf10, WHEN DATA_OUT
CHMOV FROM data_buf11, WHEN DATA_OUT
CHMOV FROM data_buf12, WHEN DATA_OUT
CHMOV FROM data_buf13, WHEN DATA_OUT
CHMOV FROM data_buf14, WHEN DATA_OUT
CHMOV FROM data_buf15, WHEN DATA_OUT
CHMOV FROM data_buf16, WHEN DATA_OUT


; Check to see if we need more SG list entries
MOVE DWT & RW_NEED_MORE_SG_ENTRIES to SFBR
INT RW_Need_More_SG, if not 0

; If we are here then all the data was transferred
; so we set a flag to indicate that
MOVE SBR | RW_ALL_DATA_TRANSFERRED to DWT

JUMP REL(RW_Handle_Phase)

; *** Script move data ENTRY
RW_Offset_patch_di:

;Relative offset will be changed so that we jump into the
;proper place in the scatter gather list
JUMP REL(Data_In_xfer); Data_In_xfer:
CHMOV FROM rw_data_buf1, WHEN DATA_IN CHMOV FROM rw_data_buf2,
WHEN DATA_IN

; 16 moves to support Scatter Gather
CHMOV FROM rw_data_buf3, WHEN DATA_IN
CHMOV FROM rw_data_buf4, WHEN DATA_IN
CHMOV FROM rw_data_buf5, WHEN DATA_IN
CHMOV FROM rw_data_buf6, WHEN DATA_IN
CHMOV FROM rw_data_buf7, WHEN DATA_IN
CHMOV FROM rw_data_buf8, WHEN DATA_IN
CHMOV FROM rw_data_buf9, WHEN DATA_IN
CHMOV FROM rw_data_buf10, WHEN DATA_IN
CHMOV FROM rw_data_buf11, WHEN DATA_IN
CHMOV FROM rw_data_buf12, WHEN DATA_IN
CHMOV FROM rw_data_buf13, WHEN DATA_IN
CHMOV FROM rw_data_buf14, WHEN DATA_IN
CHMOV FROM rw_data_buf15, WHEN DATA_IN
CHMOV FROM rw_data_buf16, WHEN DATA_IN
```

```
; Check to see if we need more SG list entries
MOVE SBR & RW_NEED_MORE_SG_ENTRIES to SFBR
INT RW_Need_More_SG, if not 0

; If we are here then all the data was transferred
; so we set a flag to indicate that
MOVE SBR | RW_ALL_DATA_TRANSFERRED to DWT
```

The following example shows an alternative method for doing scatter/gather operations in SCRIPTS. This mechanism uses a looping strategy to execute each scatter/gather entry. On each loop the DSA value is incremented by 8, effectively moving to the next scatter/gather entry in the scatter/gather list. Generally, when this strategy is used, the scatter/gather list is located at the end of the table indirect entries or is located separately from the other table indirect entries that handle (re)select, message, command and status phases. The DSA value is then restored after the scatter gather operations are complete or the target changes phase. This method of doing scatter/gather operations requires that table indirect addressing be used.

```
Move_Data:
MOVE MEMORY 4, DSA_addr, ScratchB_addr  ; save DSA
                                                address
JUMP REL(Data_In_Loop), WHEN DATA_IN

Data_Out_Loop:
MOVE FROM io_data_buf, WHEN DATA_OUT
MOVE DSA0 + 8 to DSA0 ; Update DSA for scatter gather
JUMP REL(Skip_Carry_Adds_DO), IF NOT CARRY; operations
MOVE DSA1 + 0 to DSA1 WITH CARRY
MOVE DSA2 + 0 to DSA2 WITH CARRY
MOVE DSA3 + 0 to DSA3 WITH CARRY

Skip_Carry_Adds_DO:
JUMP REL(Data_Out_Loop), WHEN DATA_OUT
MOVE MEMORY 4, ScratchB_addr, DSA_addr ; restore DSA
                                       ;  address
JUMP REL(Get_Status), WHEN STATUS
JUMP REL(Handle_Message), WHEN MSG_IN
INT Unexpected_Phase
```

Loopback Mode

The SYM53C8XX provides advanced diagnostic and testing capabilities. Loopback mode allows the SYM53C8XX to control all signals, regardless of whether it is in initiator or target role. This mode provides the ability to check the functionality of the part, insuring proper SCRIPTS instruction fetches, checking bad parity procedures, and insuring all data paths work properly. The SYM53C8XX loopback mode allows testing of both initiator and target operations. In this mode, the SYM53C8XX allows control of all SCSI signals, and actually talks to itself. The SYM53C8XX usually executes initiator instructions through the SCRIPTS program and the host CPU implements the target role by asserting and polling the appropriate SCSI signals in the SOCL, SODL, SBCL, and SBDL registers. The initiator role is accomplished using SCSI SCRIPTS and the target role is implemented using "C" code to access the 53C8XX registers. The roles could be switched to test target role applications of the SYM53C8XX.

To run loopback mode correctly, the following registers must be initialized to the proper values.

- STEST2. Bits 3 and 4 should be set to turn on loopback mode and hi-z the SCSI pins, so that signals are not asserted onto the SCSI bus. STEST 2 bits 7-6 and 0 do not affect loopback operation, but should remain clear. The values of bits 5 and 2-1 will not affect the running of loopback mode
- DCNTL. Bit 4 should be set to turn on single step mode. This allows the target program to monitor when an initiator SCRIPTS instruction has completed. Bits 3-2 should be clear, and the remaining bit values will not affect the running of loopback mode.
- DIEN. Bit 3 should be set to enable single step interrupts. This bit works in conjunction with the single step mode bit to allow for monitoring of SCRIPTS instruction completion. The remaining bit values in this register will not affect the running of loopback mode.

Loopback Example - Selection

The following example shows how to perform selection in the SCSI Loopback mode. It provides all the general code required to implement any of the various SCSI sequences in loopback mode. This example assumes that the SYM53C8XX was initialized as described above. The initiator instructions are implemented using the SYM53C8XX and SCRIPTS. The target instructions are implemented using the CPU and a "C" program.

When a SCRIPTS routine is executing, a waiting period is required to fetch the SCRIPTS instructions; this fetch time must be taken into account when writing loopback code. To insure proper operation, a delay should be inserted directly after SCRIPTS instructions have started executing. After the DSP register (2C-2Fh) is initialized with a SCRIPTS instruction address, the chip registers cannot be accessed until the instruction has been fetched and begins executing. This delay time must include:

- host arbitration
- SCRIPTS instruction fetch
- SCRIPTS instruction execution or internal bus moves

These delay times are system-dependent because of host arbitration times, host bus width, and chip clock speed.

```
/*Load DSP with address of Select w/ATN instruction*/

/* SELECT ATN tar_id, REL(This_wont_occur) */

write_longreg (DSP,SCRIPTS_sel_inst);

/* Delay to allow instruction to be fetched by SIOP */

delay(1); /* 1 ms delay, varies with system*/

/* TARGET, wait for SEL to go high and BSY to go low */

while ((siop_reg[SBCL] & 0x30) != 0x10;

/*TARGET, check ID, but really don't care what it is */

printf("Initiator: Selecting target ID
%x\n",siop_reg[SBDL]);

/*TARGET, assert BSY*/

siop_reg[SOCL] = 0x20;

/*TARGET, wait for SEL to drop */

while ((siop_reg[SBCL] & 0x10) !=0);
```

In this section of code, the Initiator Select SCRIPTS routine is started by writing the address of the Select instruction to the DSP. A delay is inserted to insure that the SIOP has time to fetch the instruction. Polling the SBCL register determines when SEL/ is active and selecting itself.

The variable `siop_reg` should be defined as a volatile pointer to the registers of the SYM53C8XX. This insures that the registers will not be shadowed internally by the CPU. Polling the SBDL register determines which SCSI ID bits are being driven. This is not a vital step in the loopback selection process, since the SYM53C8XX is selecting itself. However, SBDL should be checked to make sure the

correct bits are driven on the SCSI data bus during normal selection. The BSY/ bit is set in the SOCL register. This is a target operation performed by the CPU. Polling the SEL/ bit of the SBCL register determines when SEL/ is inactive. This indicates the "initiator" is properly responding to BSY/ being asserted by the "target."

```
/*TARGET, check for ATN*/
if (siop_reg[SBCL] & 0x08) {
   /*TARGET, assert BSY, and MSG OUT*/
siop_reg[SOCL] = 0x26;
/*Self-Selection with ATN is now complete.*/
/* Wait for single step interrupt*/
while ((siop_reg[ISTAT] & 0x03) ==0);
/*Clear all interrupts*/
junk = siop_reg[SIST0];
junk = siop_reg[SIST1]
junk = siop_reg[DSTAT];
/*Start Script Block Move instruction*/
/*to send Identify Message to "Target" */
/*MOVE 1, identify_buf, WHEN MESSAGE OUT*/
siop_reg[DCNTL] |= 0x04;
/*Wait for SCRIPTS routine to finish using host bus*/
delay(1);
```

The program checks the SBCL register to determine if the selection is with or without SATN/. This will effect the next phase that is asserted by the "target." The desired phase is asserted by setting the MSG/, C_D/, and I/O bits in the SOCL register while maintaining BSY/. This would be Message Out if SATN/ was sampled asserted or Command if SATN/ was sampled deasserted in the SBCL register. At this point, selection with ATN/ is now complete. The SIP and DIP bits in the ISTAT register are polled for a single step interrupt and any others that may have occurred. These interrupts are cleared by reading the SIST0, SIST1 and DSTAT registers. Note that the single step interrupt will be cleared by reading the DSTAT register. Other interrupts may occur, depending on the particular settings in the SIEN and DIEN registers. It is a safe procedure to make sure all interrupts are cleared, as any pending interrupts would inhibit the execution of further SCRIPTS instructions. This example uses a

polled interrupt procedure. If hardware interrupts are used, then this would be handled in an interrupt service routine.

The Start DMA operation bit of the DCNTL register is set so that the Block Move SCRIPTS instruction will begin execution. This Block Move instruction is used to transfer the identify message associated with Selection with ATN/ to the target. A delay is inserted to ensure that the SYM53C8XX has time to fetch the instruction.

The next section of code shows how to transfer bytes using loopback mode. Although this is shown in line with the rest of the sample code, it can be moved out into a separate function and used for any generic data transfer between the initiator and the target, whenever the SYM53C8XX is executing a Block Move instruction. The assertion of the SREQ/ signal is the first thing that is done in this code.The SREQ/ signal is asserted by keeping the phase bits the same and setting the SREQ/ bit in the SOCL register. This is an acceptable action for a data transfer from the “Initiator” to the “Target” (DATA OUT, MESSAGE OUT, or COMMAND phase). For a transfer from the “target” to the “initiator” (DATA IN, MESSAGE IN, or STATUS phase) the data should be placed into the SODL register before SREQ/ is asserted. Because the SYM53C8XX clocks asynchronous data in on the rising edge of SACK/, data will be corrupted if this procedure is not followed. If SREQ/ is asserted, the SYM53C8XX will immediately assert ACK/ and clock in whatever data is in the SOCL register, if the data has not been placed into the SOCL register, then incorrect data will be clocked in.

After asserting the SREQ/ signal, the SBCL register is polled for assertion of the SACK/ signal by the “initiator.” Data is then read by the “target” from the SBDL register. SREQ/ is deasserted by the “target” using the SOCL register, and the “target” polls the SBCL register for deassertion of SACK/ by the “initiator.” The byte received by the “target” is verified with the byte sent by the “initiator.”

```
/*TARGET, Get Message Byte */

/*TARGET, assert REQ, maintain all other SCSI signals*/

siop_reg[SOCL] |=0x80;

/*TARGET, wait for ACK*/

while ((siop_reg[SBCL] & 0x40) !=0)

msg_out_buf = siop_reg[SBDL]; /*read the data bus*/

siop_reg[SOCL] &=0x7f; /*deassert REQ*/

while ((siop_reg[SBCL] & 0x40) !=0) /* wait for ACK*/

/* verify message byte */

if (msg_out_buf !=identify_buf) {

loop_err = 1;

}
```

The following section of code shows the final step of the selection procedure in Loopback mode. This selection procedure could be placed into a function, as could procedures that implement command, status, message in, and data transfer phases. Upon doing this, full SCSI sequences could be implemented in loopback mode by various function calls in the proper order.

If the selection was without ATN/, then nothing else needs to be done other than assert the next phase and wait for a single step interrupt. The MSG/, C_D/, and I_O/ signals are set to command phase via the SOCL register. BSY/ is also kept asserted. The SIP and DIP bits in the ISTAT register are polled for a single step interrupt along with any other interrupts that may have occurred. These interrupts are cleared by reading the SIST1, SIST0, and DSTAT registers. Note that the single step interrupt will be cleared by reading the DSTAT register, but depending on the particular settings in the SIEN and DIEN registers, other interrupts may occur. It is a safe procedure to make sure all interrupts are cleared, as any pending interrupts would inhibit the execution of remaining SCRIPTS instructions. This example uses a polled interrupt procedure. If hardware interrupts are used then this would be handled in an interrupt service routine. The chip is now in a state to transfer command bytes. This can be accomplished by using the generic byte transfer code given earlier in this example.

```
else{ /*select without ATN*/

   printf(“Initiator: Selecting without ATN.../n);

}

/*assert BSY and Command phase*/

siop_reg[SOCL} = 0x22;

/*wait for single step int.*/

while ((siop_reg[ISTAT] & 0x03) == 0);

/* clear all interrupts */

junk = siop_reg[SIST0];

junk = siop_reg[SIST1];

junk = siop_reg[DSTAT];

/*SELECTION COMPLETE*/
```

Byte Recovery on Target Disconnect

There are three cases where a disconnect may occur. The first is during a Data Read, when a SCSI device may disconnect while it is seeking the data that was requested. This is very common, such as when a SCSI disk drive must perform a seek operation. Seeks often take many milliseconds to accomplish, and it is inefficient for the disk drive to stay active on the bus when it has nothing to transfer. A second case is after a SCSI device completes a write operation, and disconnects to empty its buffers before returning its status and command complete messages.

The third type of disconnect may occur at any time. It occurs when data is being written to a SCSI device and its internal buffers become full. The device will disconnect before the data transfer is complete to empty its buffers and avoid an overflow condition. When it does, the SCSI bus is in a different phase from that expected by the initiator, creating a phase mismatch. When this happens, the SYM53C8XX must interrupt and the CPU must perform byte recovery. When a disconnect occurs, data may be in transition; it is important to determine how much and where it is. In addition, it is crucial to know where in the SCRIPTS program the transfer was interrupted so that it can be resumed at a later time. To save the state of the chip at the time of the disconnect, get the address of the current SCRIPTS instruction and calculate the number of bytes of active data remaining to be transferred. After saving the state of the chip, update the SCRIPTS program and flush or clear the FIFO.

Saving the State of the SYM53C8XX

The first step in saving the state of the SYM53C8XX is to write the address of the current SCRIPTS instruction from the DSP register to a special table that is indexed by SCSI ID. The instruction at that address can be restored later to resume processing. The DSP is incremented as the current instruction is fetched, so it will always point to the next instruction. Therefore, the DSP will need to be decremented by 8 or 12, depending on whether the instruction was a regular SCRIPTS instruction or a Memory to Memory Move. This can be determined by reading the DCMD register. Typically, the instruction will be a Block Move. If table indirect addressing is used, it may only be necessary to update the table and not the SCRIPTS code.

Target disconnect may create a need to recover bytes in the chip's data paths. The location of the data is dependent on whether data is being moved into or out of the chip, and whether SCSI data is being transferred asynchronously or synchronously. The following steps will determine if any bytes remain in the data path when the chip halts an operation. Please consult the appropriate product data manual for

exact information on the default and extended (when supported) DMA FIFO sizes in each SYM53C8XX PCI-SCSI I/O Processor.

Asynchronous SCSI Send

1. If the DMA FIFO size is set to the default size, Look at the DFIFO and DBC registers and calculate if there are bytes left in the DMA FIFO. To make this calculation, subtract the seven least significant bits of the DBC register from the 7-bit value of the DFIFO register. AND the result with 7Fh for a byte count between zero and the FIFO size.

   If the DMA FIFO size is set to the extended size, subtract the 10 least significant bits of the DBC register from the 10-bit value of the DMA FIFO Byte Offset Counter, which consists of bits 1-0 in the CTEST5 register and bits 7-0 of the DMA FIFO register. AND the result with 3FFh for a byte count between 0 and the extended FIFO size.

2. Read bit 5 in the SSTAT0 and SSTAT2 registers to determine if any bytes are left in the SODL register. If bit 5 is set in the SSTAT0 or SSTAT2 then the least significant byte or the most significant byte in the SODL register is full, respectively. Checking this bit also reveals bytes left in the SODL register from a Chained Move operation with an odd byte count.

Synchronous SCSI Send

1. If the DMA FIFO size is set to the default size, look at the DFIFO and DBC registers and calculate if there are bytes left in the DMA FIFO. To make this calculation, subtract the seven least significant bits of the DBC register from the 7-bit value of the DFIFO register. AND the result with 7Fh for a byte count between zero and the FIFO size.

   If the DMA FIFO size is set to the extended size, subtract the 10 least significant bits of the DBC register from the 10-bit value of the DMA FIFO Byte Offset Counter, which consists of bits 1-0 in the CTEST5 register and bits 7-0 of the DMA FIFO register. AND the result with 3FFh for a byte count between 0 and the FIFO size.

2. Read bit 5 in the SSTAT0 and SSTAT2 registers to determine if any bytes are left in the SODL register. If bit 5 is set in the SSTAT0 or SSTAT2 then the least significant byte or the most significant byte in the SODL register is full, respectively. Checking this bit also reveals bytes left in the SODL register from a Chained Move operation with an odd byte count.

3 Read bit 6 in the SSTAT0 and SSTAT2 registers to determine if any bytes are left in the SODR register. If bit 6 is set in the SSTAT0 or SSTAT2 then the least significant byte or the most significant byte in the SODR register is full, respectively.

Asynchronous SCSI Receive

1 If the DMA FIFO size is set to the default size, Look at the DFIFO and DBC registers and calculate if there are bytes left in the DMA FIFO. To make this calculation, subtract the seven least significant bits of the DBC register from the 7-bit value of the DFIFO register. AND the result with 7Fh for a byte count between 0 and the FIFO size.

If the DMA FIFO size is set to the extended size, subtract the 10 least significant bits of the DBC register from the 10-bit value of the DMA FIFO Byte Offset Counter, which consists of bits 1-0 in the CTEST5 register and bits 7-0 of the DMA FIFO register. AND the result with 3FFh for a byte count between 0 and the FIFO size.

2 Read bit 7 in the SSTAT0 and SSTAT2 register to determine if any bytes are left in the SIDL register. If bit 7 is set in the SSTAT0 or SSTAT2 then the least significant byte or the most significant byte is full, respectively.

3 If any wide transfers have been performed using the Chained Move instruction, read the Wide SCSI Receive bit (SCNTL2, bit 0) to determine whether a byte is left in the SWIDE register.

Synchronous SCSI Receive

1 If the DMA FIFO size is set to the default size, subtract the seven least significant bits of the DBC register from the 7-bit value of the DFIFO register. AND the result with 7Fh for a byte count between 0 and the FIFO size.

If the DMA FIFO size is set to the extended size, subtract the 10 least significant bits of the DBC register from the 10-bit value of the DMA FIFO Byte Offset Counter, which consists of bits 1-0 in the CTEST5 register and bits 7-0 of the DMA FIFO register. AND the result with 3FFh for a byte count between 0 and the FIFO size.

2 Read the SSTAT1 register (and bit 4 of the SSTAT2 register for extended FIFO size), the binary representation of the number of valid bytes in the SCSI FIFO, to determine if any bytes are left in the SCSI FIFO.

If any wide transfers have been performed using the Chained Move instruction, read the Wide SCSI Receive bit (SCNTL2, bit 0) to determine whether a byte is left in the SWIDE register.

Updating the SCRIPTS Program

Once the number of bytes in transition has been calculated, the SCRIPTS instruction must be updated so that the correct number of bytes will be transferred when the target reselects. This is done by updating the byte count and address in the SCRIPTS program at whatever the current instruction was at the time of disconnect. The SCRIPT is stored in the host's main memory, so it may be modified at any time. This manipulation must be performed on the binary version of the instruction in host memory unless table indirect addressing is used. If table indirect mode is used, the byte count and address would be modified in the data structure instead of the binary version of the instruction.

Cleaning Up

Bytes that are already in transition must be processed. Depending on the direction of transfer and how the user writes the code, any data left in the chip must be flushed to memory (SCSI Receive only) or cleared and discarded. The Flush DMA FIFO bit in the CTEST3 register flushes the DMA FIFO data to memory. The Clear DMA FIFO bit in CTEST3 discards the data in the DMA FIFO. The Clear SCSI FIFO bit in STEST3 clears the data out of the Synchronous SCSI Receive FIFO and clears data in any other intermediate registers.

In a normal disconnect situation, when a Phase Mismatch interrupt occurs during a SCSI receive, no data should be left in the chip except in the SWIDE register.

Note: The Wide SCSI Send and Wide SCSI Receive bits are cleared by any non-wide send or receive action, such as moving message bytes. Examine these bit values first during byte recovery.

Example Byte Recovery Code

Byte recovery must be done when the SYM53C8XX receives a phase mismatch interrupt either during Data In or Data Out phase. Below are two example functions which handle these situations.

These examples use the following SCRIPTS sequence to move data:

```
Move_Data:

JUMP REL(RW_Offset_patch_di), WHEN DATA_IN

;During a write command, some devices disconnect after all the
;data has been sent and reselect with Status and msg_in. The
;following instructions prevents phase mismatch when this
;happens.
JUMP REL(RW_Handle_Phase) WHEN NOT DATA_OUT
```

```
; *** Script move data out ENTRY
RW_Offset_patch_do:

;Relative offset will be changed so that we jump
;into the proper place in the scatter gather list
JUMP REL(Data_Out_xfer); Data_Out_xfer:

CHMOV FROM data_buf1, WHEN DATA_OUT CHMOV FROM data_buf2, WHEN
DATA_OUT

; 16 moves to support Scatter Gather
CHMOV FROM data_buf3, WHEN DATA_OUT
CHMOV FROM data_buf4, WHEN DATA_OUT
CHMOV FROM data_buf5, WHEN DATA_OUT
CHMOV FROM data_buf6, WHEN DATA_OUT
CHMOV FROM data_buf7, WHEN DATA_OUT
CHMOV FROM data_buf8, WHEN DATA_OUT
CHMOV FROM data_buf9, WHEN DATA_OUT
CHMOV FROM data_buf10, WHEN DATA_OUT
CHMOV FROM data_buf11, WHEN DATA_OUT
CHMOV FROM data_buf12, WHEN DATA_OUT
CHMOV FROM data_buf13, WHEN DATA_OUT
CHMOV FROM data_buf14, WHEN DATA_OUT
CHMOV FROM data_buf15, WHEN DATA_OUT
CHMOV FROM data_buf16, WHEN DATA_OUT


; Check to see if we need more SG list entries
;In older SYM53C8XX chips, SBR = DWT
MOVE SBR & RW_NEED_MORE_SG_ENTRIES to SFBR
INT RW_Need_More_SG, if not 0

; If we are here then all the data was transferred
; so we set a flag to indicate that
MOVE SBR | RW_ALL_DATA_TRANSFERRED to DWT

JUMP REL(RW_Handle_Phase)

; *** Script move data ENTRY
RW_Offset_patch_di:

;Relative offset will be changed so that we jump into the
;proper place in the scatter gather list
JUMP REL(Data_In_xfer); Data_In_xfer:
CHMOV FROM rw_data_buf1, WHEN DATA_IN CHMOV FROM rw_data_buf2,
WHEN DATA_IN

; 16 moves to support Scatter Gather
CHMOV FROM rw_data_buf3, WHEN DATA_IN
CHMOV FROM rw_data_buf4, WHEN DATA_IN
CHMOV FROM rw_data_buf5, WHEN DATA_IN
CHMOV FROM rw_data_buf6, WHEN DATA_IN
CHMOV FROM rw_data_buf7, WHEN DATA_IN
CHMOV FROM rw_data_buf8, WHEN DATA_IN
CHMOV FROM rw_data_buf9, WHEN DATA_IN
CHMOV FROM rw_data_buf10, WHEN DATA_IN
CHMOV FROM rw_data_buf11, WHEN DATA_IN
CHMOV FROM rw_data_buf12, WHEN DATA_IN
CHMOV FROM rw_data_buf13, WHEN DATA_IN
CHMOV FROM rw_data_buf14, WHEN DATA_IN
CHMOV FROM rw_data_buf15, WHEN DATA_IN
```

```
CHMOV FROM rw_data_buf16, WHEN DATA_IN


; Check to see if we need more SG list entries
MOVE SBR & RW_NEED_MORE_SG_ENTRIES to SFBR
INT RW_Need_More_SG, if not 0

; If we are here then all the data was transferred
; so we set a flag to indicate that
MOVE SBR | RW_ALL_DATA_TRANSFERRED to DWT

JUMP REL(RW_Handle_Phase)

; *** Script move SWIDE byte ENTRY
RW_Move_swide_byte:
CHMOV 1, RW_Last_di_byte_buf, WHEN DATA_IN
INT RW_SWIDE_byte_moved
```

Example Function for handling DATA IN Phase Mismatch interrupts

```
/*************************************************************

 Function: HandleDataInPM

   Purpose : To handle clean up after a Phase Mismatch (PM)
             during Data In phase

   Input: The IO Base address of the 8XX chip
      A pointer to a variable which will indicate the
      Scatter Gather entry that was executing when the
      PM occurred, this is needed by the upper function
      if there was a byte in the SWIDE register.

   Output: Current_SG_Entry is filled in with the SG
             entry that was being serviced.
   Assumptions:  That a phase mismatch has actually
             occurred during data in.
   Restrictions: None
   Other functions called: IORead32 to read chip info
                           iowrite32 to start the script
   Global Variables Used:FirstDIMove_paddr is the
                    physical address of the first Data In
                    block move in the scatter/gather
                    list. This is needed to get the
                    location of the scatter/gather entry
                    that was being serviced when the
                    phase mismatch occurred.
                 dsa_table is the table indirect table
                    that is being used for this IO
                 script is the actual script that was
                    being executed when the phase
                    mismatch occurred.
              DATA_BUF1 is the offset into the Table
                    Indirect entries for the first Data
                    In table entry.
```

```
**************************************************/

static void HandleDataInPM(ULONG PCIDeviceIOBase, INT\
*Current_SG_Entry)

{

   ULONG Current_DSP; /* Holds current DSP value */

   /* where am I in the SG list? */
   Current_DSP = IORead32(PCIDeviceIOBase+DSP) - 8;
   *Current_SG_Entry=(VINT)(Current_DSP-\FirstDIMove_paddr)/
8;

   /* On Data In phase mismatch interrupts the part is
automatically flushed so there is no need to check for residual
data in the part, except for data in the SWIDE byte*/

   /* now update the address and count */
   dsa_table[DATA_BUF1 + *Current_SG_Entry].address +=
      dsa_table[DATA_BUF1 + *Current_SG_Entry].count -
      (IORead32(PCIDeviceIOBase+DBC) & 0x00FFFFFl);
   dsa_table[DATA_BUF1 + *Current_SG_Entry].count =
      IORead32(PCIDeviceIOBase+DBC) & 0x00FFFFFl;

   /* update the jump offset into the SG list */
   script[(INT) (Ent_RW_Offset_patch_di/4) + 1] =
      (ULONG) *Current_SG_Entry * 8;

   /* move the byte in SWIDE if necessary */
   if (IORead8(PCIDeviceIOBase+SCNTL2) & 0x01)
   {

      /* patch move to get byte out of chip */
      script[(INT) E_RW_Last_di_byte_buf_Used[0]] =
         buffer_table[DATA_BUF1 +
         *Current_SG_Entry].address;

      /* start script to move byte */
      iowrite32(PCIDeviceIOBase+DSP,
         getPhysAddr(rw_script) +
         Ent_RW_Move_swide_byte);

   }

   else    /* nothing in swide so start the disconnect
             /*script */
      iowrite32(PCIDeviceIOBase+DSP,
         getPhysAddr(rw_script) + Ent_RW_Handle_Phase);
}
```

Example Function for handling DATA OUT Phase Mismatch interrupts:

```
/************************************************************

 Function: HandleDataOutPM

   Purpose: To handle clean up after a Phase Mismatch (PM)during
Data Out phase
   Input: A pointer the pcidev_record.
   Output: None
   Assumptions: That a phase mismatch has actually
            occurred during data out.
   Restrictions: None
   Other functions called:IORead32/8 to read chip info
            RMWon to set bits in chip registers
            iowrite32 to start the script
   Global Variables Used:FirstDOMove_paddr is the
                  physical address of the first Data
                  Out block move in the scatter/gather
                  list. This is needed to get the
                  location of the scatter/gather entry
                  that was being serviced when the
                  phase mismatch occurred.
            dsa_table is the table indirect table that
                  is being used for this IO
            script is the actual script that was being
                  executed when the phase mismatch
                  occurred.
            DATA_BUF1 is the offset into the Table
                  Indirect entries for the first Data
                  In table entry.
*****************************************************/

static void HandleDataOutPM(pcidev_record *PCIDevice)

{

   ULONG Current_DSP;/* holds current dsp value */
   INT Current_SG_Entry;/* Used to calc. Current SG
                           entry */
   UINT DFIFO_val;       /* Holds chip DFIFO value */
   UINT Bytes_remaining;/* Used to accout for other
                           bytes in chip */

   /* where am I in the SG list? */
   Current_DSP = IORead32(PCIDeviceIOBase+DSP) - 8;
   Current_SG_Entry = (INT) (Current_DSP -
FirstDOMove_paddr) / 8;

   /* now update the address and count */
   buffer_table[DATA_BUF1 + Current_SG_Entry].address +=
     buffer_table[DATA_BUF1 + Current_SG_Entry].count -
       (IORead32(PCIDeviceIOBase+DBC) & 0x00FFFFFFl);
   buffer_table[DATA_BUF1 + Current_SG_Entry].count =
      IORead32(PCIDeviceIOBase+DBC) & 0x00FFFFFFl;

   /* Update count and address to reflect any data left in the
chip */
```

```
   /* First check for data in the DMA FIFO */
   /* The variable DFIFO_val is a combination of bits
   /*1-0 of CTEST5 and bits 7-0 of the DFIFO register
   /*this will take care of both the extended FIFO
devices
   /*and all others */
   DFIFO_val = ((IORead8(PCIDeviceIOBase+CTEST5) &
0x03) << 8) |
               (IORead8(PCIDeviceIOBase+DFIFO));

   if (IORead8(PCIDeviceIOBase+CTEST5) & 0x20)/* big
                                              fifo */
      Bytes_remaining = (DFIFO_val - (UINT)
         IORead32(PCIDevice->base_addr2+DBC) & 0x3FF) &
         0x3FF;

   else/* default FIFO size*/
      Bytes_remaining = (DFIFO_val - (UINT)
         IORead32(PCIDevice->base_addr2+DBC) & 0x7F) &
         0x7F;

   /* now check the other regs that may contain data*/

   /* SODL LSB Full?*/
   if (IORead8(PCIDevice->base_addr2+SSTAT0) & 0x20)
      Bytes_remaining++;

   /* SODL MSB Full?*/
   if (IORead8(PCIDevice->base_addr2+SSTAT2) & 0x20
      ) Bytes_remaining++;

   /* SODR LSB Full?*/
   if (IORead8(PCIDevice->base_addr2+SSTAT0) & 0x40)
      Bytes_remaining++;

   /* SODR MSB Full?*/
   if (IORead8(PCIDevice->base_addr2+SSTAT2) & 0x40)
      Bytes_remaining++;

   /* Now update the TI entry */
   rw_buffer_table[RW_DATA_BUF1 +
   Current_SG_Entry].address -= Bytes_remaining;
   rw_buffer_table[RW_DATA_BUF1 +
   Current_SG_Entry].count += Bytes_remaining;

   /* update the jump offset into the SG list */
   rw_script[(INT) (Ent_RW_Offset_patch_do/4) + 1] =
   (ULONG) Current_SG_Entry * 8;

   /*clear the dma fifo to get any left over data out */
   RMWon(PCIDevice->base_addr2+CTEST3, 0x04);

   /* start the disconnect script */
   iowrite32(PCIDeviceIOBase, getPhysAddr(rw_script) +
   Ent_RW_Handle_Phase);

}
```

Synchronous Negotiation and Transfer

The SYM53C8XX must negotiate a set of synchronous parameters for each synchronous device on the SCSI bus. The parameters for each SCSI device are saved in memory, and reloaded into the registers before communications resume between the set of devices. A sample synchronous negotiation SCRIPTS program is supplied in Appendix D. Once the synchronous parameters acceptable to the target are received in the Message In phase, an interrupt returns control to the interrupt service routine, which could program the clock dividers and the synchronous parameters in the SCTNL3 and SXFER registers. The parameters would then be saved for this synchronous device.

When this device is selected again, the SELECT FROM command could be used to indicate table indirect addressing. If table indirect addressing is used, the SCNTL3, SDID, and SXFER registers would be loaded from the table entry. If the device reselects the initiator, these parameters need to be reloaded into the registers before the data transfer begins. One method for loading them is to use a table indirect Select instruction with the alternate address jump programmed to the next instruction. This instruction must be executed after determining the ITLQ nexus and loading the DSA to point to the proper I/O data structure.

```
;ITLQ nexus complete and DSA loaded prior to performing
;this Select
SELECT FROM SCSI ID, REL(Next_Instr)

Next_Instr:
;begin I/O
```

The negotiated transfer information is stored in a table for use in later connections to a particular target. This information can be stored in the DSA table for use with table indirect Select and Reselect SCRIPTS instructions. The I/O command structure must have all four bytes contiguous in system memory, as shown below.

SCNTL3	SDID	SXFER	(00)

Interrupt Handling

The SCRIPTS processor in the SYM53C8XX performs most functions independently of the host microprocessor. However, certain interrupt situations must be handled by the external microprocessor. This section explains all aspects of interrupts as they apply to the SYM53C8XX.

Polling and Hardware Interrupts

The external microprocessor can be informed of an interrupt condition by polling or hardware interrupts. Polling means that the microprocessor must continually loop and read a register until it detects a bit set that indicates an interrupt. This method is the fastest, but it wastes CPU time that could be used for other system tasks. The preferred method of detecting interrupts in most systems is hardware interrupts. In this case, the SYM53C8XX will assert the Interrupt Request (IRQ/) line that will interrupt the microprocessor when an interrupt condition occurs, causing the microprocessor to execute an interrupt service routine. A hybrid approach can also be used that would use hardware interrupts for long waits, and polling for short waits.

Registers

The registers in the SYM53C8XX that are used for detecting or defining interrupts are the ISTAT, SIST0, SIST1, DSTAT, SIEN0, SIEN1, and DIEN.

ISTAT

The ISTAT is the only register that can be accessed as a slave during SCRIPTS operation, therefore it is the register that is polled when polled interrupts are used. It is also the first register that should be read when the IRQ/ pin has been asserted in response to a hardware interrupt. The INTF (Interrupt on the Fly) bit should be the first interrupt serviced. It must be written to one to be cleared. This interrupt must be cleared before servicing any other interrupts. If the SIP bit in the ISTAT register is set, then a SCSI-type interrupt has occurred and the SIST0 and SIST1 registers should be read. If the DIP bit in the ISTAT register is set, then a DMA-type interrupt has occurred and the DSTAT register should be read. SCSI-type and DMA-type interrupts may occur simultaneously, so in some cases both SIP and DIP may be set.

SIST0 and SIST1

The SIST0 and SIST1 registers contain the SCSI-type interrupt bits. Reading these registers will determine which condition or conditions caused the SCSI-type interrupt, and will clear that SCSI interrupt condition. If the SYM53C8XX is receiving data from the SCSI bus

and a fatal interrupt condition occurs, the SYM53C8XX will attempt to send the contents of the DMA FIFO to memory before generating the interrupt. If the SYM53C8XX is sending data to the SCSI bus and a fatal SCSI interrupt condition occurs, data could be left in the DMA FIFO. Because of this, the DMA FIFO Empty (DFE) bit in DSTAT should be checked. If this bit is clear, set the CLF (Clear DMA FIFO) and CSF (Clear SCSI FIFO) bits before continuing. The CLF bit is bit 2 in CTEST3. The FLF bit is bit 3 in CTEST3. The CSF bit is bit 1 in STEST3.

DSTAT

The DSTAT register contains the DMA-type interrupt bits. Reading this register will determine which condition or conditions caused the DMA-type interrupt, and will clear that DMA interrupt condition. Bit 7 in DSTAT, DFE (DMA FIFO Empty), is purely a status bit; it will not generate an interrupt under any circumstances and will not be cleared when read. DMA interrupts will flush neither the DMA nor SCSI FIFOs before generating the interrupt, so the DFE bit in the DSTAT register should be checked after any DMA interrupt. If the DFE bit is clear, then the FIFOs must be cleared by setting the CLF (Clear DMA FIFO) and CSF (Clear SCSI FIFO) bits, or flushed by setting the FLF (Flush DMA FIFO) bit.

SIEN0 and SIEN1

The SIEN0 and SIEN1 registers are the interrupt enable registers for the SCSI interrupts in SIST0 and SIST1.

DIEN

The DIEN register is the interrupt enable register for DMA interrupts in DSTAT.

DCNTL (SYM53C825A, 53C875, 53C876, 53C885, 53C895 only)

When bit 1 in this register is set, the IRQ/ pin will not be asserted when an interrupt condition occurs. The interrupt is not lost or ignored, but merely masked at the pin. Clearing this bit when an interrupt is pending will immediately cause the IRQ/ pin to assert. As with any register other than ISTAT, this register cannot be accessed except by a SCRIPTS instruction during SCRIPTS execution.

Fatal vs. Non-Fatal Interrupts

A fatal interrupt, as the name implies, always causes SCRIPTS to stop running. All non-fatal interrupts become fatal when they are enabled by setting the appropriate interrupt enable bit. Interrupt masking will be discussed later in this section. All DMA interrupts (indicated by the DIP bit in ISTAT and one or more bits in DSTAT being set) are fatal.

Some SCSI interrupts (indicated by the SIP bit in the ISTAT and one or more bits in SIST0 or SIST1 being set) are non-fatal. When the SYM53C8XX is operating in Initiator mode, only the Function Complete (CMP), Selected (SEL), Reselected (RSL), General Purpose Timer Expired (GEN), and Handshake to Handshake Timer Expired (HTH) interrupts are non-fatal. When operating in Target mode CMP, SEL, RSL, Target mode: SATN/ active (M/A), GEN, and HTH are non-fatal. Refer to the description for the Disable Halt on a Parity Error or SATN/ active (Target Mode Only) (DHP) bit in the SCNTL1 register to configure the chip's behavior when the SATN/ interrupt is enabled during Target mode operation. The Interrupt on the Fly interrupt is also non-fatal, since SCRIPTS can continue when it occurs.

The reason for non-fatal interrupts is to prevent SCRIPTS from stopping when an interrupt occurs that does not require service from the CPU. This prevents an interrupt when arbitration is complete (CMP set), when the SYM53C8XX has been selected or reselected (SEL or RSL set), when the initiator has asserted SATN/ (target mode: SATN/ active), or when the General Purpose or Handshake to Handshake timers expire. These interrupts are not needed for events that occur during high-level SCRIPTS operation.

Masking

Masking an interrupt means disabling or ignoring that interrupt. Interrupts can be masked by clearing bits in the SIEN0 and SIEN1 (for SCSI interrupts) interrupt enable registers or the DIEN (for DMA interrupts) interrupt enable register. How the chip will respond to masked interrupts depends on: whether polling or hardware interrupts are being used; whether the interrupt is fatal or non-fatal; and whether the chip is operating in initiator or target mode.

If a non-fatal interrupt is masked and that condition occurs, SCRIPTS will not stop, the appropriate bit in the SIST0 or SIST1 will still be set, the SIP bit in the ISTAT will not be set, and the IRQ/ pin will not be asserted. See the section on non-fatal vs. fatal interrupts for a list of the non-fatal interrupts.

If a fatal interrupt is masked and that condition occurs, then SCRIPTS will still stop, the appropriate bit in the DSTAT, SIST0, or SIST1 register will be set, and the SIP or DIP bits in the ISTAT will be set, but the IRQ/ pin will not be asserted.

When the chip is initialized, enable all fatal interrupts if you are using hardware interrupts. If a fatal interrupt is disabled and that interrupt condition occurs, SCRIPTS will halt and the system will never know it unless it times out and checks the ISTAT after a certain period of inactivity.

If you are polling the ISTAT instead of using hardware interrupts, then masking a fatal interrupt will make no difference since the SIP and DIP bits in the ISTAT inform the system of interrupts, not the IRQ/ pin.

Masking an interrupt after IRQ/ is asserted will not cause IRQ/ to be deasserted.

Stacked Interrupts

The SYM53C8XX has the ability to stack interrupts if they occur one after the other. If the SIP or DIP bits in the ISTAT register are set (first level), then there is already at least one pending interrupt and any future interrupts will be stacked in extra registers behind the SIST0, SIST1, and DSTAT registers (second level). When two interrupts have occurred and the two levels of the stack are full, any further interrupts will set additional bits in the extra registers behind SIST0, SIST1, and DSTAT. When the first level of interrupts is cleared, all the interrupts that came in afterward will move into the SIST0, SIST1, and DSTAT. After the first interrupt is cleared by reading the appropriate register, the IRQ/ pin will be deasserted for a minimum of three CLKs; the stacked interrupt(s) will move into the SIST0, SIST1, or DSTAT; and the IRQ/ pin will be asserted once again.

Since a masked non-fatal interrupt will not set the SIP or DIP bits, interrupt stacking will not occur as a result of a masked, non-fatal interrupt. A masked, non-fatal interrupt will still post the interrupt in SIST0, but will not assert the IRQ/ pin. Since no interrupt is generated, future interrupts will move right into the SIST0 or SIST1 instead of being stacked behind another interrupt. When another condition occurs that generates an interrupt, the bit corresponding to the earlier masked non-fatal interrupt will still be set.

A related situation to interrupt stacking is when two interrupts occur simultaneously. Since stacking does not occur until the SIP or DIP bits are set, there is a small timing window in which multiple interrupts can occur but will not be stacked. These could be multiple SCSI interrupts (SIP set), multiple DMA interrupts (DIP set), or multiple SCSI and multiple DMA interrupts (both SIP and DIP set).

As previously mentioned, DMA interrupts will not attempt to flush the FIFOs before generating the interrupt. It is important to set either the Clear DMA FIFO (CLF) and Clear SCSI FIFO (CSF) bits if a DMA interrupt occurs and the DMA FIFO Empty (DFE) bit is not set. This is because any future SCSI interrupts will not be posted until the DMA FIFO is clear of data. These ‘locked out’ SCSI interrupts will be posted as soon as the DMA FIFO is empty.

Halting in an Orderly Fashion

When an interrupt occurs, the SYM53C8XX will attempt to halt in an orderly fashion.

- If in the middle of an instruction fetch, the fetch will be completed, except in the case of a Bus Fault. Execution will not begin, but the DSP will point to the next instruction since it is updated when the current instruction is fetched.
- If the DMA direction is a write to memory and a SCSI interrupt occurs, the SYM53C8XX will attempt to flush the DMA FIFO to memory before halting. Under any other circumstances only the current cycle will be completed before halting, so the DFE bit in DSTAT should be checked to see if any data remains in the DMA FIFO.
- SCSI SREQ/SACK handshakes that have begun will be completed before halting.
- The SYM53C8XX will attempt to clean up any outstanding synchronous offset before halting.
- In the case of Transfer Control Instructions, once instruction execution begins it will continue to completion before halting.
- If the instruction is a JUMP/CALL WHEN/IF <phase>, the DSP will be updated to the transfer address before halting.
- All other instructions may halt before completion.

Sample Interrupt Service Routine

The following is a sample of an interrupt service routine for the SYM53C8XX. It can be repeated if polling is used, or should be called when the IRQ/ pin is asserted if hardware interrupts are used.

1 Read ISTAT.

2 If the INTF bit is set, it must be written to a one to clear this status.

3 If only the SIP bit is set, read SIST0 and SIST1 to clear the SCSI interrupt condition and get the SCSI interrupt status. The bits in the SIST0 and SIST1 tell which SCSI interrupt(s) occurred and determine what action is required to service the interrupt(s).

4 If only the DIP bit is set, read the DSTAT to clear the interrupt condition and get the DMA interrupt status. The bits in the DSTAT will tell which DMA interrupt(s) occurred and determine what action is required to service the interrupt(s).

5 If both the SIP and DIP bits are set, read SIST0, SIST1, and DSTAT to clear the SCSI and DMA interrupt condition and get the interrupt status. If using 8-bit reads of the SIST0, SIST1, and DSTAT registers to clear interrupts, insert a 12 CLK delay between the consecutive reads to ensure that the

interrupts clear properly. Both the SCSI and DMA interrupt conditions should be handled before leaving the ISR. It is recommended that the DMA interrupt be serviced before the SCSI interrupt, because a serious DMA interrupt condition could influence how the SCSI interrupt is acted upon.

When using polled interrupts, go back to step 1 before leaving the interrupt service routine, in case any stacked interrupts moved in when the first interrupt was cleared. When using hardware interrupts, the IRQ/ pin will be asserted again if there are any stacked interrupts. This should cause the system to re-enter the interrupt service routine. The example program in Appendix F contains an interrupt service routine for an initiator; Appendix G is a sample interrupt service routine that is more typical for a target device.

Migrating Existing Software to Ultra and Ultra2 SCSI

Ultra SCSI and Ultra2 SCSI extend the Fast SCSI-2 specification to allow synchronous transfer periods to be negotiated down as low as 50 ns (Ultra SCSI) or 25 ns (Ultra2 SCSI). This allows a maximum transfer rate of 20 MB/s on an 8-bit SCSI bus or 40 MB/s on a wide SCSI bus for Ultra SCSI, and 40 MB/s on an 8-bit bus or 80 MB/s on a wide SCSI bus for Ultra2 SCSI. Refer to Chapter 1 to determine which members of the SYM53C8XX family support Ultra SCSI and Ultra2 SCSI.

To achieve Ultra SCSI or Ultra2 SCSI transfer rates, existing software programs must be updated to reflect changes in the following areas of the SYM53C8XX (additional minor changes may be needed to migrate existing software to support all the features in the new device):

1. SCNTL3 Register CCF bits - adjust the bit values to reflect the desired clock divider

2. SCNTL3 Register SCF bits - adjust the bit values to reflect the SCLK frequency, doubled or quadrupled if applicable

3. SXFER Register XFERP bits - adjust the bit values to reflect the desired divider values for the synchronous period

4. Adjust the Clock input as required for the SCSI processor being used
   - With the SYM53C860, add an external 80MHz SCSI clock

- With the SYM53C875, use an 80 MHz external SCSI clock or use an external 40 MHz clock and enable the SCSI clock doubler
- With the SYM53C895, use an 80MHz clock for Ultra SCSI or use an external 40 MHz clock with the clock quadrupler for Ultra2 SCSI

  Note: the SYM53C885 and SYM53C876 require a 40MHz clock and use of the clock doubler.

5 Ultra Enable bit, SCNTL3 register - set this bit to enable Ultra SCSI or Ultra2 SCSI transfers

Clock Divider Bits

Two registers are used to divide down the clock. The first is the SCNTL3 register. The CCF bits in this register determine the SCSI core speed used for asynchronous transfers and any other timings (such as selection time-out). These bits are set based on the input clock frequency and do not change. The SCF bits are used to determine the timings for synchronous transfers and can be changed whenever the SYM53C8XX connects to a different device on the SCSI bus.

The SCF bits in the SCNTL3 register are used in conjunction with the XFERP bits in the SXFER register to determine the synchronous period. To get a transfer rate of 10 MB/s with a 40 MHz clock, program the SCF bits to 001 for a divide by one factor and then program the XFERP bits for 000 for a divide by 4 factor. 40 MHz divided by 1 and then divided by 4 is 10 MB/s. Other combinations of these two sets of bits can be used to select a variety of synchronous transfer rates. For more information on the bit combinations that are supported, see the clock divider bit descriptions in the chip data manuals.

Ultra Enable Bit

The Ultra Enable bit (also known as the Fast-20 Enable bit) adjusts the chip's timings to be compliant with the Fast-20 proposed standard. It should be set when the synchronous transfer period is less than 100 ns and cleared when it is greater than or equal to 100 ns.

Loading the New Register Values

Since the Ultra Enable bit and the clock dividers are in the SCNTL3 and SXFER registers, these registers can be automatically loaded during a selection or reselection by using Table Indirect Addressing. This allows the chips to transparently talk with a combination of Ultra SCSI, Ultra2 SCSI, and Fast SCSI devices on the same SCSI bus.

Negotiating Synchronous Transfers

The easiest way to calculate the synchronous transfer period is by multiplying the clock period by the clock divider values. For example, a 40 MHz clock is a 25 ns period. (25 ns)*(1)*(4)=100ns, which is the Fast SCSI-2 synchronous transfer period.

If you are running an 80 MHz clock (12.5 ns period) and only negotiated for Fast SCSI-2 rather than Ultra SCSI, the SCF bits would need to be programmed for SCLK/2 and the XFERP bits for 4 which would be (12.5ns)*(2)*(4)=100ns.

The SCSI-2 specification states that synchronous transfer rates must be a multiple of 4 ns. However with an 80 MHz clock, the period must be a multiple of 12.5 ns. Ultra SCSI is defined to be a 20 mega-transfer per second maximum, which would be a 50 ns period. Since 50 ns is not a multiple of 4, most SCSI devices cannot negotiate for this exact rate. Unless future revisions of the standard make a different recommendation, most devices will probably negotiate for a 48 ns period. The SYM53C8XX cannot be programmed for a 48 ns period since it is not a multiple of 12.5 ns, so driver programs should specify a 50 ns period and the chip should negotiate for a 48 ns period. This is acceptable because the SCSI-2 specification allows you to transfer data at a slower rate than what you negotiated for, but not faster.

To program the chip for a full Ultra SCSI transfer rate of 50 ns using the required 80 MHz clock, program the SCF bits for SCLK/1 and select an XFERP of 4. This comes out to (12.5ns)*(1)*(4)=50 ns.

Using the SCSI Clock Doubler

The SYM53C875, SYM53C876, and SYM53C885 can double the frequency of a 40-50 MHz SCSI clock, allowing the system to perform Ultra SCSI transfers in systems that do not have 80 MHz clock input. This option is user-selectable with bit settings in the STEST1, STEST3, and SCNTL3 registers. At power-on or reset, the doubler is disabled and powered down. Follow these steps to use the clock doubler:

1. Set the SCLK Doubler Enable bit (STEST1, bit 3)
2. Wait 20 µs
3. Halt the SCSI clock by setting the Halt SCSI Clock bit (STEST3 bit 5)
4. Set the clock conversion factor using the SCF and CCF fields in the SCNTL3 register
5. Set the SCLK Doubler Select bit (STEST1, bit 2)
6. Clear the Halt SCSI Clock bit

Using the SCSI Clock Quadrupler

The SYM53C895 can quadruple the frequency of a 40 MHz SCSI clock, allowing the system to perform Ultra2 SCSI transfers. This option is user-selectable with bit settings in the STEST1, STEST3, and SCNTL3 registers. At power-on or reset, the quadrupler is disabled and powered down. Follow these steps to use the clock quadrupler:

1. Set the SCLK Quadrupler Enable bit (STEST1, bit 3)
2. Poll bit 5 of the STEST4 register. The SYM53C895 sets this bit as soon as it locks in the 160 MHz frequency. The frequency lockup takes approximately 100 microseconds.
3. Halt the SCSI clock by setting the Halt SCSI Clock bit (STEST3 bit 5)
4. Set the clock conversion factor using the SCF and CCF fields in the SCNTL3 register
5. Set the SCLK Quadrupler Select bit (STEST1, bit 2)
6. Clear the Halt SCSI Clock bit

Using the SCRIPTS RAM

The SYM53C825A, SYM53C875, SYM53C876, SYM53C885, and SYM53C895 have 4k bytes (1k x 32 bit) of internal, general purpose RAM. The RAM is designed to store SCRIPTS instructions and I/O data structure information, but is not limited to this type of information. When the chip fetches SCRIPTS instructions or Table Indirect information from the internal RAM, these fetches remain internal to the chip and do not use the PCI bus. Other types of access to the RAM by the SYM53C825A/53C875/53C876/53C885/ 53C895 use the PCI bus, as if they were external accesses. This section discusses loading of SCRIPTS and Table Indirect information into the SCRIPTS RAM, and programming techniques when using the RAM.

The RAM can be relocated by the PCI system BIOS anywhere in 32-bit address space. The RAM Base Address register, located in the chip's PCI configuration space, contains the base address of the internal RAM. This register is similar to the ROM Base Address register in the PCI Configuration register set. To simplify loading of SCRIPTS instructions, the base address of the RAM will appear in the SCRATCHB register when bit 3 of the CTEST2 register is set. The RAM is byte-accessible from the PCI bus and will be visible to any bus mastering device on the bus. Accesses made externally (i.e. by the CPU) follow the same timing sequence as a standard slave register access, except that the target wait states required will drop from 5 to 3.

Loading SCRIPTS RAM

SCRIPTS instructions can be loaded into the internal RAM in one of two ways. The first way is to simply copy the instructions into the RAM with the CPU. Another method is to use a MOVE MEMORY instruction, which copies the SCRIPTS instructions from their initial location in host memory to the SCRIPTS RAM. This method is especially useful in the Intel processor real mode of operation, because the SCRIPTS RAM is generally mapped by PCI system BIOS outside the region where the processor can access it. The syntax of the move instruction is as follows:

```
MOVE MEM Script_Inst_Bytes, SRC_Phys_Addr, \Script_RAM_Phys_Addr
```

Where `Script_Inst_Bytes` is the number of bytes of instructions to copy, `SRC_Phys_Addr` is the physical starting address of the static SCRIPTS array to be copied into SCRIPTS RAM, and `Script_RAM_Phys_Addr` is the physical base address of the SCRIPTS RAM itself (found in the SCRATCHB register). To create data structures such as table indirect tables, create a pointer to the

location in SCRIPTS RAM that will be used to store the data. An example of this is in Figure 9-1.

Figure 9-1
Storing Data Structures in SCRIPTS RAM

```
struct _table {/* Table indirect entry */
  uquad count;
  uquad address;
};
typedef struct table;
#define SCRAM_TABLE_OFFSET 0xC00; /* Locate table info at bottom
1K of SCRIPTS
       RAM*/
void main() {
table *buffer_table;  /* pointer to table indirect entries */
ulong SCRAM_Phys_Addr;
ulong Table_Phys_Addr;
   /* Get RAM physical address */
outpw(ChipIOBase+CTEST2,  0x08);/* Set bit 3 */
/* Get RAM Base in ScratchB */
SCRAM_Phys_Addr = (ulong) ((ulong) (inpw(ChipBaseIO+
SCRATCHB2) << 16) | inpw(ChipIOBase+SCRATCHB));  /* Read Reg*/
outpw(ChipIOBase+CTEST2, 0x00);/* Clear bit 3 */
/* Create pointer to RAM for Table */
Table_Phys_Addr = SCRAM_Phys_Addr + SCRAM_Table_Offset;
buffer_table = PhystoVirt(Table_Phys_Addr);
}
```

The routine “PhystoVirt” should convert the physical address of the table location in the SCRIPT RAM to a virtual address that can be used as a pointer in “C”.

Programming Techniques when Using SCRIPTS RAM

SCRIPTS programs may be stored on the chip, outside the chip, or both internally and externally. When SCRIPTS code is located both internally and externally, the following techniques will allow the internal SCRIPTS to successfully communicate with the external SCRIPTS and vice versa.

1. Two source (`.SS`) files should be created, one with the SCRIPTS programs that are to be located externally and the other with the SCRIPTS programs that are to be located internally.

2. The internal and external SCRIPTS programs must be given unique array identifiers by using the PROC statement at the beginning of each, so that both can be linked into a driver. This causes the compiler to generate SCRIPTS arrays without the default SCRIPTS name.

3. Both source files should be compiled with the `-p` option instead of the `-o` option. This prevents the generation of data structures which share common names between the two files, causing a 'C' compile time conflict with both files being linked into a driver.

4. All jumps between the internal and external SCRIPTS routines should be absolute and should use EXTERNs as the destination address variable. This allows the proper jump address to be patched once the base addresses of both SCRIPTS programs have been established at run time.

5. Any labels that are to be jumped to from the opposite SCRIPTS program should be defined as entry points with the ENTRY declarative. This causes the compiler to provide the proper offset information in the compiled output file so that physical addresses can be resolved at runtime.

6. All labels, externs, and relative buffers should have unique names in each SCRIPTS program to prevent 'C' compile time conflicts.

7. All jumps which move within the same SCRIPTS program should use the REL modifier.

8. The file that contains the internal SCRIPTS program should be processed by the RAMFIX utility to eliminate any other conflicts between the two files. The RAMFIX utility is included on the diskette enclosed in this programming guide, or it can be downloaded from the Symbios Logic BBS.

Figure 9-2 through Figure 9-5 are the internal and external `SCRIPTS.LIS` and `.OUT` files, and illustrate the interactions between the two. Certain parts of the program text appear in bold type to highlight the coding differences when both internal and external RAM are used for SCRIPTS program storage. The numbered notes at the end of each example program reference numbered items in the far left column of the program text.

Figure 9-2
External Script (.LIS):

```
     1 ARCH 825A
     2
     3 ABSOLUTE done=0xff
     4
1:   5 EXTERN Int_Start
     6 EXTERN Int_dataout
     7
2:   8 ENTRY Ext_Start
     9 ENTRY Ext_done
    10
3:  11 00000000:          PROC Ext_Script:
    12 00000000:               Ext_Start:
    13 00000000: 78344500 00000000   MOVE 0x45 to SCRATCHA0
    14 00000008: 78354600 00000000   MOVE 0x46 to SCRATCHA1
    15 00000010: 80880000 00000018   JUMP REL(Entry_point)
    16 00000018: 78364700 00000000   MOVE 0x47 to SCRATCHA2
    17 00000020: 78374800 00000000   MOVE 0x48 to SCRATCHA3
4:  18 00000028: 80080000 00000000   JUMP Int_Start
    19
    20 00000030:                   Entry_point:
    21 00000030: 6A360000 00000000   MOVE SFBR to SCRATCHA2
    22 00000038: 785C0000 00000000   MOVE 0x00 to SCRATCHB0
    23 00000040: 785D0100 00000000   MOVE 0x01 to SCRATCHB1
    24 00000048: 785F0200 00000000   MOVE 0x02 to SCRATCHB3
5:  25 00000050: 80080000 00000000   JUMP Int_dataout
    26
    27 00000058:                   Ext_done:
    28 00000058: 98080000 000000FF   INT done
    29
    30
```

NOTES:

1: Jump labels that are located in the internal SCRIPTS program are defined as EXTERNs to facilitate patching at driver runtime.

2: Labels that will be jumped to from the internal SCRIPTS program are defined as ENTRYs to facilitate patching at driver run time.

3: PROC directive used to override the default SCRIPTS array name and replace it with Ext_Script

4: This is a jump to a location in the internal SCRIPTS program and should be patched at driver init time.

5: This is a jump to a location in the internal SCRIPTS program and should be patched at driver init time.

Figure 9-3
External Script (.OUT):

```
typedef unsigned long ULONG;
1:ULONGExt_Script[] = {
   0x78344500L,0x00000000L,
   0x78354600L,0x00000000L,
   0x80880000L,0x00000018L,
   0x78364700L,0x00000000L,
   0x78374800L,0x00000000L,
   0x80080000L,0x00000000L,
   0x6A360000L,0x00000000L,
   0x785C0000L,0x00000000L,
   0x785D0100L,0x00000000L,
   0x785F0200L,0x00000000L,
   0x80080000L,0x00000000L,
   0x98080000L,0x000000FFL
};
2:ULONG E_Int_dataout_Used[] = {
   0x00000015L
};
ULONG E_Int_Start_Used[] = {
   0x0000000BL
};
#define A_done0x000000FFL
3:#define Ent_Ext_done         0x00000058L
#define Ent_Ext_Start        0x00000000L
```

NOTES:

1: The use of the PROC statement has forced the array to be named Ext_Script instead of SCRIPT so that a compile time conflict is avoided.

2: The offsets in these data structures indicate where the internal SCRIPTS jump address should be patched.

3: These are the offsets into the Ext_Script array of the entry points that are being jumped to from the internal SCRIPTS program. They are used to calculate the internal to external jump physical addresses to be patched into the internal SCRIPTS program.

Figure 9-4
Internal Script (.LIS):

```
   1   ARCH 825A
   2
   3   ABSOLUTE scsi_id=0x00
   4   ABSOLUTE resel=0x01
   5
   6   EXTERN identify_buf={0x80}
   7   EXTERN cmd_buf=6{??}
   8   EXTERN data_buf=512{??}
   9   EXTERN stat_buf=1{??}
   10  EXTERN msgin_buf=1{??}
   11
1:12   EXTERN Ext_Start
   14
2:15   ENTRY Int_Start
   16  ENTRY Int_dataout
   17
3:18 00000000: PROC Int_Script:
   19 00000000: Int_Start:
   20 00000000: 45000000 00000058   SELECT ATN scsi_id,
REL(reselected)
   21 00000008:                    ident:
   22 00000008: 86830000 00000008   JUMP REL(send_cmd), WHEN
NOT MSG_OUT
   23 00000010: 0E000001 00000000   MOVE 1, identify_buf, WHEN
MSG_OUT
   24 00000018:                    send_cmd:
   25 00000018: 0A000006 00000000   MOVE 6, cmd_buf, WHEN CMD
4:26 00000020: 80080000 00000000   JUMP Ext_Start
   27 00000028:                    Int_dataout:
   28 00000028: 08000200 00000000   MOVE 512, data_buf, WHEN
DATA_OUT
   29 00000030:                    stat:
   30 00000030: 0B000001 00000000   MOVE 1, stat_buf, WHEN
STATUS
   31 00000038:                    msgin:
   32 00000038: 0F000001 00000000   MOVE 1, msgin_buf, WHEN
MSG_IN
   33
```

```
   34 00000040:                    complete:
  35 00000040: 7C027F00 00000000   MOVE SCNTL2 & 0x7F to SCNTL2
   36 00000048: 60000040 00000000    CLEAR ACK
   37 00000050: 48000000 00000000    WAIT DISCONNECT
5:38 00000058: 80080000 00000000    JUMP Ext_done
   39
   40 00000060:                    reselected:
   41 00000060: 98080000 00000001    INT resel
  42
```

NOTES:

1: Jump labels that are located in the external SCRIPTS program are defined as EXTERNs to facilitate patching at driver run time.

2: Labels that will be jumped to from the external SCRIPTS program are defined as ENTRYs to facilitate patching at driver run time.

3: PROC directive used to override the default SCRIPTS array name and replace it with Int_Script

4: This is a jump to a location in the external SCRIPTS program and should be patched at driver init time.

5: This is a jump to a location in the external SCRIPTS program and should be patched at driver init time.

Figure 9-5
Internal SCRIPTS Program (.OUT):

```
typedef unsigned long ULONG;
1:ULONGInt_Script[] = {
   0x45000000L,0x00000058L,
   0x86830000L,0x00000008L,
   0x0E000001L,0x00000000L,
   0x0A000006L,0x00000000L,
   0x80080000L,0x00000000L,
   0x08000200L,0x00000000L,
   0x0B000001L,0x00000000L,
   0x0F000001L,0x00000000L,
   0x7C027F00L,0x00000000L,
   0x60000040L,0x00000000L,
   0x48000000L,0x00000000L,
   0x80080000L,0x00000000L,
   0x98080000L,0x00000001L
};
ULONG E_cmd_buf_Used[] = {
```

```
    0x00000007L
};
ULONG E_data_buf_Used[] = {
    0x0000000BL
};
2:ULONG E_Ext_Start_Used[] = {
    0x00000009L
};
ULONG E_Ext_done_Used[] = {
    0x00000017L
};
ULONG E_identify_buf_Used[] = {
    0x00000005L
};
ULONG E_msgin_buf_Used[] = {
    0x0000000FL
};
ULONG E_stat_buf_Used[] = {
    0x0000000DL
};
#define A_scsi_id0x00000000L
#define A_resel0x00000001L
3:
#define Ent_Int_dataout      0x00000028L
#define Ent_Int_Start        0x00000000L
```

NOTES:

1: The use of the PROC statement has forced the array to be named Int_Script instead of SCRIPT so that a compile time conflict is avoided.

2: The offsets in these data structures indicate where the internal SCRIPTS jump address should be patched.

3: These are the offsets into the Int_Script array of the entry points that are being jumped to from the external SCRIPTS program. They are used to calculate the external to internal jump physical addresses to be patched into the external SCRIPTS program.

Patching Internal and External SCRIPTS Programs

The following routine will patch the correct values into the above two SCRIPTS programs so that they can interact properly. The following assumptions are made in this routine:

1. The Int_Script array was copied into the SCRIPTS RAM at the starting location of the RAM.
2. The Ext_Script is already 32-bit aligned.
3. The variable ChipIOBase contains the IO base address of the chips register set.
4. VirttoPhys is a routine that will convert a virtual pointer to a physical address.

```
void main() {
ulong Int_Script_Phys_Addr;
ulong Ext_Script_Phys_Addr;

/* Get RAM physical address, which is assumed to be */
/* the internal SCRIPTS physical address */
outpw(ChipIOBase+CTEST2, regval | 0x08);/* Set bit 3 to get
             RAM Base in ScratchB*/
Int_Script_Phys_Addr = (ulong) ((ulong) (inpw(ChipBaseIO+
SCRATCHB2) << 16) | inpw(ChipIOBase+SCRATCHB));  /* Read Reg*/
outpw(ChipIOBase+CTEST5, 0x00);/* Clear bit 3 */

Ext_Script_Phys_Addr = (ulong) VirttoPhys(Ext_Script);
/* Patch External Script entries */
Ext_Script[E_Int_dataout_Used[0]] = Int_Script_Phys_Addr +
Ent_Int_dataout;
Ext_Script[E_Int_Start_Used[0]] = Int_Script_Phys_Addr +
                         Ent_Int_Start;
/* Patch Internal SCRIPTS entries */
/* The cmd_buf, data_buf, identify_buf, stat_buf */
/* and msgin_buf should also be done but they will not be */
/* shown in this example as they are not pertinent */
Int_Script[E_Ext_done_Used[0]] = Ext_Script_Phys_Addr +
Ent_Ext_done;
Int_Script[E_Ext_Start_Used[0]] = Ext_Script_Phys_Addr +
Ent_Ext_Start;
}
```

Chapter 10

Multi-Threaded I/O

Overview

The SYM53C8XX allows multi-threaded I/O operations with minimal external processor intervention, in systems that support multi-tasking. Multi-threaded algorithms must be used any time more than one task is active in the system. Figure 10-1 shows a typical situation where multiple tasks are accessing multiple devices simultaneously. The path between Task 1 and Disk 2 is highlighted to show how information might be transferred. The device driver must schedule and control the I/O requests based on such considerations as what devices are available, and the relative priorities of the requests.

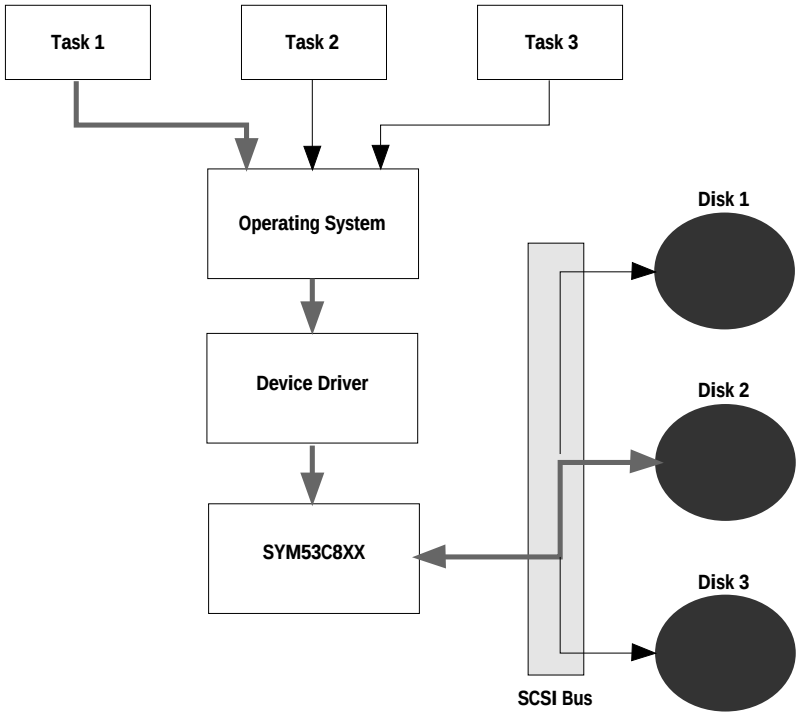


Figure 10-1
Multi-threaded System Operation

Multi-threaded algorithms are similar to single threaded algorithms with disconnects, but a new element, called the scheduler, is added. The scheduler keeps track of SCSI bus operations when more than one task is active at a time. SCRIPTS code must be stored in RAM to allow multi-threaded operation, because SCRIPTS and the CPU must modify SCRIPTS dynamically. A multi-threaded SCRIPTS algorithm contains three parts: the main SCRIPTS, the scheduler SCRIPTS, and the reselect SCRIPTS. These areas are described in detail after the overview of the steps that occur in a multi-threaded I/O. This example shows how to implement a scheduler in SCRIPTS. This is only one method of implementing a scheduler. Many users choose to schedule I/Os in an upper layer, such as in the "C" driver code.

Multi-threaded Operations Flow

Figure 10-2 demonstrates the sequential flow of steps in a multi-threaded operation. The heavy lines in the figure represent the initial flow of information for a new operation. The lighter weight lines represent the flow as the chip finishes pending steps of a multi-threaded operation.

To begin a multi-threaded operation, the user application determines that an I/O is needed, and makes an I/O request of the operating system. The operating system then sets up and starts the appropriate device driver. The main driver program modifies the SCSI scheduler routine to call the appropriate I/O SCRIPTS instructions. At that point, normal processing continues as the SYM53C8XX executes the instructions of the SCRIPTS routine.

When the CPU issues a request for service, it writes a JUMP to the scheduler to start the I/O. The SYM53C8XX selects the SCRIPT needed to perform the requested action. That SCRIPTS instruction then writes a NOP to the scheduler to prevent the same I/O from being re-started. The number of entries (JUMPs) in the scheduler at any one time will be the number of scheduled but not started. The chip then executes the SCRIPTS subroutine and interrupts at completion.

When the SYM53C8XX has no more instructions to execute, it jumps to the scheduler SCRIPTS area. If no new I/Os are scheduled, the SYM53C8XX jumps to a WAIT RESELECT instruction. If a new I/O is scheduled, the chip will then execute the JUMP instruction in the scheduler entry that corresponds to the SCSI ID of the target device, to go to the main SCRIPTS area.

If the chip's operation must halt until another peripheral device retrieves data, a Wait RESELECT SCRIPT is executed. When the chip is reselected by the target, it resumes execution of the main I/O routine While the chip waits to be reselected by the target device, the CPU may call the chip by setting the SIGP bit. The SYM53C8XX schedules a new I/O and repeats the cycle described above.

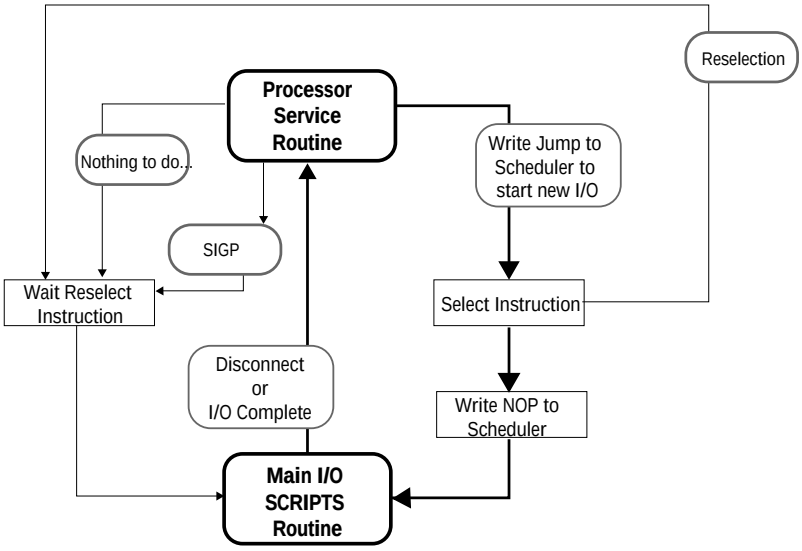


Figure 10-2
Multi-threaded SCRIPTS Operational Flow

SCRIPTS Areas

The main SCRIPTS area contains the SCRIPTS to perform the standard operations associated with a SCSI command, such as transferring messages, commands, and data. The scheduler SCRIPTS area contains a three-SCRIPTS entry for each job the CPU schedules. The scheduler is modified at run time. When the operating system interface receives an I/O request, it creates an area in host memory for the scheduler information for that request. It tracks each request it receives. New requests are classified outstanding as they are processed and performed. Upon completion of the I/O request, the hardware interface returns a completed status to the operating system interface which then updates the status of the request. The reselect SCRIPTS area is the portion of SCRIPTS code that is used after the target disconnects and the SYM53C8XX is waiting to be reselected.

Multi-Threaded SCRIPTS Example

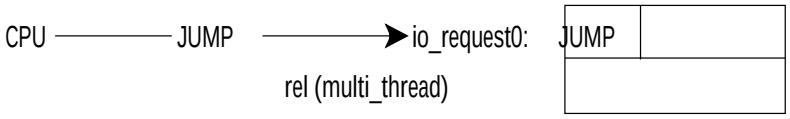
An example SCRIPTS operation for the SYM53C8XX is illustrated below. This example is for multi-threaded I/O where only one command is sent to each target at a time. To send more than one command to any target, tagged command queueing must be used. For more complex situations such as this, it may be preferable to use

"C" code for scheduling I/Os. The SCRIPTS program must be modified to look at the queue tag messages, and there must be a DSA table entry for each possible outstanding tagged command per target ID, instead of just one per target ID as in this example. This program appears in its entirety in Appendix C.

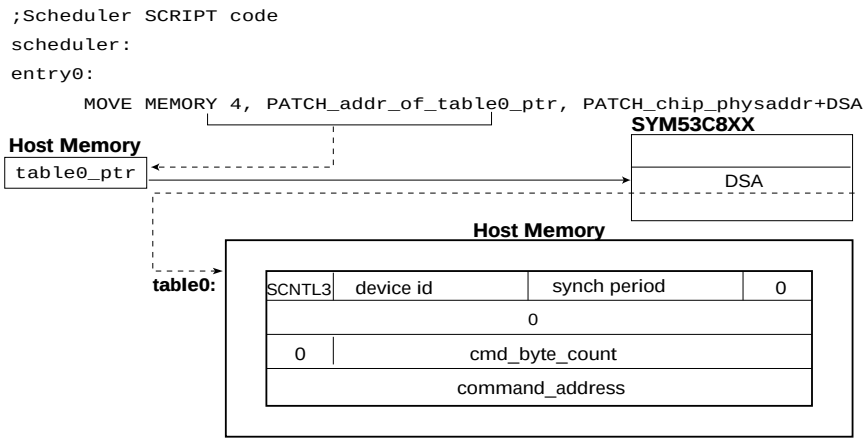
Any item in the code examples that is preceded by "PATCH_" needs to be patched by the driver. Patching only occurs once, when the driver is initially loaded. After initialization, all required addresses are in the SCRIPTS array. For more information on instruction patching, refer to Chapter 7.

Both dashed and solid lines are used in some of the program illustrations. The dashed lines indicate pointers, and the solid lines indicate data movement in the direction indicated by the arrows.

1 The first step occurs when the CPU writes a JUMP into the io_requestX scheduler slot



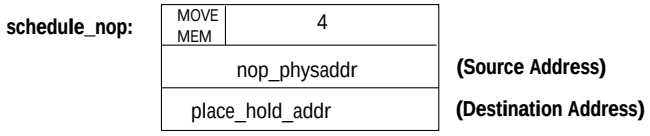
2 Next, the CPU may need to set the SIGP bit in the SYM53C8XX to indicate that an I/O needs to be processed. As soon as this happens, the SIOP will JUMP to the scheduler. The first instruction in the scheduler will set up the DSA to point to the correct table in the example.



3 Table0 has the nexus information about any previously negotiated synchronous transfer period and offset. It also contains the SCSI ID of the target device. Clock divider information for the SCNTL3 register would also be included in this table. Also, the operating system will build the command and other buffer information into this table prior to starting this I/O.

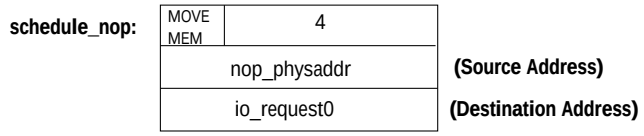
Next, the SCRIPTS instruction will move the address of the IO_requestX into the schedule_nop SCRIPTS destination address field. This will allow the multi-threaded SCRIPTs instruction to write a NOP into the io_requestX location in the scheduler to indicate that the I/O has been started.

Before:




```
MOVE MEMORY 4, PATCH_SCRIPTphysaddr+io_request0,
              PATCH_SCRIPTphysaddr+schedule_NOP+8
```

After:



4 The final task of the scheduler is to jump to the multi-thread SCRIPTS subroutine.

5
```
io_request0:
JUMP rel (multi_thread)
```

6 The main SCRIPTS routine will execute a Select With Attention instruction to connect to the appropriate SCSI device.

```
;Main SCRIPT code
multi_thread:
 SELECT ATN FROM SCSI_id, REL (wait_for_reselect)
```

7 Once the two devices are connected, the SCRIPTS instruction must write the NOP into the scheduler routine to avoid trying to start the I/O again. This is accomplished using a Memory to Memory Move command. The source address will be the address of a NOP SCRIPTS instruction. The destination address is the io_requestX location that was patched into place_hold_addr in the scheduler.

```
schedule_nop:
          MOVE MEMORY 4, PATCH_nop_physaddr,PATCH_place_hold_addr
```

8 Next, the will continue just as in single-threaded mode until a disconnect occurs.

```
 JUMP REL (to_decisions), WHEN NOT MSG_OUT
id_msg_out:
 MOVE FROM identify_msg_buf, WHEN MSG_OUT
 .
 .
 .
```

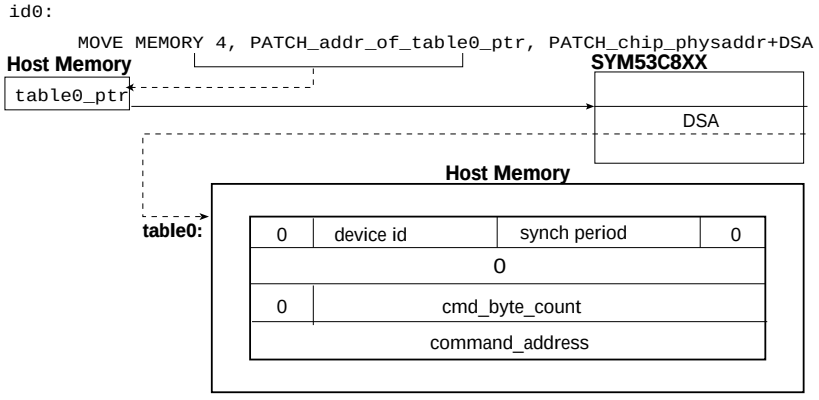
9 Upon disconnection, the initiator will jump to the wait_for_reselect SCRIPT. It will then wait for any device that had previously disconnected to reconnect. If a reselect occurs, the code will continue. If the device gets selected or the processor issues a SIGP, the SCRIPTS will continue at the alternate jump address. Setting the SIGP bit allows the processor to start a new I/O, instead of just waiting for a previous I/O to reconnect.

```
;Reselected SCRIPT code
wait_for_reselect:
 WAIT RESELECT REL (CPU_set_SIGP)
```

10 Once the initiator is reselected it is necessary to determine which SCSI ID has reselected it. In the SYM53C8XX, the ID of the device that reselected the SYM53C8XX is in the SSID register.

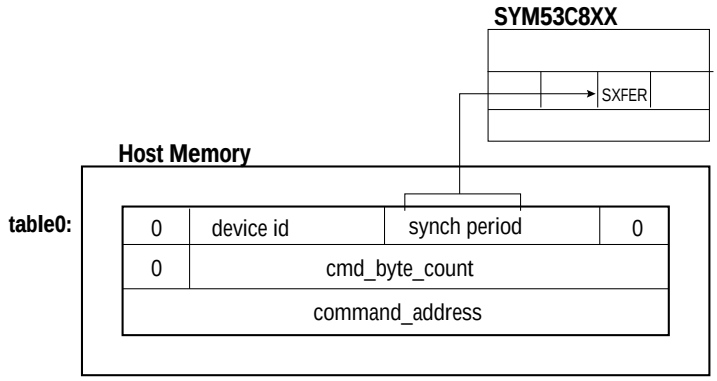
```
SCSI_id_jump_table:
 MOVE SSID to SFBR
 JUMP REL (id_0), IF 0x00
 JUMP REL (id_1), IF 0x01
 JUMP REL (id_2), IF 0x02
 INT reselect_id_error
```

11 Next, the DSA will need to be written to the address of the correct table, depending on which SCSI ID has reselected the initiator.



12 Upon reselection, it is not necessary to re-negotiate for synchronous data transfer parameters. These can be restored to the SXFER register and the SCNTL3 register from the information stored in the table.

```
id0:
      MOVE MEMORY 1, PATCH_addr_of_table0 + 2, PATCH_chip_physaddr+SXFER
```



13 Once the DSA is pointing to the correct table, table indirect SCRIPTS can be used to receive the identify message.

```
MOVE FROM identify_msg_buf, WHEN MSG_IN
CLEAR ACK
```

14 Finally, the SCRIPT will continue with a normal I/O until it has completed.

```
JUMP REL (to_decisions)
id_1:
MOVE MEMORY 4, PATCH_addr_of_table1_ptr,
PATCH_chip_physaddr+DSA
MOVE MEMORY 1, PATCH_addr_of_table 1+2,
PATCH_chip_physaddr+SXFER
MOVE FROM identify_msg_buf, WHEN MSG_IN
CLEAR ACK
JUMP REL (to_decisions)
id_2:
MOVE MEMORY 4, PATCH_addr_of_table 2_ptr,
PATCH_chip_physaddr+DSA
MOVE MEMORY 1, PATCH_addr_of_table 2+2,
PATCH_chip_physaddr+SXFER
MOVE FROM identify_msg_buf, WHEN MSG_IN
CLEAR ACK
JUMP REL (to_decisions)
```

Using the SIGP bit to Abort an Instruction

The SIGP (Signal Process) bit in the ISTAT register is used to pass a flag to a running SCRIPTS instruction. The SIGP bit is used to signal that an I/O is ready for execution and has already been scheduled by the host processor. The only SCRIPTS instructions directly affected by this bit are Wait Select and Wait Reselect. Setting the SIGP bit causes the instruction to jump immediately to the alternate address. For more information on this bit, refer to the SYM53C8XX data manuals. The following SCRIPTS code is an example of how to use the SIGP bit when attempting to abort a Wait Reselect or Wait Select instruction, assuming that the device is in the initiator role.

```
;***********************************
reselect_entry:
 WAIT RESELECT alt_sig_p
;  if here, got reselected
handle_resel:
         *
         *
         *

;***********************************
selected_entry:
 WAIT SELECT alt_sig_p
;  if here, got selected
handle_sel:
         *
         *
         *

;***********************************
alt_sig_p:
;  We assume that the sig_p bit was set,
;  and a reselection needs to be performed.
;  If here because of a selection or
;  reselection or if a selection or
;  reselection occurred during the jump after
;  sig_p bit was set, the alternate address
;  'sel_resel' will be taken.
:  Setup relevant information for this IO.
 RESELECT FROM scsi_id, sel_resel
;  if here, sig_p was set and there was no
;  selection or reselection

 MOVE CTEST2 TO SFBR
;  clear sig_p bit

 MOVE FROM ident_msg, WITH MSG_IN
;  from this point a reselection is performed
```

```
;  as normal by moving through the SCSI phases
        *
        *
        *


;***********************************
sel_resel:
;  if here, we have been selected or reselected
;  and sig_p may or may not have been set.
 MOVE SIST0 & 0x20 TO SFBR
;  get selected bit

 JUMP sel, IF 0x20
;  if we got selected

 MOVE SIST0 & 0x10 TO SFBR
;  get reselected bit

 JUMP resel, IF 0x10
;  if we got reselected

 INT sel_resel_error
;  big error, should have been selected
;  or reselected

;***********************************
sel:
;  if here, selection occurred and sig_p may or
;  may not have been set.  But process selection
;  no matter what.
 JUMP handle_sel

;***********************************
resel:
;  if here, reselection occurred and sig_p may or
;  may not have been set.  But process reselection
;  no matter what.
 JUMP handle_resel:
```

I/O Completion

When the SYM53C8XX completes an I/O, it must inform the host system. Upon completion of an I/O, the programmer may want to signal the system processor in one of several ways:

- Write to an address to generate an external interrupt. This allows completely interrupt-driven software.
- Write to memory to signal the I/O driver. The driver would then poll the memory location, or, optionally, a general purpose output pin could be used to tell the processor the location contains information. For example, the `status_buf` or `msg_in_buf` would be polled for good status or command complete to signal an I/O had completed.

Example:

```
MOVE 1, status_buf, WHEN STATUS
MOVE 1, msg_in_buf, WHEN MSG_IN
INT error_not_cmd_complete, IF NOT 0
CLEAR ACK
WAIT DISCONNECT
MOVE MEMORY 1, IO_DONE_BUF, DONE_YET_BUF
JUMP scheduler
```

- Execute a SCRIPTS INT instruction. This is the simplest method. It causes the SCSI SCRIPTS to stop processing.

Example: `INT io_complete`

- Execute a Memory to Memory Move to a predetermined location. Then execute an INTFLY instruction to indicate to the processor to look at the predetermined location to determine which I/O has completed.

Chapter 11

Programming Multifunction Devices

The SYM53C876 and SYM53C885 are multifunction devices, with SCSI functions that are similar to the SYM53C875. This chapter provides instructions for enabling and using features that are unique to these products.

The SYM53C885 is wide, single-ended SCSI/Fast Ethernet multifunction controller. The SCSI portion of the SYM53C885 is functionally identical to the SYM53C875, with the addition of power management features and programmable PCI bus arbitration priority. The SYM53C876 is a dual-port SCSI controller based on the SYM53C875. For more detailed information on the SYM53C885 and SYM53C876 devices, please refer to the specific product data manual.

Using the SYM53C885 Power Management Feature

The SYM53C885 supports two power management modes: Coma Mode and Snooze Mode. Before it can enter one of these modes, the chip must be in the following condition:

1. No master cycles occurring (No SCSI SCRIPTS are running).
2. No SCSI transactions occurring.
3. In Target mode with select turned off.
4. No pending interrupts.
5. All interrupts are disabled except Wakeup interrupt (WI). This will ensure WI is the only interrupt in the queue and minimize hang-up conditions that may be caused by interrupts generated in the disabled circuitry.

Coma Mode

Coma mode is the lowest-power mode available on the chip. All functions are powered down except for those necessary to cause the chip to exit coma mode. Coma mode in the SCSI function deactivates the following circuits:

- Entire SCSI core, except for register accesses
- SCSI Transceivers

To put the SCSI function into coma mode, set the CM bit in the CTEST0 register. To take the SCSI function out of coma mode, reset the CM bit in the CTEST0 register (PCI slave access is still operational in coma mode).

Snooze Mode

Snooze mode deactivates all circuits except those that detect a predetermined activity on the associated interface. These predetermined activities are: a SCSI bus reset in the SCSI function; or the reception of a Magic Packet in the Ethernet function. When this activity is detected, the chip generates a “wakeup” interrupt. The interrupt service routine should then take the appropriate function out of snooze mode.

To put the SCSI function into snooze mode:

1. Set the WI bit in the SIEN1 register.
2. Set the SM bit in the CTEST0 register

In snooze mode, all data transmission circuitry is inoperative.

Upon reception of a SCSI bus reset, the chip generates an interrupt. The SIP bit in the ISTAT register and the WI bit in the SIEN1 register will be set.

To take the SCSI function out of snooze mode:

1. Reset the SM bit in the CTEST0 register.
2. Reset the WI bit in the SIEN1 register.

Register Bits Used for Power Management

The ISTAT, SIEN1, SIST1 and CTEST0 registers are used for the SCSI implementation of the power management feature. The table below shows the register and corresponding bits used by the power management feature:

Table 11-1
SYM53C885 Power Management Registers

Register	Bit	Bit Name
CTEST0	4	Coma Mode Enable (CM)
CTEST0	3	Snooze Mode Enable (SM)
SIEN1	3	Wakeup Interrupt Enable (WIE)
SIST1		Wakeup Interrupt (WI)
ISTAT	1	SCSI Interrupt pending (SIP)

Programming the SYM53C885 Internal Arbiter

There are three independent bus-mastering functions inside the SYM53C885: the SCSI controller, the Ethernet transmit channel, and the Ethernet receive channel. Each channel has a register which allows programming of a three-bit arbitration priority level. Zero is the lowest priority, and seven is the highest priority.

For example, if the Ethernet receive channel has an arbitration priority level of 3, and the SCSI channel has an arbitration priority level of 1, then the SCSI channel is allowed to transmit or receive one burst of data over the PCI bus for every three bursts of data that the Ethernet receive channel transmits (Enet arbitration level - SCSI arbitration level + 1). If the SCSI channel's arbitration priority level was changed to two, then the SCSI channel would be allowed to transmit or receive one burst of data over the PCI bus for every two bursts of data that the Ethernet receive channel transmits. If arbitration priority levels for all bus mastering functions are set to the same value, then the arbitration algorithm defaults to a round-robin arbitration scheme.

To program the SCSI function's internal arbitration priority, set the SCSI function's three bit priority field (AP2-AP0) to the desired binary arbitration level. The SCSI arbitration priority bits are CTEST0 bits 2-0. To program the Ethernet function's internal arbitration priority, refer to the Symbios Logic E*thernet Programming Guide*.

Chapter 12

Using the SYM53C8XX in Target Applications

Overview

The SYM53C8XX family of PCI-SCSI I/O Processors can run on target as well as host devices. Target operation is very similar to host operation, except that the SYM53C8XX responds to SCSI commands from the host rather than initiating the commands. The basic structure of all target operations is:

1. The SYM53C8XX issues a Wait Select instruction
2. The SCSI bus goes into Message Out phase
3. The SYM53C8XX performs a series of Block Moves corresponding to the next four SCSI bus phases, as illustrated in Table 12-1
4. The SYM53C8XX issues a Disconnect instruction to disconnect the target device from the bus.

Table 12-1
SCSI Protocol and Target SCRIPTS Instructions

Bus Phase	Definition	SCRIPTS instruction
Bus Free	This phase indicates that the SCSI bus is available.	NA
Arbitration	This phase allows the initiator to gain control of the SCSI bus.	NA
Selection	During this phase, the target responds to the initiator's selection.	WAIT SELECT
Message Out	During this phase, the target may receive messages from the initiator, such as queuing and error recovery information.	MOVE WITH MESSAGE OUT

Table 12-1
SCSI Protocol and Target SCRIPTS Instructions (Continued)

Bus Phase	Definition	SCRIPTS instruction
Command	During this phase, the target may receive commands in the form of a command descriptor block (CDB) to the target buffer.	MOVE
Data In/Out	Data In and Data Out phases are used to send data to the initiator or to the target and are used dependent on the information transferred during the Command phase. This phase is optional. For example, a Test Unit Ready command does not require a data transfer.	MOVE
Status	During this phase, the target sends status information to the initiator about the previously executed CDB.	MOVE
Message In	During this phase, the target sends messages to the initiator. These messages can acknowledge or reject previously sent initiator messages. They also can provide other information like queuing, disconnect, or parity errors.	MOVE
Disconnect	This phase is used to end the target device's connection with the bus.	DISCONNECT
Bus Free	After successful completion of an I/O operation and a request for disconnect, the bus returns to the Bus Free state, indicating that it is now available.	DISCONNECT

Registers Used for Target Operation

For target operation, only a few of the operating register values are different from initiator operation. Table 12-2 summarizes the register bit operations that are of particular interest for target operation.

Table 12-2
Register Bits Used for Target Operation

Register Name	Bits	Description
RESPID1, RESPID0	all	Setting multiple bits in these registers allows the SYM53C8XX to respond to multiple SCSI IDs
SCNTL0	0	Set this bit to make the SYM53C8XX a target device by default
SCID	5	Set this bit to allow the SYM53C8XX to respond to bus-initiated selection at the chip ID in the RESPID1-0 registers
SCNTL1	5	When this bit is clear, the SYM53C8XX halts the data transfer when a parity error is detected or when the SATN/ signal is asserted.

Using SCRIPTS for Target Operations

SCRIPTS instructions operate identically in target or initiator mode, except for certain forms that are valid in only one mode; these exceptions are all noted in the individual instruction descriptions in Chapter 3. When the target device is moving data to the SCSI bus and is halted for any reason, the residual data in the FIFO must be cleaned up before the transfer can resume. It is most common to empty the FIFOs, send a Restore Pointers message and start the transfer again.

Most interrupts to the target operation are expected. The floppy disk provided with this programming guide contains a sample interrupt service routine for a target device.

Sample Target Operation SCRIPTS Program

This section uses a sample SCRIPTS program to illustrate programming techniques for the SYM53C8XX chips when operating in target mode. This program is used for testing and development of Symbios Logic products. The full text of the SCRIPTS source file and accompanying code for target operation may be downloaded from the Symbios Logic BBS or from the floppy disk included with this programming guide.

```
8xxtarg.ss   Revision 2.2 2/12/96

;

; This software was written by Symbios Logic Inc. to
; develop and test new products. Symbios Logic assumes
; no liability for its use. This software is released
; to the public domain to illustrate certain
; programming techniques for the SYM53C8xx chips in
; target mode.

;
```

The ABSOLUTE declarations in this program are the types of interrupts that the target will generate. The SYM53C8XX issues interrupts to notify the host of completed actions or to find out what action to take next.

```
;ABSOLUTE DECLARATIONS

ABSOLUTE read_access_medium= 0x00

ABSOLUTE write_access_medium= 0x01

ABSOLUTE last_write_disconnect= 0x02

ABSOLUTE seek_command= 0x03
```

```
ABSOLUTE set_up_synch_neg= 0x04
ABSOLUTE set_up_wide_neg= 0x05
ABSOLUTE non_handled_msg    = 0x06
ABSOLUTE bad_extended_msg= 0x07
ABSOLUTE message_sent           = 0x08
ABSOLUTE request_sense_command= 0x09
ABSOLUTE inquiry_command= 0x0a
ABSOLUTE read_capacity_command= 0x0b
ABSOLUTE start_stop_command= 0x0c
ABSOLUTE format_unit= 0x0d
ABSOLUTE send_diagnostic= 0x0e
ABSOLUTE command_aborted= 0x0f
ABSOLUTE illegal_cmd= 0x10
ABSOLUTE got_SIGP = 0x11
ABSOLUTE done_with_copy= 0x12
ABSOLUTE got_selected= 0x13
ABSOLUTE done_with_busy_command= 0x14
```

The EXTERNs in this program are variables used for Memory-to-Memory Move operations, such as moving SCRIPTS from program memory into RAM or moving data from one memory location to another.

```
EXTERN count
EXTERN source_address
EXTERN destination_address
```

This section defines the table format and layout. Each entry in the table represents a two-dword entry in a data structure. Each entry contains a byte count and an address that points to a buffer that is used for Block Move instructions. The buffer must be declared in the driver code.

> Note: the declared values and sizes are only for the SCRIPTS debugger, NVPCI. The assembler does not use these and the information is not included in the “C” code. The buffers must be set up in the driver program.

```
TABLE table_indirect \
```

```
    msg_out_buf = 1{??}, \
    cmd_buf= 12{??}, \
    synch_neg_msg_out = 2{??}, \
    wide_neg_msg_out = 1{??}, \
    neg_msg_in = {0x01, 0x03, 0x01, 0x19, 0x08}, \
    stat_buf = {0x02}, \
    identify_msg_in_buf = {0x80},\
    msg_in_buf = 1{??}, \
    data_buf = 512{??}, \
    save_pointers = {0x02}, \
    disconnect_msg = {0x02, 0x04}, \
    selector_id = ID{0x33, 0x07, 0x00, 0x00}, \
    sense_data_buf = {0x00,0x00,0x06,0x00, 0x00, 0x00, \
            0x00, 0x0a, 0x00, 0x00, 0x00, 0x00, \
            0x29, 0x00, 0x00, 0x00, 0x00, 0x00}, \
  inquiry_data_buf = {0x00,0x00,0x02,0x00, 0x1f,0x00, \
              0x00, 0x10, 0x20, 0x20, 0x20, 0x20, \
              0x20, 0x20, 0x20, 0x20, 0x20, 0x20, \
              0x20, 0x20, 0x20, 0x20, 0x20, 0x20, \
              0x20, 0x20, 0x20, 0x20, 0x20, 0x20, \
              0x20, 0x20, 0x20, 0x20, 0x20, 0x20}, \
    capacity_data_buf = {0x00, 0x80, 0x02, 0x00}

;*************************************************
```

The ENTRY declarations are starting points in the SCRIPTS program that will be referred to in “C” code. See the output file explanation for more information on how these are assembled and used in the driver code.

```
; ENTRY declarations
ENTRY wait_select
ENTRY msg_out_phase
ENTRY tur
ENTRY stopped_busy_tur
ENTRY request_sense
```

```
ENTRY read_return

ENTRY read_reconnect

ENTRY write_return

ENTRY write_reconnect

ENTRY synch_wide_neg_return

ENTRY msg_in_phase

ENTRY inquiry

ENTRY read_capacity

ENTRY stopped_busy_wait_select

ENTRY copy_data
```

The `wait_select` label is the generic starting point for target operations. The SYM53C8XX waits here until it is selected. It jumps to Command phase if ATN is not set, or performs one of the other commands described in the comments below. It will jump to an alternate label if the SIGP bit is set.

```
wait_select:

   wait select rel(SIGP_set) ;wait to be selected

   jump rel(command_phase), if not atn ;SCSI-1
                                       ;initiator
                                       ;support

   move from msg_out_buf, with msg_out ;get message
                                       ;byte

   move sfbr to scratchb0 ;save the identify message

   call rel(msg_out_phase), if atn ;stay in message if
                                   ;atn still active
```

If the SYM53C8XX is selected without ATN, it goes directly to Command phase to support SCSI-1 initiators. The chip receives the command descriptor block and performs various functions, described in the program comments, depending on the contents of the command.

```
command_phase:

   move from cmd_buf, with cmd ;get SCSI command

   move scntl1 & 0xdf to scntl1 ;turns on the halt on
                                ;parity error or atn

   jump rel(read), if 0x08 ;jump to set up read
                           ;(6-byte read)
```

```
int write_access_medium, if 0x0a ;interrupt to set
                                  ;up write
                                  ;(6-byte write)

int seek_command, if 0x0b ;interrupt to perform seek

int seek_command, if 0x2b ;interrupt to perform seek

jump rel(read), if 0x28 ;jump to set up read
                        ;(10-byte read)

int write_access_medium, if 0x2a ;interrupt to set
                                  ;up write
                                  ;(10-byte write)

jump rel(tur), if 0x00 ;jump to test unit ready

int request_sense_command,if 0x03 ;interrupt to set
                                  ;up request sense
                                  ;command

int inquiry_command, if 0x12 ;interrupt to set up
                             ;inquiry command

int read_capacity_command,if 0x25 ;interrupt to set
                                  ;up read capacity
                                  ;command

int start_stop_command, if 0x1b ;interrupt to set
                                ;up start/stop unit
                                ;command


jump rel(tur), if 0x2f ;verify command, go to tur

jump rel(reserve_unit), if 0x16 ;jump to reserve
                                ;unit

jump rel(release_unit), if 0x17 ;jump to release
                                ;unit

int send_diagnostic, if 0x1d ;interrupt to set up
                             ;send diagnostic

int format_unit, if 0x04 ;interrupt to set up
                         ;format unit

int illegal_cmd ;interrupt on any other command
```

In Message Out phase, the initiator moves in other types of messages, such as wide or synchronous negotiation, or NOPs.

```
msg_out_phase:

   return, if not atn ;return if atn gone

   move from msg_out_buf, with msg_out ;get message
                                       ;byte
```

```
jump rel(extended_msg), if 0x01 ;jump if extended
                                 ;message

jump rel(abort), if 0x06 ;jump if abort message

jump rel(msg_out_phase), if 0x08;jump back if nop
                                 ; message

int non_handled_msg ;interrupt if can't handle
                    ; message
```

If the chip receives a byte in the message phase indicating an extended message, then it jumps to these commands.

```
extended_msg:

   int bad_extended_msg, if not atn;if atn gone,
                                    ;extended message
                                    ;was bad

   move from msg_out_buf, with msg_out;get next
                                       ;message byte

   int bad_extended_msg, if not atn ;if atn gone,
                                    ;extended message
                                    ;was bad

   move from msg_out_buf, with msg_out;this byte shows
                                       ;type of message

   jump rel(synch_neg), if 0x01 ;0x01 is a synchronous
                                ;negotiation message

   jump rel(wide_neg), if 0x03 ;0x03 is a wide
                               ;negotiation message

   int bad_extended_msg ;interrupt on any other type
```

In these commands, the SYM53C8XX moves synchronous period and offset data from the synchronous negotiation message and interrupts to set up the synchronous operation and return message.

```
synch_neg:

   move from synch_neg_msg_out, with msg_out;move in
                                            ;the period
                                            ;and offset

   int set_up_synch_neg ;interrupt to set up
                        ;synchronous and message
```

If the extended message is for wide negotiation, the SYM53C8XX expects one more byte with SCSI bus width information, then interrupts to set up the answer

```
wide_neg:
```

```
   move from wide_neg_msg_out, with msg_out ;move in
                                             ;the width

   int set_up_wide_neg ;interrupt to set up answer
```

After the interrupt service routine executes, the SYM53C8XX sends its return negotiation message.

```
synch_wide_neg_return:

   move from neg_msg_in, with msg_in ;move out our
                                     ;answer to the
                                     ;negotiation

   jump rel(command_phase), if not atn;jump to
                                       ;command_phase
                                       ;if atn gone

   jump rel(msg_out_phase);jump to msg_out_phse if atn
                          ;still active
```

If the target device goes into Message In phase, an exception condition has occurred that requires the target device to send some kind of recovery message to the initiator. With these commands, the SYM53C8XX sends the message and interrupts to determine what to do next. Some of the messages that the target might need to send are Message Reject, Restore Data Pointers, or some other error message.

```
msg_in_phase:

   move from msg_in_buf, with msg_in ;move a message
                                    ; to the initiator

   int message_sent ;interrupt to determine what to
                    ;do next
```

This is the final sequence of commands for any I/O. The SYM53C8XX sends a status message, disconnects from the SCSI bus, executes an interrupt on the fly, and goes back to the `wait_select` label to get ready for next command

```
tur: ; (Test Unit Ready)

   move from stat_buf, with status ;send out status
                                   ;byte

   move from msg_in_buf, with msg_in ;send out message
                                     ;byte

   move 0x20 to scntl1 ;turns off the halt on parity
                       ;error or atn

   disconnect ;disconnect from the SCSI bus
```

```
   intfly ;interrupt to signal end of process

   jump rel(wait_select)
```

This series of commands is the same as the Test Unit Ready label , but the SYM53C8XX has been selected while processing another command or has been issued a Stop command. The device stops and sends back a Status of busy if the command is one the target device does not have to accept when busy or stopped.

```
stopped_busy_tur:

   move from stat_buf, with status ;send out status
                                   ;byte

   move from msg_in_buf, with msg_in ;send out message
                                     ;byte

   move 0x20 to scntl1 ;turns off the halt on parity
                       ;error or atn

   disconnect ;disconnect from the SCSI bus

   move scratcha1 to sfbr ;get the busy flag

   int done_with_busy_command, if 0x01 ;if busy,
                                       ;interrupt to
                                       ;continue

   intfly ;interrupt to signal end of process

   jump rel(stopped_busy_wait_select)
```

These labels send the sense, inquiry, or capacity data requested by the initiator. The SYM53C8XX moves the data and checks to see which Test Unit Ready command to use next

```
request_sense:

   move from sense_data_buf, with data_in ;move the
                                          ;sense data
                                          ;from the
                                          ;buffer

   move scratcha2 to sfbr ;get the stopped/busy flag

   jump rel(tur) if 0x00 ;go to the appropriate status\
                         ;and message phases

   jump rel(stopped_busy_tur)

inquiry:

   move from inquiry_data_buf, with data_in ;move out
                                           ;inquiry data

   move scratcha2 to sfbr ;get the stopped/busy flag
```

```
   jump rel(tur) if 0x00 ;go to the appropriate status
                         ;and message phases

   jump rel(stopped_busy_tur)

read_capacity:

   move from capacity_data_buf, with data_in;move out
                                             ;read
                                          ;capacity data

   move scratcha2 to sfbr ;get the stopped/busy flag

   jump rel(tur) if 0x00 ;go to the appropriate status
                           ;and message phases

   jump rel(stopped_busy_tur)
```

The read label is the starting point for all read commands. If disconnects are allowed, the chip jumps to the read_disconnect label. Read return is used after read information is set up in the data buffer. A series of commands determine if the transfer is finished. If so, then the SYM53C8XX goes to Test Unit Ready or tries to disconnect again.

```
read:

   move scratchb0 to sfbr ;get identify message

   jump rel(read_disconnect) if 0x40 and mask 0xbf
                           ;jump disconnect if
                            ;disconnects are allowed

   int read_access_medium ;interrupt to read data
                           ;from medium


read_return:

   move from data_buf, with data_in;move the data out
                                    ;from the buffer

   move scratchb1 to sfbr ;get the 'finished' flag

   jump rel(tur) if 0x00 ;jump to status and message
                           ;if transfer done

   move scratchb0 to sfbr ;get identify message

   jump rel(read_disconnect) if 0x40 and mask 0xbf
                              ;jump to disconnect if
                              ;disconnects are allowed

   move from save_pointers, with msg_in ;move out the
                                   ;save pointers message

   call rel(msg_out_phase) if atn ;jump to message out
                                    ;if atn active
```

```
   int read_access_medium ;interrupt to access medium
```

The read_disconnect label disconnects the device from the bus, and sets the Semaphore bit to tell the ISR it is disconnected.

```
read_disconnect:

   move from disconnect_msg, with msg_in ;move out the
                                         ;disconnect message

   call rel(msg_out_phase) if atn ;jump to message out
                                  ;if atn active

   move 0x20 to scntl1 ;turns off the halt on parity
                       ;error or atn

   disconnect ;disconnect from the bus

   move 0x10 to istat ;set the semaphore bit to say we
                      ;are disconnected

   int read_access_medium ;interrupt to read data from
                          ;medium
```

The read_reconnect label performs reselection, moves the identify message from the message in buffer, and jumps to send the data.

```
read_reconnect:

   reselect from selector_id, rel(alt_got_selected)
                              ;reselect the initiator

   move scntl1 & 0xdf to scntl1 ;turns on the halt on
                                parity error or atn

   move from identify_msg_in_buf, with msg_in ;move in
                                      identify message

   jump rel(read_return) ;jump to send data
```

On a write, the SYM53C8XX interrupts immediately to set up counts for moving data. It takes data from the initiator, then begins the write. When writing is done, control jumps to the Test Unit Ready label.

```
write_return:

   move from data_buf, with data_out ;move the data
                                     ;into the buffer

   move scratchb0 to sfbr ;get identify message

   jump rel(write_disconnect) if 0x40 and mask 0xbf
                              ;jump to disconnect if
                              ;disconnects allowed
```

```
   move scratchb1 to sfbr ;get the 'finished' flag

   jump rel(tur) if 0x10 ;jump to status and
                         ;message if transfer done

   move from save_pointers, with msg_in ;move out the
                                        ;save pointers
                                        ;message

   call rel(msg_out_phase) if atn ;jump to message out
                                  ;if atn active

   int write_access_medium;interrupt to read data
                          ;from medium
```

The `write_disconnect` label does all same things as the `read_disconnect`. It sets the semaphore bit and issues one of two interrupts, depending on whether or not this is the last write of the transfer.

```
write_disconnect:

   move from disconnect_msg, with msg_in ;move out the
                                         ;disconnect message

   call rel(msg_out_phase) if atn ;jump to message out
                                  ;if atn active

   move 0x20 to scntl1 ;turns off the halt on parity
                       ;error or atn

   disconnect ;disconnect from the bus

   move 0x10 to istat ;set the semaphore bit to say
                      ;we are disconnected

   move scratchb1 to sfbr ;get the 'finished' flag

   int last_write_disconnect if 0x10;special interrupt
                                    ;after last data
                                    ;phase

   int write_access_medium;interrupt to read data
                          ;from medium
```

The `write_reconnect` label operates the same as `read_reconnect`.

```
write_reconnect:

   reselect from selector_id, rel(alt_got_selected)
                          ;reselect the initiator

   move scntl1 & 0xdf to scntl1 ;turns on the halt on
                                ;parity error or atn
```

```
move from identify_msg_in_buf, with msg_in ;move in
                                          ;identify message

move scratchb1 to sfbr ;get the 'finished' flag

jump rel(tur) if 0x00 ;jump to status and message
                      ;if transfer done

jump rel(write_return) ;jump to get data
```

The `reserve_unit` label sets a reservation flag, gets the ID of the initiator who sent the command, and jumps to Test Unit Ready to complete command.

```
reserve_unit:

   move 0x01 to scratchb2 ;set 'reserved' in
                          ;reservation flag

   move ssid & 0x7f to sfbr ;get the ID of who
                            ;reserved us

   move sfbr to scratchb3 ;move ID into storage buffer

   jump rel(tur) ;go to status and message
```

The `release_unit` command clears the reserved flag and goes to Test Unit Ready.

```
release_unit:

   move 0x00 to scratchb2 ;set 'not reserved' in
                          ;reservation flag

   jump rel(tur) ;go to status and message
```

The `abort` label turns off the halt on parity or ATN bit, and disconnects from the bus. The chip executes this command when it receives an Abort message for the command in process. The interrupt service routine then cleans up the job.

```
abort:

   move 0x20 to scntl1 ;turns off the halt on parity
                       ;error or atn

   disconnect ;go to bus free

   int command_aborted ;int to notify driver that
                       ;command was aborted
```

The SYM53C8XX only performs this routine if it is selected while it is stopped or busy working on another command. The Request Sense, Test Unit Ready, Inquiry, and Read Capacity commands are valid while the target device is busy or stopped and the chip must respond to them. If the chip is stopped, it will only respond to one of these commands or the Start command.

```
stopped_busy_wait_select:

   wait select rel(SIGP_set) ;wait to be selected

   move from msg_out_buf, with msg_out ;get message
                                       ;byte

   call rel(msg_out_phase), if atn ;stay in message\
                                   ;if atn still active

   move from cmd_buf, with cmd ;get SCSI command

   move scntl1 & 0xdf to scntl1 ;turns on the halt on
                                ;parity error or atn


   jump rel(stopped_busy_tur), if 0x00 ;jump to test
                                       ;unit ready

   int request_sense_command, if 0x03;interrupt to set
                              ;up request sense command

   int inquiry_command, if 0x12 ;interrupt to set up
                                ;inquiry command

   int read_capacity_command, if 0x25 ;interrupt to
                                      ;set up read
                                   ;capacity command


   move sfbr to scratcha3 ;save the first byte of the
                          ;command

   move scratcha1 to sfbr ;get the busy flag

   jump rel(stopped_busy_tur), if 0x01 ;if busy, go
                                       ;right to status
                                       ;and message


   move scratcha3 to sfbr ;restore the first byte of
                          ;the command

   int start_stop_command, if 0x1b ;interrupt to set
                                    ;up start/stop unit
                                    ;command
```

```
   jump rel(stopped_busy_tur) ;go to status and
                               ;message for any other
                               command


alt_got_selected:

   int got_selected ;interrupt because got selected
                     ;during reselect attempt


SIGP_set:

   int got_SIGP ; taking interrupt because got SIGP


copy_data:

   move memory count, source_address,
destination_address ;memory move to write
                     ;SCRIPTS RAM and to
                    ;transfer data to and
                    ;from upper memory

   int done_with_copy ;signal completion of memory move
```

Synchronous Negotiation by a Target Device

For target operation, negotiating occurs when a synchronous negotiation message is received from the initiator. Once this message is received, a SCRIPTS Interrupt instruction would be executed to determine the necessary response. Once the synchronous parameters for a particular initiator have been established, they should be saved in a table for later reconnects to the same device. If reselecting an initiator, the RESELECT FROM command can be used to indicate table indirect addressing. The SXFER, SCNTL3, and SDID register values would then be loaded from the table entry. When selected by an initiator that has previously negotiated for synchronous transfers, these registers would need to be reloaded from memory before the target goes to the data transfer phase.

Chapter 13

Debugging the SYM53C8XX

Overview

The SCRIPTS registers and the SCSI registers contain information that may be helpful in debugging the chip. Table 13-1 shows the information contained in the registers:

Table 13-1
Registers Useful for Debugging SYM53C8XX

Information	Register	Remarks
Information regarding the most recent interrupt	ISTAT	Check this register first, since its contents may be affected by reading or writing other registers.
Current SCRIPTS instruction	DCMD and DBC (first 32 bits); DNAD or DSPS (second 32 bits)	The DCMD and DBC always contain the op code of the most recently executed SCRIPTS instruction. Use the cross reference file created from the SCRIPTS source by NASM to interpret the contents. The DSPS or DNAD contains the second 32-bit field of the SCRIPTS instruction fetched.
Next SCRIPTS Instruction address	DSP	Contains the address of the next instruction to be fetched. This is analogous to the program counter of a microprocessor. Instruction addresses are on 8-byte boundaries (except Memory Move, which is on a 12-byte boundary) and so the value in the DSP should be eight past the address of the current instruction.
SCSI Bus Control Lines	SBCL	Contains the current state of SCSI control lines.
SCSI Bus Data Lines	SBDL	Contains the current status of SCSI data lines.

Table 13-1
Registers Useful for Debugging
SYM53C8XX (Continued)

Information	Register	Remarks
Last SCSI Phase serviced	SOCL	Contains the phase to match (initiator) or the phase driven (target) from the last SCRIPTS instruction executed.
Last SCSI data byte sent	SODL	Contains the last byte transferred to the SCSI bus.
Last SCSI data byte received	SIDL	Contains the last byte transferred in from the SCSI bus.
First byte received from Block Move instruction executed	SFBR	Contains the first byte of a block move transferred in from SCSI. It also contains SCSI identities after a reselection, if using 53C700 compatibility mode and if the IDs are in the 0-7 range.
53C8XX SCSI ID	SCID	Contains the SCSI ID of the SYM53C8XX chip.
Destination SCSI ID	SDID	Contains the identity of the target for the last select or reselect instruction executed.
Response ID	RESPID0, RESPID1 (wide SCSI devices only)	Contains the IDs that the chip responds to on the SCSI bus. The chip can respond to multiple IDs , so more than one bit can be set in these registers.

Chip Debugging Guidelines

1

a Check the register initialization routine. Several registers should be checked in this step. The most important registers to verify are listed in Chapter 6.

b Save and print out the data values in all SYM53C8XX registers at the time the problem occurs. Record the value of the ISTAT register first, since further register accesses may trigger interrupts that were not caused by the initial problem. If there is not an interrupt, abort the SCRIPTS operation by writing to the ABRT bit in the ISTAT register. This will cause a DMA abort interrupt. Reset this bit before reading the DSTAT register to prevent further interrupts from being generated. Clear the interrupt(s) following the method suggested in Chapter 6.

Once the interrupts have been cleared, the registers listed in Table 13-1 contain most of the critical information. If there is no indication of what is causing the problem, it might be helpful to look at the rest of the registers.

2 Use the DSP, DSPS, DCMD, and DBC registers to determine where SCRIPTS execution was stopped. The `.LIS` file generated by NASM using the `-l` option can be very helpful in this step. Compare the listings to the debugging register values to determine what might be causing the problem.

3 If the problem has not yet been discovered, examine logic analyzer traces of both the host bus and the SCSI bus to verify that SCRIPTS fetches are occurring correctly. They can also be used to compare data transferred between the two interfaces.

4 Perform timing verification using a logic analyzer. Signal quality issues and clock problems may require the use of an oscilloscope.

After this information has been gathered and examined, if no problem has been revealed, this information along with your code can enable an Symbios Logic system engineer to assist with your debugging efforts.

Common Problems/ Things to Check

1. The CPU is accessing registers other than ISTAT while SCRIPTS are running. ISTAT is the only register that can be accessed during SCRIPTS operation.

2. The RESPID register(s) are not initialized. This would keep the chip from responding to any selection/reselection. Make sure these registers are initialized correctly.

3. Verify signal connectivity. (Make sure that the chip pins are all connected to board traces.) Verify power and ground connection to the chip. Verify that decoupling capacitors are connected as recommended in the chip data manual to avoid noise problems.

4. Make sure that the Enable Response to Selection/Reselection bits are set correctly.

Glossary

Address A specific location in memory, designated either numerically or by a symbolic name.

Address Range A contiguous block of memory, designated by a starting address and an ending address.

Common Command Block (CCB) Contains the information required by the hardware interface of the device driver for a specific request.

Declarative Keywords Words in the SCRIPTS programming language used to control the different aspects of code generation

Patching Modifying some elements of the SCRIPTS array after buffers have been allocated.

PCI (Peripheral Component Interconnect) A high-performance interface for personal computers and workstations.

Label A symbol representing a specific location in the section of memory used for code.

Loopback Mode A diagnostic mode that allows the SYM53C8XX to control all signals, to test both initiator and target operations of the chip.

NASM A DOS command line assembler that supports Symbios Logic SCSI processors.

SCSI Small Computer System Interface

SCSI SCRIPTS A high level instruction set for programming the SYM53C8XX family of PCI-SCSI I/O Processors.

Symbol An identifier used to represent a location in memory. The identifier may be any combination of alphanumeric characters allowed by the lexical rules of the programming language being used.

Appendix A

NASM Error Messages

Errors

24 bit value expected

The value specified is not within the range of a 24-bit unsigned integer. The value must be between 0 and 4M.

Something other than a value was found.

8 bit value expected

The value specified is not within the range of a 8-bit unsigned integer. The value must be between 0 and 255.

Something other than a value was found.

ACK, ATN,TARGET or CARRY expected string

String was found instead of ACK, ATN, TARGET or CARRY.

AND or OR expected string

String was found instead of AND or OR.

ATN specified multiple times

The ATN field may only be specified once per instruction

Cannot compare CARRY and Data

The command is requesting that both a comparison of the SFBR register to the specified data and a test of the carry bit take place, but only one test is allowed.

Cannot compare PHASE and Data

The command is requesting that both a comparison of the SCSI bus phase and a comparison of the SFBR register to the specified data take place, but only one test is allowed.

Cannot specify PHASE when using ATN

The use of PHASE and ATN are mutually exclusive.

Cannot use MASK without compare Data

Valid Data must be present when using the MASK field.

Cannot use Pass for count address

The PASS feature cannot be used in a count field. The current format of the output file does not support this.

Carry operations not available on 53c700 architectures

The Carry feature is only available on the 53C710 or higher architectures.

CARRY specified multiple times

CARRY may only be specified once per instruction.

CHMOV 53c720, 53c770, 53c82X, 53c875, 53c876, 53c885, and 53c895 architectures only

The CHMOV instruction is only available on the chips that support wide SCSI.

Comma expected string

String was found instead of a comma.

CTEST7 53c700 and 53c710 architectures only

The CTEST7 register is only available on the 53C700/710 architectures.

CTEST8 53c700 and 53c710 architectures only

The CTEST8 register is only available on the 53C700/710 architectures.

Data list expected string

String was found instead of a list of initialized data.

Data specified multiple times

The Data field may only be specified once for a given instruction.

Data specifier expected string

String was found instead of a Data specifier. A Data specifier is used to specify the size of a data area and to initialize that data area.

Declaration expected string

String was found when a declaration was expected. A declaration is an assignment of a variable to some value or data specifier.

Divide or mod by zero

DSAREL: 53c810A, 53c825A, 53C860, 53c875, and 53c895 architectures only

The DSAREL keyword is only supported by the chips that support Load and Store instructions.

Entry identifier expected name

Name was found instead of an identifier. An identifier is a symbol that has not previously been declared.

Expression must evaluate to a constant string

A label, relative, external, or an undeclared identifier, string, does not evaluate to a known value. The value must be known at assembly time.

Expression or External expected

GPCNTL 53C720, 53C770, and 53C8XX only

The GPCNTL register is available only on the 53C720 and higher architectures

GPREG 53c720,53C770, and 53C8XX architectures only

The GPREG register is only available on the 53C720 and higher architecture

ID specifier only valid for table entries

IF or WHEN expected string

String was found instead of one of IF or WHEN.

INTFLY:53c720, 53C770, and 53C8XX architecture only

The INTFLY instruction is only available on the 53C720 and higher architectures.

Invalid Address string

String was found instead of a valid address. A valid address is an expression, external, relative, table, or an absolute.

Invalid assignment

Invalid character/s

Invalid constant type

Invalid destination address string

String was found instead of a valid destination address. A valid destination address is an expression, external, relative, table or an absolute.

Invalid register operator string

String was instead of a valid operator. Valid operators are '+', '-', '|', '&'.

Invalid register value

Value must be in the range 0-3Fh for the 53c700/710 and 0-5Ch for the 53c720.

Invalid SCSI id

Value must have only one bit set (bits 0-7) for the 53C700/710/810 and must be in the range of 0-15 for the 53c720/820/825.

Invalid syntax string

String was found and not expected causing an unknown syntax error.

Invalid test condition string

String was found instead of a valid test condition. The valid test conditions are CARRY, a PHASE, an 8 bit value, or a MASK.

LCRC 53c710 architectures only

The LCRC register is only available on the 53C710 architecture

Left parenthesis expected string

String was found instead of a left parentheses.

LOAD: 53c810A, 53c825A, 53c860, 53c875, 53c876, 53c885, and 53c895 architectures

The LOAD instruction is only supported by the 53C810A and higher architectures

LOAD: Count must not exceed 4 bytes

Four bytes is the maximum byte count to LOAD.

Logical end of line '\' expected string

A logical line separator is needed before continuing the directive on a new line.

MACNTL 53c720, 53c770, and 53c8XX architectures only

The MACNTL register is available only on the 53C720 and higher architectures

MASK specified multiple times

MASK may only be specified once per instruction.

Memory Move operations not available on 53c700 architectures

The Memory Move instruction is only available on the 53C710 and higher architectures.

Memory Move Noflush only available on 53c810A, 53c825A, 53c860, 53c875, 53c876, 53c885, and 53c895 architectures.

The No Flush option is only available in the 53c810A and higher architectures.

Old EXTERNAL directive, use new EXTERNAL directive string

When the Debug switch is on, the operand string must be declared with the new EXTERNAL directive syntax. The new syntax informs the debugger of the size of the external variable.

Old RELATIVE directive, use new RELATIVE directive string

When the Debug switch is on, the operand string must be declared with the new RELATIVE directive syntax. The new syntax informs the debugger of the size of the relative data area.

One register must be SFBR or both the same.

Register move instruction requires either the source or destination register be the SFBR register, or both the source and destination be the same register.

Only use CARRY with Addition or Subtraction.

The CARRY bit can only be checked when either an addition or subtraction operation is used.

Operand must be a TABLE entry string

When the Debug switch is on, the operand where string resides must be of type TABLE entry. This is used for table indirect addressing and to inform the debugger about the size of the table.

Parenthesis must match when PASS is used as an argument

When a PASS variable is used as an argument the parentheses must match.

PHASE expected string

String was found instead of a PHASE.

PHASE specified multiple times

Redeclaration of Label string

The string has previously been declared as a label or some other type of identifier other than an ENTRY.

Redeclaration of TABLE identifier

The string has previously been declared as a TABLE name or some other type of identifier. Only one TABLE declaration per source file is allowed.

Register or Data24 value expected string

String was found instead of a register or a 24 bit value.

Register right of operand must be SFBR

In a Move to SFBR operation, SFBR must be to the right of the operand.

Relative addressing not available on 53c700 architecture

Relative addressing is not supported by the 53c700 architecture.

RESPID 53c81X architecture only

The RESPID register is only one byte in the 53C810.

RESPID0 53c720, 53c770, 53c82X, 53C875, 53c876, 53c885, and 53c895 architectures only

The RESPID0 register is only available in devices that support Wide SCSI.

RESPID1 53c720, 53c770, 53c82X, 53C875, 53c876, 53c885, and 53c895 architectures only

The RESPID1 register is only available in devices that support Wide SCSI.

Right parenthesis expected string

String was found instead of a right parentheses.

SBDL 53c700, 53c710, and 53c81X architectures only

The SBDL register is only one byte in the 53C700, 53C710, and 53C81X architectures.

SBDL0 53c720, 53c770, 53c82X, 53C875, 53c876, 53c885, and 53C895 architectures only

The SBDL register is two bytes in the devices that support Wide SCSI.

SBDL1 53c720, 53c770, 53c82X, 53C875, 53c876, 53c885, and 53C895 architectures only

The SBDL register is two bytes in the devices that support WIde SCSI.

SCNTL2 53c720,53c770, and 53c8XX architectures only.

The SCNTL2 register is only available on the 53C720 and higher architectures.

SCNTL3 53c720, 53c770, and 53c8XX architectures only

The SCNTL3 register is only available on the 53C720 and higher architectures.

Scratch0 53c710 architectures only

The SCRATCH0 register is only available on the 53C710 architecture.

Scratch1 53c710 architectures only

The SCRATCH1 register is only available on the 53C710 architecture.

Scratch2 53c710 architectures only.

The SCRATCH2 register is only available on the 53C710 architecture.

Scratch3 53c710 architectures only

The SCRATCH3 register is only available on the 53C710 architecture.

Scratcha0 53c720, 53c770, and 53c8XX architectures only

The SCRATCHA0 register is only available on the 53C720 and higher architectures.

Scratcha1 53c720, 53c770, and 53c8XX architectures only

The SCRATCHA1 register is only available on the 53C720 and higher architectures.

Scratcha253c720, 53c770, and 53c8XX architectures only

The SCRATCHA2 register is only available on the 53C720 and higher architectures.

Scratcha3 53c720, 53c770, and 53c8XX architectures only

The SCRATCHA3 register is only available on the 53C720 and higher architectures.

Scratchb0 53c720, 53c770, and 53c8XX architectures only

The SCRATCHB0 register is only available on the 53C720 and higher architectures.

Scratchb1 53c720, 53c770, and 53c8XX architectures only

The SCRATCHB1 register is only available on the 53C720 and higher architectures.

Scratchb2 53c720, 53c770, and 53c8XX architectures only

The SCRATCHB2 register is only available on the 53C720 and higher architectures.

Scratchb3 53c720, 53c770, and 53c8XX architectures only

The SCRATCHB3 register is only available on the 53C720 and higher architectures.

Scratchc0 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchc1 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchc2 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchc3 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchd0 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchd1 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchd2 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchd3 53c770, 53c825A, 53C875,53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratche0 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratche1 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratche2 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratche3 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchf0 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchf1 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchf2 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchf3 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchg0 53c770, 53c825A, 53C875,53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchg1 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchg2 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchg3 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchh0 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchh1 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchh2 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchh3 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchi0 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchi1 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchi2 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchi3 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchj0 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchj1 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchj2 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

Scratchj3 53c770, 53c825A, 53C875, 53c876, 53c885, and 53c895 architectures only

The SCRATCHC-J registers are only available on the 53C770, 53C825A, 53C875, and 53C895 architectures.

SELID0: 53c720, 535c770, and 53c8XX architectures only

SELID1: 53c720, 53c770, and 53c8XX architectures only

Separator expected ',' or '\\'

A comma or a logical line separator is needed to delimit declarations.

SIDL 53c700, 53C710, and 53c81X architectures only

The SIDL register is only one byte on the 53C700, 53C710, and 53C81X chips.

SIDL0 53c720, 53c770, 53C82X, 53c875, 53c876, 53c885, and 53c895 architectures only

The SIDL register is two bytes on the chips that support Wide SCSI.

SIDL1 53c720, 53c770, 53c82X, 53C875, 53c876, 53c885, and 53C895 architectures only

The SIDL register is two bytes on the chips that support Wide SCSI.

SHL 53c720, 53c770, and 53c8XX architectures only

The shift left instruction is only supported on 53C720 and higher architectures.

SHR 53c720, 53c770, and 53c8XX architectures only

The shift right instruction is only supported on 53C720 and higher architectures.

SIEN 53c700 and 53c710 architectures only

The SIEN register is only available on the 53C700/710 architectures.

SIEN0 53c720, 53c770, and 53c8XX architectures only

The SIEN0 register is only available on the 53C720 and higher architectures.

SIEN1 53c720, 53c770, and 53c8XX architectures only

The SIEN1 register is only available on the 53C720 and higher architectures.

SIST0 53c720, 53c770, and 53c8XX architectures only

The SIST0 register is only available on the 53C720 and higher architectures.

SIST1 53c720, 53c770, and 53c8XX architectures only

The SIST1 register is only available on the 53C720 and higher architectures.

SLPAR 53c720, 53c770, and 53c8XX architectures only

The SLPAR register is only available on the 53C720 and higher architectures.

SODL 53c700, 53c710, and 53C81X architectures only

The SODL register is one byte only on the 53C700, 53C710, and 53C81X chips.

SODL0 53c720, 53c770, 53c82X, 53c875, 53c876, 53c885, and 53c895 architectures only

The SODL register is two bytes on the chips that support Wide SCSI.

SODL1 53c720, 53c770, 53c82X, 53c875, 53c876, 53c885, and 53C895 architectures only

The SODL register is two bytes on the chips that support Wide SCSI.

SSID 53c720, 53c770, and 53c8XX architectures only

The SSID register is only available on the 53C720 and higher architectures.

STEST0 53c720, 53c770, and 53c8XX architectures only

The STEST0 register is only available on the 53C720 and higher architectures.

STEST1 53c720, 53c770, and 53c8XX architectures only

The STEST1 register is only available on the 53C720 and higher architectures.

STEST2 53c720, 53c770, and 53c8XX architectures only

The STEST2 register is only available on the 53C720 and higher architectures.

STEST3 53c720, 53c770, and 53c8XXarchitectures only

The STEST3 register is only available on the 53C720 and higher architectures.

STEST4 53c895 architecture only

The STEST4 register is only available on the53C895.

STIME0 53c720, 53c770, and 53c8XX architectures only

The STIME0 register is only available on the 53C720 and higher architecture.

STIME1 53c720, 53c770, and 53c8XX architectures only

The STIME1 register is only available on the 53C720 and higher architectures.

STORE: 53c810A, 53c825A, 53c860, 53c875, 53c876, 53c885, and 53c895 architectures

The STORE instruction is only supported by the 53C810A and higher architectures

STORE: Count must not exceed 4 bytes

Four bytes is the maximum byte count to STORE.

SWIDE 53c720, 53c82X, 53c875, 53c876, 53c885, and 53c895 architectures only

The SWIDE register is only available on the Symbios Logic SCSI processors that support wide SCSI.

TABLE directive not available on 53c700 architecture

Table indirect operations are not supported by the 53C700.

Table indirect operations not available on 53c700 architecture

Table indirect addressing is not supported by the 53c700 architecture.

Table name expected string

The directive TABLE was found without a table name declaration.

Unexpected EOF

End of file was found when not expected.

Unresolved Label or Identifier string

String was used but never declared as a label, external, relative, absolute or table.

WITH or WHEN expected

XOR 53c720, 53c810, and 53c825 only

XOR operations are only supported on 53C720 and higher architectures

Fatal Errors

Fatal Error allocating input file buffer(s)

Fatal File Not found

The file named filename was not found in the path specified.

Fatal Memory allocation error

Not enough dynamic memory available to complete assembly of the file. Try dividing up file or freeing memory.

Fatal No source file specified.

A source file to assemble must be specified on the command line. Try specifying source files first before options.

Fatal Opening file

The filename specified can not be opened for some unknown reason.

Fatal read permission denied for file

The filename specified can not be opened with read access.

Warnings

ACK specified multiple times.

The ACK bit can only be specified once per instruction.

ATN specified multiple times.

The ATN bit can only be specified once per instruction.

Cannot extract pass information correctly

The pass variable is poorly formatted and may not have been correctly interpreted.

CARRY specified multiple times.

The CARRY bit can only be specified once per instruction.

Initializer value truncated to byte value

Initialization of data by byte offset only.

Debug record contains old format EXTERNAL statement, data size unknown

Use the new style EXTERNAL directive where data specifiers are used.

Debug record contains old format RELATIVE statement, Data size unknown

Use the new style RELATIVE directive where data specifiers are used.

Initializer value truncated to byte

Possible truncation of constant value

The value of the constant may have been truncated. This is caused by the ASCII conversion of the value.

Relative offset value truncated

Source and.bin file have same the name

The binary file has the same name as the source. The binary file will be renamed or not created.

Source and Error file have same the name

The error file has the same name as the source. The error file will be renamed or not created.

Source and listing file have the same name

The listing file has the same name as the source. The listing file will be renamed or not created.

Source and Object file the same name

The object file and source file have the same name. The object file will be renamed or not created

Source and Out file have the same name

The output file and source file have the same name. The output file will be renamed or not created.

TARGET specified multiple times.

The TARGET bit can only be specified once per instruction

Appendix B

Register Summaries

SYM53C810A Operating Registers

Register 00 (80)
SCSI Control Zero (SCNTL0)
Read/Write

ARB1	ARB0	START	WATN	EPC	RES	AAP	TRG
7	6	5	4	3	2	1	0
Default>>>							
1	1	0	0	0	X	0	0

Bit 7 **ARB1 (Arbitration mode bit 1)**
Bit 6 **ARB0 (Arbitration mode bit 0)**
Bit 5 **START (Start sequence)**
Bit 4 **WATN (Select with SATN/ on a start sequence)**
Bit 3 **EPC (Enable parity checking)**
Bit 2 **Reserved**
Bit 1 **AAP (Assert SATN/ on parity error)**
Bit 0 **TRG (Target role)**

Register 01 (81)
SCSI Control One (SCNTL1)
Read/Write

EXC	ADB	DHP	CON	RST	AESP	IARB	SST
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 **EXC (Extra clock cycle of data setup)**
Bit 6 **ADB (Assert SCSI data bus)**
Bit 5 **DHP (Disable Halt on Parity Error or ATN) (Target Only)**
Bit 4 **CON (Connected)**
Bit 3 **RST (Assert SCSI RST/ signal)**
Bit 2 **AESP (Assert even SCSI parity (force bad parity))**
Bit 1 **IARB (Immediate Arbitration)**
Bit 0 **SST (Start SCSI Transfer)**

Register 02 (82)
SCSI Control Two (SCNTL2)
Read/Write

SDU	RES	RES	RES	RES	RES	RES	RES
7	6	5	4	3	2	1	0
Default>>>							
0	X	X	X	X	X	X	X

Bit 7 **SDU (SCSI Disconnect Unexpected)**
Bits 6-0 **Reserved**

Register 03 (83)
SCSI Control Three (SCNTL3)
Read/Write

RES	SCF2	SCF1	SCF0	RES	CCF2	CCF1	CCF0
7	6	5	4	3	2	1	0
Default>>>							
X	0	0	0	X	0	0	0

Bit 7 **Reserved**
Bits 6-4 **SCF2-0 (Synchronous Clock**
Bit 3 **Reserved**
Bits 2-0 **CCF2-0 (Clock Conversion Factor)**

Register 04 (84)
SCSI Chip ID (SCID)
Read/Write

RES	RRE	SRE	RES	RES	ENC2	ENC1	ENC0
7	6	5	4	3	2	1	0
Default>>>							
X	0	0	X	X	0	0	0

Bit 7 **Reserved**
Bit 6 **RRE (Enable Response to Reselection)**
Bit 5 **SRE (Enable Response to Selection)**
Bit 4-3 **Reserved**
Bits 2-0 **Encoded Chip SCSI ID, bits 2-0**

Register 05 (85)
SCSI Transfer (SXFER)
Read/Write

TP2	TP1	TP0	RES	MO3	MO2	MO1	MO0
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	X	0	0	0	0

Bits 7-5 TP2-0 (SCSI Synchronous Transfer Period)
Bit 4 Reserved
Bits 3-0 MO3-MO0 (Max SCSI synchronous offset)

Register 06 (86)
SCSI Destination ID (SDID)
Read/Write

RES	RES	RES	RES	RES	ENC2	ENC1	ENC0
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	X	0	0	0

Bits 7-3 Reserved
Bits 2-0 Encoded destination SCSI ID

Register 07 (87)
General Purpose (GPREG)
Read/Write

RES	RES	RES	RES	RES	RES	GPIO1	GPIO0
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	X	X	0	0

Bits 7-2 Reserved
Bits 1-0 GPIO1-GPIO0 (General Purpose)

Register 08 (88)
SCSI First Byte Received (SFBR)
Read/Write

1B7	1B6	1B5	1B4	1B3	1B2	1B1	1B0
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Register 09 (89)
SCSI Output Control Latch (SOCL)
Read /Write

REQ	ACK	BSY	SEL	ATN	MSG	C/D	I/O
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 REQ(Assert SCSI REQ/ signal)
Bit 6 ACK(Assert SCSI ACK/ signal)
Bit 5 BSY(Assert SCSI BSY/ signal)
Bit 4 SEL(Assert SCSI SEL/ signal)
Bit 3 ATN(Assert SCSI ATN/ signal)
Bit 2 MSG(Assert SCSI MSG/ signal)
Bit 1 C/D(Assert SCSI C_D/ signal)
Bit 0 I/O(Assert SCSI I_O/ signal)

Register 0A (8A)
SCSI Selector ID (SSID)
Read Only

VAL	RES	RES	RES	RES	ENID2	ENID1	ENID0
7	6	5	4	3	2	1	0
Default>>>							
0	X	X	X	X	0	0	0

Bit 7 VAL (SCSI Valid Bit)
Bits 6-3 Reserved
Bits 2-0 Encoded Destination SCSI ID

Register 0B (8B)
SCSI Bus Control Lines (SBCL)
Read Only

REQ	ACK	BSY	SEL	ATN	MSG	C/D	I/O
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	X	X	X	X

Bit 7 REQ (SREQ/ status)
Bit 6 ACK (SACK/ status)
Bit 5 BSY (SBSY/ status)
Bit 4 SEL (SSEL/ status)
Bit 3 ATN SATN/ status)
Bit 2 MSG (SMSG/ status)
Bit 1 C/D (SC_D/ status)
Bit 0 I/O (SI_O/ status)

Register 0C (8C)
DMA Status (DSTAT)
Read Only

DFE	MDPE	BF	ABRT	SSI	SIR	RES	IID
7	6	5	4	3	2	1	0
Default>>>							
1	0	0	0	0	0	X	0

Bit 7 DFE (DMA FIFO empty)
Bit 6 MDPE (Master Data Parity Error)
Bit 5 BF (Bus fault)
Bit 4 ABRT (Aborted)
Bit 3 SSI (Single step interrupt)
Bit 2 SIR (SCRIPTS interrupt instruction received)
Bit 1 Reserved
Bit 0 IID (Illegal instruction detected)

Register 0D (8D)
SCSI Status Zero (SSTAT0)
Read Only

ILF	ORF	OLF	AIP	LOA	WOA	RST	SDP0/
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 ILF (SIDL full)
Bit 6 ORF (SODR full)
Bit 5 OLF (SODL full)
Bit 4 AIP (Arbitration in progress)
Bit 3 LOA (Lost arbitration)
Bit 2 WOA (Won arbitration)
Bit 1 RST/ (SCSI RST/ signal)
Bit 0 SDP/ (SCSI SDP/ parity signal)

Register 0E (8E)
SCSI Status One (SSTAT1)
Read Only

FF3	FF2	FF1	FF0	SDP0L	MSG	C/D	I/O
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	X	X	X	X

Bits 7-4 FF3-FF0 (FIFO flags)
Bit 3 SDPL (Latched SCSI parity)
Bit 2 MSG (SCSI MSG/ signal)
Bit 1 C/D (SCSI C_D/ signal)
Bit 0 I/O (SCSI I_O/ signal)

Register 0F (8F)
SCSI Status Two (SSTAT2)
(Read Only)

RES	RES	RES	RES	RES	RES	LDSC	RES
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	X	X	1	X

Bits 7-2 Reserved
Bit 1 LDSC (Last Disconnect)
Bit 0 Reserved

Registers 10-13 (90-93)
Data Structure Address (DSA)
Read/Write

Register 14 (94)
Interrupt Status (ISTAT)
(Read/Write)

ABRT	SRST	SIGP	SEM	CON	INTF	SIP	DIP
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 ABRT (Abort operation)
Bit 6 SRST (Software reset)
Bit 5 SIGP (Signal process)
Bit 4 SEM (Semaphore)
Bit 3 CON (Connected)
Bit 2 INTF (Interrupt on the Fly)
Bit 1 SIP (SCSI interrupt pending)
Bit 0 DIP (DMA interrupt pending)

Register 18 (98)
Chip Test Zero (CTEST0)
Read/Write

Register 19 (99)
Chip Test One (CTEST1)
Read Only

FMT3	FMT2	FMT1	FMT0	FFL3	FFL2	FFL1	FFL0
7	6	5	4	3	2	1	0
Default>>>							
1	1	1	1	0	0	0	0

Bits 7-4 FMT3-0 (Byte empty in DMA FIFO)
Bits 3-0 FFL3-0 (Byte full in DMA FIFO)

Register 1A (9A)
Chip Test Two (CTEST2)
Read Only

DDIR	SIGP	CIO	CM	RES	TEOP	DREQ	DACK
7	6	5	4	3	2	1	0
Default>>>							
0	0	X	X	0	0	0	1

Bit 7 DDIR (Data transfer direction)
Bit 6 SIGP (Signal process)
Bit 5 CIO (Configured as I/O)
Bit 4 CM (Configured as memory)
Bit 3 Reserved
Bit 2 TEOP (SCSI true end of process)
Bit 1 DREQ (Data request status)
Bit 0 DACK (Data acknowledge status)

Register 1B (9B)
Chip Test Three (CTEST3)
Read/Write

V3	V2	V1	V0	FLF	CLF	FM	WRIE
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	0	0	0	0

Bits 7-4 V3-V0 (Chip revision level)
Bit 3 FLF (Flush DMA FIFO)
Bit 2 CLF (Clear DMA FIFO)
Bit 1 FM (Fetch pin mode)
Bit 0 WRIE (Write and Invalidate Enable)

Registers 1C-1F (9C-9F)
Temporary (TEMP)
Read/Write

Register 20 (A0)
DMA FIFO (DFIFO)
Read/Write

RES	BO6	BO5	BO4	Bo3	BO2	BO1	BO0
7	6	5	4	3	2	1	0
Default>>>							
X	0	0	0	0	0	0	0

Bit 7 Reserved
Bits 6-0 BO6-BO0 (Byte offset counter)

Register 21 (A1)
Chip Test Four (CTEST4)
Read/Write

BDIS	ZMOD	ZSD	SRTM	MPEE	FBL2	FBL1	FBL0
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 BDIS (Burst Disable)
Bit 6 ZMOD (High impedance mode)
Bit 5 ZSD (SCSI Data High Impedance)
Bit 4 SRTM (Shadow Register Test Mode)
Bit 3 MPEE (Master Parity Error Enable)
Bits 2-0 FBL2-FBL0 (FIFO byte control)

Register 22 (A2)
Chip Test Five (CTEST5)
Read/Write

ADCK	BBCK	RES	MASR	DDIR	RES	RES	RES
7	6	5	4	3	2	1	0
Default>>>							
0	0	X	0	0	X	X	X

Bit 7 ADCK (Clock address incrementor)
Bit 6 BBCK (Clock byte counter)
Bit 5 Reserved
Bit 4 MASR (Master control for set or reset pulses)
Bit 3 DDIR (DMA direction)
Bits 2-0 Reserved

Register 23 (A3)
Chip Test Six (CTEST6)
Read/Write

DF7	DF6	DF5	DF4	DF3	DF2	DF1	DF0
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bits 7-0 DF7-DF0 (DMA FIFO)

Registers 24-26 (A4-A6)
DMA Byte Counter (DBC)
Read/Write

Register 27 (A7)
DMA Command (DCMD)
Read/Write

Registers 28-2B (A8-AB)
DMA Next Address (DNAD)
Read/Write

Registers 2C-2F (AC-AF)
DMA SCRIPTS Pointer (DSP)
Read/Write

Registers 30-33 (B0-B3)
DMA SCRIPTS Pointer Save (DSPS)
Read/Write

Registers 34-37 (B4-B7)
Scratch Register A (SCRATCH A)
Read/Write

Register 38 (B8)
DMA Mode (DMODE)
Read/Write

BL1	BL0	SIOM	DIOM	ERL	ERMP	BOF	MAN
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7-6 BL1-BL0 (Burst length)
Bit 5 SIOM (Source I/O-Memory Enable)
Bit 4 DIOM (Destination I/O-Memory Enable)
Bit 3 ERL (Enable Read Line)
Bit 2 ERMP (Enable Read Multiple)
Bit 1 BOF (Burst Op Code Fetch Enable)
Bit 0 MAN (Manual Start Mode)

Register 39 (B9)
DMA Interrupt Enable (DIEN)
Read/Write

RES	MDPE	BF	ABRT	SSI	SIR	RES	IID
7	6	5	4	3	2	1	0
Default>>>							
X	0	0	0	0	0	X	0

Bit 7 Reserved
Bit 6 MDPE (Master Data Parity Error)
Bit 5 BF (Bus fault)
Bit 4 ABRT (Aborted)
Bit 3 SSI (Single step interrupt)
Bit 2 SIR (SCRIPTS interrupt instruction received
Bit 1 Reserved
Bit 0 IID (Illegal instruction detected)

Register 3A (BA)
Scratch Byte Register (SBR)
Read/Write

Register 3B (BB)
DMA Control (DCNTL)
Read/Write

CLSE	PFF	PFEN	SSM	IRQM	STD	IRQD	COM
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 CLSE (Cache Line Size Enable)
Bit 6 PFF (Pre-Fetch Flush)
Bit 5 PFEN (Pre-fetch Enable)
Bit 4 SSM (Single-step mode)
Bit 3 IRQM (IRQ Mode)
Bit 2 STD (Start DMA operation)
Bit 1 IRQD (IRQ Disable)
Bit 0 COM (53C700 compatibility)

Register 3C-3F (BC-BF)
Adder Sum Output (ADDER)
Read Only

Register 40 (C0)
SCSI Interrupt Enable Zero (SIEN0)
Read/Write

M/A	CMP	SEL	RSL	SGE	UDC	RST	PAR
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 M/A (SCSI Phase Mismatch - Initiator Mode; SCSI ATN Condition - Target Mode)
Bit 6 CMP (Function Complete)
Bit 5 SEL (Selected)
Bit 4 RSL (Reselected)
Bit 3 SGE (SCSI Gross Error)
Bit 2 UDC (Unexpected Disconnect)
Bit 1 RST (SCSI Reset Condition)
Bit 0 PAR (SCSI Parity Error)

Register 41 (C1)
SCSI Interrupt Enable One (SIEN1)
Read/Write

RES	RES	RES	RES	RES	STO	GEN	HTH
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	X	0	0	0

Bits 7-3 Reserved
Bit 2 STO (Selection or Reselection Time-out)
Bit 1 GEN (General Purpose Timer Expired)
Bit 0 HTH (Handshake to Handshake timer Expired)

Register 42 (C2)
SCSI Interrupt Status Zero (SIST0)
Read Only

M/A	CMP	SEL	RSL	SGE	UDC	RST	PAR
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 M/A (Initiator Mode: Phase Mismatch; Target Mode: SATN/ Active)
Bit 6 CMP (Function Complete)
Bit 5 SEL (Selected)
Bit 4 RSL (Reselected)
Bit 3 SGE (SCSI Gross Error)
Bit 2 UDC (Unexpected Disconnect)
Bit 1 RST (SCSI RST/ Received)
Bit 0 PAR (Parity Error)

Register 43 (C3)
SCSI Interrupt Status One (SIST1)
Read Only

RES	RES	RES	RES	RES	STO	GEN	HTH
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	X	0	0	0

Bits 7-3 Reserved
Bit 2 STO (Selection or Reselection Time-out)
Bit 1 GEN (General Purpose Timer Expired)
Bit 0 HTH (Handshake-to-Handshake Timer Expired)

Register 44 (C4)
SCSI Longitudinal Parity (SLPAR)
Read/Write

Register 46 (C6)
Memory Access Control (MACNTL)
Read/Write

TYP3	TYP2	TYP1	TYP0	DWR	DRD	PSCPT	SCPTS
7	6	5	4	3	2	1	0
Default>>>							
0	1	0	1	0	0	0	0

Bits 7-4 TYP3-0 (Chip Type)
Bit 3 DWR (DataWR)
Bit 2 DRD (DataRD)
Bit 1 PSCPT (Pointer SCRIPTS)
Bit 0 SCPTS (SCRIPTS)

Register 47 (C7)
General Purpose Pin Control (GPCNTL)
Read/Write

ME	FE	RES	RES	RES	RES	GPIO1	GPIO0
7	6	5	4	3	2	1	0
Default>>>							
0	0	X	0	1	1	1	1

Bit 7 Master Enable
Bit 6 Fetch Enable
Bit 5 -2 Reserved
Bits 1-0 GPIO1_EN– GPIO0_EN (GPIO Enable)

Register 48 (C8)
SCSI Timer Zero (STIME0)
Read /Write

HTH	HTH	HTH	HRH	SEL	SEL	SEL	SEL
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bits 7-4 HTH (Handshake-to-Handshake Timer Period)
Bits 3-0 SEL (Selection Time-Out)

Register 49 (C9)
SCSI Timer One (STIME1)
Read/Write

RES	RES	RES	RES	GEN3	GEN2	GEN1	GEN0
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	0	0	0	0

Bits 7-4 Reserved
Bits 3-0 GEN3-0 (General Purpose Timer Period)

Register 4A (CA)
Response ID (RESPID)
Read/Write

Register 4C (CC)
SCSI Test Zero (STEST0)
Read Only

RES	SSAID2	SSAID1	SSAID0	SLT	ART	SOZ	SOM
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	0	X	1	1

Bit 7 Reserved
Bits 6-4 SSAID (SCSI Selected As ID)
Bit 3 SLT (Selection response logic test)
Bit 2 ART (Arbitration Priority Encoder Test)
Bit 1 SOZ (SCSI Synchronous Offset Zero)
Bit 0 SOM (SCSI Synchronous Offset Maximum)

Register 4D (CD)
SCSI Test One (STEST1)
Read/Write

SCLK	SISO	RES	RES	RES	RES	RES	RES
7	6	5	4	3	2	1	0
Default>>>							
0	0	X	X	X	X	X	X

Bit 7 SCLK
Bit 6 SISO (SCSI Isolation Mode)
Bits 5-0 Reserved

Register 4E (CE)
SCSI Test Two (STEST2)
Read/Write

SCE	ROF	RES	SLB	SZM	RES	EXT	LOW
7	6	5	4	3	2	1	0
Default>>>							
0	0	X	0	0	X	0	0

Bit 7 SCE (SCSI Control Enable)
Bit 6 ROF (Reset SCSI Offset)
Bit 5 Reserved
Bit 4 SLB (SCSI Loopback Mode)
Bit 3 SZM (SCSI High-Impedance Mode)
Bit 2 Reserved
Bit 1 EXT(Extend SREQ/SACK filtering)
Bit 0 LOW (SCSI Low level Mode)

Register 4F (CF)
SCSI Test Three (STEST3)
Read/Write

TE	STR	HSC	DSI	RES	TTM	CSF	STW
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	X	0	0	0

Bit 7 TE (TolerANT Enable)
Bit 6 STR (SCSI FIFO Test Read)
Bit 5 HSC (Halt SCSI Clock)
Bit 4 DSI (Disable Single Initiator Response)
Bit 3 Reserved
Bit 2 TTM (Timer Test Mode)
Bit 1 CSF (Clear SCSI FIFO)
Bit 0 STW (SCSI FIFO Test Write)

Register 50 (D0)
SCSI Input Data Latch (SIDL)
Read Only

Registers 54 (D4)
SCSI Output Data Latch (SODL)
Read/Write

Registers 58 (D8)
SCSI Bus Data Lines (SBDL)
Read Only

Registers 5C-5F (DC-DF)
Scratch Register B (SCRATCHB)
(Read/Write)

SYM53C815 Operating Registers

Register 00 (80)
SCSI Control Zero (SCNTL0)
Read/Write

ARB1	ARB0	START	WATN	EPC	RES	AAP	TRG
7	6	5	4	3	2	1	0
Default>>>							
1	1	0	0	0	X	0	0

Bit 7 **ARB1 (Arbitration mode bit 1)**
Bit 6 **ARB0 (Arbitration mode bit 0)**
Bit 5 **START (Start sequence)**
Bit 4 **WATN (Select with SATN/ on a start sequence)**
Bit 3 **EPC (Enable parity checking)**
Bit 2 **Reserved**
Bit 1 **AAP (Assert SATN/ on parity error)**
Bit 0 **TRG (Target role)**

Register 01 (81)
SCSI Control One (SCNTL1)
Read/Write

EXC	ADB	DHP	CON	RST	AESP	IARB	SST
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 **EXC (Extra clock cycle of data setup)**
Bit 6 **ADB (Assert SCSI data bus)**
Bit 5 **DHP (Disable Halt on Parity Error or ATN) (Target Only)**
Bit 4 **CON (Connected)**
Bit 3 **RST (Assert SCSI RST/ signal)**
Bit 2 **AESP (Assert even SCSI parity (force bad parity))**
Bit 1 **IARB (Immediate Arbitration)**
Bit 0 **SST (Start SCSI Transfer)**

Register 02 (82)
SCSI Control Two (SCNTL2)
Read/Write

SDU	RES	RES	RES	RES	RES	RES	RES
7	6	5	4	3	2	1	0
Default>>>							
0	X	X	X	X	X	X	X

Bit 7 **SDU (SCSI Disconnect Unexpected)**
Bits 6-0 **Reserved**

Register 03 (83)
SCSI Control Three (SCNTL3)
Read/Write

RES	SCF2	SCF1	SCF0	RES	CCF2	CCF1	CCF0
7	6	5	4	3	2	1	0
Default>>>							
X	0	0	0	X	0	0	0

Bit 7 **Reserved**
Bits 6-4 **SCF2-0 (Synchronous Clock**
Bit 3 **Reserved**
Bits 2-0 **CCF2-0 (Clock Conversion Factor)**

Register 04 (84)
SCSI Chip ID (SCID)
Read/Write

RES	RRE	SRE	RES	RES	ENC2	ENC1	ENC0
7	6	5	4	3	2	1	0
Default>>>							
X	0	0	X	X	0	0	0

Bit 7 **Reserved**
Bit 6 **RRE (Enable Response to Reselection)**
Bit 5 **SRE (Enable Response to Selection)**
Bit 4-3 **Reserved**
Bits 2-0 **Encoded Chip SCSI ID, bits 2-0**

Register 05 (85)
SCSI Transfer (SXFER)
Read/Write

TP2	TP1	TP0	RES	MO3	MO2	MO1	MO0
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	X	0	0	0	0

Bits 7-5 **TP2-0 (SCSI Synchronous Transfer Period)**
Bit 4 **Reserved**
Bits 3-0 **MO3-MO0 (Max SCSI synchronous offset)**

Register 06 (86)
SCSI Destination ID (SDID)
Read/Write

RES	RES	RES	RES	RES	ENC2	ENC1	ENC0
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	X	0	0	0

Bits 7-3 **Reserved**
Bits 2-0 **Encoded destination SCSI ID**

Register 07 (87)
General Purpose (GPREG)
Read/Write

RES	RES	RES	GPIO4	GPIO3	GPIO2	GPIO1	GPIO0
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	X	X	0	0

Bits 7-2 Reserved
Bits 1-0 GPIO1-GPIO0 (General Purpose)

Register 08 (88)
SCSI First Byte Received (SFBR)
Read/Write

1B7	1B6	1B5	1B4	1B3	1B2	1B1	1B0
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Register 09 (89)
SCSI Output Control Latch (SOCL)
Read /Write

REQ	ACK	BSY	SEL	ATN	MSG	C/D	I/O
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 REQ(Assert SCSI REQ/ signal)
Bit 6 ACK(Assert SCSI ACK/ signal)
Bit 5 BSY(Assert SCSI BSY/ signal)
Bit 4 SEL(Assert SCSI SEL/ signal)
Bit 3 ATN(Assert SCSI ATN/ signal)
Bit 2 MSG(Assert SCSI MSG/ signal)
Bit 1 C/D(Assert SCSI C_D/ signal)
Bit 0 I/O(Assert SCSI I_O/ signal)

Register 0A (8A)
SCSI Selector ID (SSID)
Read Only

VAL	RES	RES	RES	RES	ENID2	ENID1	ENID0
7	6	5	4	3	2	1	0
Default>>>							
0	X	X	X	X	0	0	0

Bit 7 VAL (SCSI Valid Bit)
Bits 6-3 Reserved
Bits 2-0 Encoded Destination SCSI ID

Register 0B (8B)
SCSI Bus Control Lines (SBCL)
Read Only

REQ	ACK	BSY	SEL	ATN	MSG	C/D	I/O
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	X	X	X	X

Bit 7 REQ (SREQ/ status)
Bit 6 ACK (SACK/ status)
Bit 5 BSY (SBSY/ status)
Bit 4 SEL (SSEL/ status)
Bit 3 ATN SATN/ status)
Bit 2 MSG (SMSG/ status)
Bit 1 C/D (SC_D/ status)
Bit 0 I/O (SI_O/ status)

Register 0C (8C)
DMA Status (DSTAT)
Read Only

DFE	MDPE	BF	ABRT	SSI	SIR	RES	IID
7	6	5	4	3	2	1	0
Default>>>							
1	0	0	0	0	0	X	0

Bit 7 DFE (DMA FIFO empty)
Bit 6 MDPE (Master Data Parity Error)
Bit 5 BF (Bus fault)
Bit 4 ABRT (Aborted)
Bit 3 SSI (Single step interrupt)
Bit 2 SIR (SCRIPTS interrupt instruction received)
Bit 1 Reserved
Bit 0 IID (Illegal instruction detected)

Register 0D (8D)
SCSI Status Zero (SSTAT0)
Read Only

ILF	ORF	OLF	AIP	LOA	WOA	RST	SDP0/
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 ILF (SIDL full)
Bit 6 ORF (SODR full)
Bit 5 OLF (SODL full)
Bit 4 AIP (Arbitration in progress)
Bit 3 LOA (Lost arbitration)
Bit 2 WOA (Won arbitration)
Bit 1 RST/ (SCSI RST/ signal)
Bit 0 SDP/ (SCSI SDP/ parity signal)

Register 0E (8E)
SCSI Status One (SSTAT1)
Read Only

FF3	FF2	FF1	FF0	SDP0L	MSG	C/D	I/O
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	X	X	X	X

Bits 7-4 FF3-FF0 (FIFO flags)
Bit 3 SDPL (Latched SCSI parity)
Bit 2 MSG (SCSI MSG/ signal)
Bit 1 C/D (SCSI C_D/ signal)
Bit 0 I/O (SCSI I_O/ signal)

Register 0F (8F)
SCSI Status Two (SSTAT2)
(Read Only)

RES	RES	RES	RES	RES	RES	LDSC	RES
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	X	X	1	X

Bits 7-2 Reserved
Bit 1 LDSC (Last Disconnect)
Bit 0 Reserved

Registers 10-13 (90-93)
Data Structure Address (DSA)
Read/Write

Register 14 (94)
Interrupt Status (ISTAT)
(Read/Write)

ABRT	SRST	SIGP	SEM	CON	INTF	SIP	DIP
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 ABRT (Abort operation)
Bit 6 SRST (Software reset)
Bit 5 SIGP (Signal process)
Bit 4 SEM (Semaphore)
Bit 3 CON (Connected)
Bit 2 INTF (Interrupt on the Fly)
Bit 1 SIP (SCSI interrupt pending)
Bit 0 DIP (DMA interrupt pending)

Register 18 (98)
Chip Test Zero (CTEST0)
Read/Write

Register 19 (99)
Chip Test One (CTEST1)
Read Only

FMT3	FMT2	FMT1	FMT0	FFL3	FFL2	FFL1	FFL0
7	6	5	4	3	2	1	0
Default>>>							
1	1	1	1	0	0	0	0

Bits 7-4 FMT3-0 (Byte empty in DMA FIFO)
Bits 3-0 FFL3-0 (Byte full in DMA FIFO)

Register 1A (9A)
Chip Test Two (CTEST2)
Read Only

DDIR	SIGP	CIO	CM	RES	TEOP	DREQ	DACK
7	6	5	4	3	2	1	0
Default>>>							
0	0	X	X	0	0	0	1

Bit 7 DDIR (Data transfer direction)
Bit 6 SIGP (Signal process)
Bit 5 CIO (Configured as I/O)
Bit 4 CM (Configured as memory)
Bit 3 Reserved
Bit 2 TEOP (SCSI true end of process)
Bit 1 DREQ (Data request status)
Bit 0 DACK (Data acknowledge status)

Register 1B (9B)
Chip Test Three (CTEST3)
Read/Write

V3	V2	V1	V0	FLF	CLF	FM	RES
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	0	0	0	0

Bits 7-4 V3-V0 (Chip revision level)
Bit 3 FLF (Flush DMA FIFO)
Bit 2 CLF (Clear DMA FIFO)
Bit 1 FM (Fetch pin mode)
Bit 0 Reserved

Registers 1C-1F (9C-9F)
Temporary (TEMP)
Read/Write

Register 20 (A0)
DMA FIFO (DFIFO)
Read/Write

RES	BO6	BO5	BO4	Bo3	BO2	BO1	BO0
7	6	5	4	3	2	1	0
Default>>>							
X	0	0	0	0	0	0	0

Bit 7 Reserved
Bits 6-0 BO6-BO0 (Byte offset counter)

Register 21 (A1)
Chip Test Four (CTEST4)
Read/Write

BDIS	ZMOD	ZSD	SRTM	MPEE	FBL2	FBL1	FBL0
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 BDIS (Burst Disable)
Bit 6 ZMOD (High impedance mode)
Bit 5 ZSD (SCSI Data High Impedance)
Bit 4 SRTM (Shadow Register Test Mode)
Bit 3 MPEE (Master Parity Error Enable)
Bits 2-0 FBL2-FBL0 (FIFO byte control)

Register 22 (A2)
Chip Test Five (CTEST5)
Read/Write

ADCK	BBCK	RES	MASR	DDIR	RES	RES	RES
7	6	5	4	3	2	1	0
Default>>>							
0	0	X	0	0	X	X	X

Bit 7 ADCK (Clock address incrementor)
Bit 6 BBCK (Clock byte counter)
Bit 5 Reserved
Bit 4 MASR (Master control for set or reset pulses)
Bit 3 DDIR (DMA direction)
Bits 2-0 Reserved

Register 23 (A3)
Chip Test Six (CTEST6)
Read/Write

DF7	DF6	DF5	DF4	DF3	DF2	DF1	DF0
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bits 7-0 DF7-DF0 (DMA FIFO)

Registers 24-26 (A4-A6)
DMA Byte Counter (DBC)
Read/Write

Register 27 (A7)
DMA Command (DCMD)
Read/Write

Registers 28-2B (A8-AB)
DMA Next Address (DNAD)
Read/Write

Registers 2C-2F (AC-AF)
DMA SCRIPTS Pointer (DSP)
Read/Write

Registers 30-33 (B0-B3)
DMA SCRIPTS Pointer Save (DSPS)
Read/Write

Registers 34-37 (B4-B7)
Scratch Register A (SCRATCH A)
Read/Write

Register 38 (B8)
DMA Mode (DMODE)
Read/Write

BL1	BL0	SIOM	DIOM	RES	RES	BOF	MAN
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7-6 BL1-BL0 (Burst length)
Bit 5 SIOM (Source I/O-Memory Enable)
Bit 4 DIOM (Destination I/O-Memory Enable)
Bit 3 ERL (Enable Read Line)
Bit 2 Reserved
Bit 2 Reserved
Bit 1 BOF (Burst Op Code Fetch Enable)
Bit 0 MAN (Manual Start Mode)

Register 39 (B9)
DMA Interrupt Enable (DIEN)
Read/Write

RES	MDPE	BF	ABRT	SSI	SIR	RES	IID
7	6	5	4	3	2	1	0
Default>>>							
X	0	0	0	0	0	X	0

Bit 7 Reserved
Bit 6 MDPE (Master Data Parity Error)
Bit 5 BF (Bus fault)
Bit 4 ABRT (Aborted)
Bit 3 SSI (Single step interrupt)
Bit 2 SIR (SCRIPTS interrupt instruction received
Bit 1 Reserved
Bit 0 IID (Illegal instruction detected)

Register 3A (BA)
DMA Watchdog Timer (DWT)
Read/Write

Register 3B (BB)
DMA Control (DCNTL)
Read/Write

RES	RES	RES	SSM	IRQM	STD	RES	COM
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bits 7-5 Reserved
Bit 4 SSM (Single-step mode)
Bit 3 IRQM (IRQ Mode)
Bit 2 STD (Start DMA operation)
Bit 1 Reserved
Bit 0 COM (53C700 compatibility)

Register 3C-3F (BC-BF)
Adder Sum Output (ADDER)
Read Only

Register 40 (C0)
SCSI Interrupt Enable Zero (SIEN0)
Read/Write

M/A	CMP	SEL	RSL	SGE	UDC	RST	PAR
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 M/A (SCSI Phase Mismatch - Initiator Mode; SCSI ATN Condition - Target Mode)
Bit 6 CMP (Function Complete)
Bit 5 SEL (Selected)
Bit 4 RSL (Reselected)
Bit 3 SGE (SCSI Gross Error)
Bit 2 UDC (Unexpected Disconnect)
Bit 1 RST (SCSI Reset Condition)
Bit 0 PAR (SCSI Parity Error)

Register 41 (C1)
SCSI Interrupt Enable One (SIEN1)
Read/Write

RES	RES	RES	RES	RES	STO	GEN	HTH
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	X	0	0	0

Bits 7-3 Reserved
Bit 2 STO (Selection or Reselection Time-out)
Bit 1 GEN (General Purpose Timer Expired)
Bit 0 HTH (Handshake to Handshake timer Expired)

Register 42 (C2)
SCSI Interrupt Status Zero (SIST0)
Read Only

M/A	CMP	SEL	RSL	SGE	UDC	RST	PAR
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 M/A (Initiator Mode: Phase Mismatch; Target Mode: SATN/ Active)
Bit 6 CMP (Function Complete)
Bit 5 SEL (Selected)
Bit 4 RSL (Reselected)
Bit 3 SGE (SCSI Gross Error)
Bit 2 UDC (Unexpected Disconnect)
Bit 1 RST (SCSI RST/ Received)
Bit 0 PAR (Parity Error)

Register 43 (C3)
SCSI Interrupt Status One (SIST1)
Read Only

RES	RES	RES	RES	RES	STO	GEN	HTH
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	X	0	0	0

Bits 7-3 Reserved
Bit 2 STO (Selection or Reselection Time-out)
Bit 1 GEN (General Purpose Timer Expired)
Bit 0 HTH (Handshake-to-Handshake Timer Expired)

Register 44 (C4)
SCSI Longitudinal Parity (SLPAR)
Read/Write

Register 46 (C6)
Memory Access Control (MACNTL)
Read/Write

TYP3	TYP2	TYP1	TYP0	DWR	DRD	PSCPT	SCPTS
7	6	5	4	3	2	1	0
Default>>>							
00	11	00	10	0	0	0	0

Bits 7-4 TYP3-0 (Chip Type)
Bit 3 DWR (DataWR)
Bit 2 DRD (DataRD)
Bit 1 PSCPT (Pointer SCRIPTS)
Bit 0 SCPTS (SCRIPTS)

Register 47 (C7)
General Purpose Pin Control (GPCNTL)
Read/Write

ME	FE	RES	GPIO4	GPIO3	GPIO2	GPIO1	GPIO0
7	6	5	4	3	2	1	0
Default>>>							
0	0	X	0	1	1	1	1

Bit 7 Master Enable
Bit 6 Fetch Enable
Bit 5 Reserved
Bits 4-2 GPIO4_EN–GPIO2_EN (GPIO Enable)
Bits 1-0 GPIO1_EN– GPIO0_EN (GPIO Enable)

Register 48 (C8)
SCSI Timer Zero (STIME0)
Read /Write

HTH	HTH	HTH	HRH	SEL	SEL	SEL	SEL
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bits 7-4 HTH (Handshake-to-Handshake Timer Period)
Bits 3-0 SEL (Selection Time-Out)

Register 49 (C9)
SCSI Timer One (STIME1)
Read/Write

RES	RES	RES	RES	GEN3	GEN2	GEN1	GEN0
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	0	0	0	0

Bits 7-4 Reserved
Bits 3-0 GEN3-0 (General Purpose Timer Period)

Register 4A (CA)
Response ID (RESPID)
Read/Write

Register 4C (CC)
SCSI Test Zero (STEST0)
Read Only

RES	RES	RES	RES	SLT	ART	SOZ	SOM
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	0	X	1	1

Bits 7-4 Reserved
Bit 3 SLT (Selection response logic test)
Bit 2 ART (Arbitration Priority Encoder Test)
Bit 1 SOZ (SCSI Synchronous Offset Zero)
Bit 0 SOM (SCSI Synchronous Offset Maximum)

Register 4D (CD)
SCSI Test One (STEST1)
Read/Write

SCLK	RES	RES	RES	RES	RES	RES	RES
7	6	5	4	3	2	1	0
Default>>>							
0	0	X	X	X	X	X	X

Bit 7 SCLK
Bits 5-0 Reserved

Register 4E (CE)
SCSI Test Two (STEST2)
Read/Write

SCE	ROF	RES	SLB	SZM	RES	EXT	LOW
7	6	5	4	3	2	1	0
Default>>>							
0	0	X	0	0	X	0	0

Bit 7 SCE (SCSI Control Enable)
Bit 6 ROF (Reset SCSI Offset)
Bit 5 Reserved
Bit 4 SLB (SCSI Loopback Mode)
Bit 3 SZM (SCSI High-Impedance Mode)
Bit 2 Reserved
Bit 1 EXT(Extend SREQ/SACK filtering)
Bit 0 LOW (SCSI Low level Mode)

Register 4F (CF)
SCSI Test Three (STEST3)
Read/Write

TE	STR	HSC	DSI	RES	TTM	CSF	STW
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	X	0	0	0

Bit 7 TE (TolerANT Enable)
Bit 6 STR (SCSI FIFO Test Read)
Bit 5 HSC (Halt SCSI Clock)
Bit 4 DSI (Disable Single Initiator Response)
Bit 3 Reserved
Bit 2 TTM (Timer Test Mode)
Bit 1 CSF (Clear SCSI FIFO)
Bit 0 STW (SCSI FIFO Test Write)

Register 50 (D0)
SCSI Input Data Latch (SIDL)
Read Only

Registers 54 (D4)
SCSI Output Data Latch (SODL)
Read/Write

Registers 58 (D8)
SCSI Bus Data Lines (SBDL)
Read Only

Registers 5C-5F (DC-DF)
Scratch Register B (SCRATCHB)
(Read/Write)

SYM53C825A Operating Registers

Register 00 (80)
SCSI Control Zero (SCNTL0)
Read/Write

ARB1	ARB0	START	WATN	EPC	RES	AAP	TRG
7	6	5	4	3	2	1	0
Default>>>							
1	1	0	0	0	X	0	0

Bit 7 ARB1 (Arbitration Mode bit 1)
Bit 6 ARB0 (Arbitration Mode bit 0)
Bit 5 START (Start Sequence)
Bit 4 WATN (Select with SATN/ on a Start Sequence)
Bit 3 EPC (Enable Parity Checking)
Bit 2 Reserved
Bit 1 AAP (Assert SATN/ on Parity Error)
Bit 0 TRG (Target Mode)

Register 01 (81)
SCSI Control One (SCNTL1)
Read/Write

EXC	ADB	DHP	CON	RST	AESP	IARB	SST
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 EXC (Extra Clock Cycle of Data Setup)
Bit 6 ADB (Assert SCSI Data Bus)
Bit 5 DHP (Disable Halt on Parity Error or ATN) (Target Only)
Bit 4 CON (Connected)
Bit 3 RST (Assert SCSI RST/ Signal)
Bit 2 AESP (Assert Even SCSI Parity (force bad parity))
Bit 1 IARB (Immediate Arbitration)
Bit 0 SST (Start SCSI Transfer)

Register 02 (82)
SCSI Control Two (SCNTL2)
Read/Write

SDU	CHM	SLPMD	SLPHBEN	WSS	VUE0	VUE1	WSR
7	6	5	4	3	2	1	0
Default>>>							
0	0	X0	X0	0	0X	0X	0

Bit 7 SDU (SCSI Disconnect Unexpected)
Bit 6 CHM (Chained Mode)
Bit 5 SLPMD (SLPAR Mode Bit)
Bit 4 SLPHBEN (SLPAR High Byte Enable)
Bit 3 WSS (Wide SCSI Send)
Bit 2 VUE0 (Vendor Unique Enhancements bit 0)
Bit 1 VUE1 (Vendor Unique Enhancements bit 1)
Bit 0 WSR (Wide SCSI Receive)

Register 03 (83)
SCSI Control Three (SCNTL3)
Read/Write

RES	SCF2	SCF1	SCF0	EWS	CCF2	CCF1	CCF0
7	6	5	4	3	2	1	0
Default>>>							
X0	0	0	0	0	0	0	0

Bit 7 Reserved
Bits 6-4 SCF2-0 (Synchronous Clock Conversion Factor)
Bit 3 EWS (Enable Wide SCSI)
Bits 2-0 CCF2-0 (Clock Conversion Factor)

Register 04 (84)
SCSI Chip ID (SCID)
Read/Write

RES	RRE	SRE	RES	ENC3	ENC2	ENC1	ENC0
7	6	5	4	3	2	1	0
Default>>>							
X	0	0	X	0	0	0	0

Bit 7 Reserved
Bit 6 RRE (Enable Response to Reselection)
Bit 5 SRE (Enable Response to Selection)
Bit 4 Reserved
Bits 3-0 Encoded Chip SCSI ID, bits 3-0

Register 05 (85)
SCSI Transfer (SXFER)
Read/Write

TP2	TP1	TP0	MO4	MO3	MO2	MO1	MO0
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	X0	0	0	0	0

Bits 7-5 TP2-0 (SCSI Transfer Period)
Bits 4-0 MO4-MO0 (Max SCSI Synchronous Offset)

Register 06 (86)
SCSI Destination ID (SDID)
Read/Write

RES	RES	RES	RES	ENC3	ENC2	ENC1	ENC0
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	0	0	0	0

Bits 7-4 Reserved
Bits 3-0 Encoded Destination SCSI ID

Register 07 (87)
General Purpose (GPREG)
Read/Write

RES	RES	RES	GPIO4	GPIO3	GPIO2	GPIO1	GPIO0
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	0	X	X	X	X

Bits 7-5 Reserved
Bits 4-0 GPIO4-GPIO0 (General Purpose)

Register 08 (88)
SCSI First Byte Received (SFBR)
Read/Write

1B7	1B6	1B5	1B4	1B3	1B2	1B1	1B0
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Register 09 (89)
SCSI Output Control Latch (SOCL)
Read /Write

REQ	ACK	BSY	SEL	ATN	MSG	C/D	I/O
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 REQ(Assert SCSI REQ/ Signal)
Bit 6 ACK(Assert SCSI ACK/ Signal)
Bit 5 BSY(Assert SCSI BSY/ Signal)
Bit 4 SEL(Assert SCSI SEL/ Signal)
Bit 3 ATN(Assert SCSI ATN/ Signal)
Bit 2 MSG(Assert SCSI MSG/ Signal)
Bit 1 C/D(Assert SCSI C_D/ Signal)
Bit 0 I/O(Assert SCSI I_O/ Signal)

Register 0A (09)
SCSI Selector ID (SSID)
Read Only

VAL	RES	RES	RES	ENID3	ENID2	ENID1	ENID0
7	6	5	4	3	2	1	0
Default>>>							
0	X	X	X	0	0	0	0

Bit 7 VAL (SCSI Valid)
Bits 6-4 Reserved
Bits 3-0 Encoded Destination SCSI ID

Register 0B (8B)
SCSI Bus Control Lines (SBCL)
Read Only

REQ	ACK	BSY	SEL	ATN	MSG	C/D	I/O
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	X	X	X	X

Bit 7 REQ (SREQ/ Status)
Bit 6 ACK (SACK/ Status)
Bit 5 BSY (SBSY/ Status)
Bit 4 SEL (SSEL/ Status)
Bit 3 ATN (SATN/ Status)
Bit 2 MSG (SMSG/ Status)
Bit 1 C/D (SC_D/ Status)
Bit 0 I/O (SI_O/ Status)

Register 0C (8C)
DMA Status (DSTAT)
Read Only

DFE	MDPE	BF	ABRT	SSI	SIR	RES	IID
7	6	5	4	3	2	1	0
Default>>>							
1	0	0	0	0	0	X	0

Bit 7 DFE (DMA FIFO Empty)
Bit 6 MDPE (Master Data Parity Error)
Bit 5 BF (Bus Fault)
Bit 4 ABRT (Aborted)
Bit 3 SSI (Single Step Interrupt)
Bit 2 SIR (SCRIPTS Interrupt Instruction Received)
Bit 1 Reserved
Bit 0 IID (Illegal Instruction Detected)

Register 0D (8D)
SCSI Status Zero (SSTAT0)
Read Only

ILF	ORF	OLF	AIP	LOA	WOA	RST	SDP0/
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 ILF (SIDL Least Significant Byte Full)
Bit 6 ORF (SODR Least Significant Byte Full)
Bit 5 OLF (SODL Least Significant Byte Full)
Bit 4 AIP (Arbitration in Progress)
Bit 3 LOA (Lost Arbitration)
Bit 2 WOA (Won Arbitration)
Bit 1 RST/ (SCSI RST/ Signal)
Bit 0 SDP0/ (SCSI SDP0/ Parity Signal)

Register 0E (8E)
SCSI Status One (SSTAT1)
Read Only

FF3	FF2	FF1	FF0	SDP0L	MSG	C/D	I/O
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	X	X	X	X

Bits 7-4 FF3-FF0 (FIFO Flags
Bit 3 SDP0L (Latched SCSI Parity)
Bit 2 MSG (SCSI MSG/ Signal)
Bit 1 C/D (SCSI C_D/ Signal)
Bit 0 I/O (SCSI I_O/ Signal)

Register 0F (8F)
SCSI Status Two (SSTAT2)
(Read Only)

ILF1	ORF1	OLF1	FF4	SPL1	RES	LDSC	SDP1
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	X	X	1	X

Bit 7 ILF1 (SIDL Most Significant Byte Full)
Bit 6 ORF1 (SODR Most Significant Byte Full)
Bit 5 OLF1 (SODL Most Significant Byte Full)
Bit 4 FF4 (FIFO Flags bit 4)
Bit 3 SPL1(Latched SCSI parity for SD15-8)
Bit 2 DIFFSENSE SENSE
Bit 1 LDSC (Last Disconnect)
Bit 0 SDP1 (SCSI SDP1 Signal)

Registers 10-13 (90-93)
Data Structure Address (DSA)
Read/Write

Register 14 (94)
Interrupt Status (ISTAT)
(Read/Write)

ABRT	SRST	SIGP	SEM	CON	INTF	SIP	DIP
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 ABRT (Abort Operation)
Bit 6 SRST (Software Reset)
Bit 5 SIGP (Signal Process)
Bit 4 SEM (Semaphore)
Bit 3 CON (Connected)
Bit 2 INTF (Interrupt on the Fly)
Bit 1 SIP (SCSI Interrupt Pending)
Bit 0 DIP (DMA Interrupt Pending)

Register 18 (98)
Chip Test Zero (CTEST0)
Read/Write

Register 19 (99)
Chip Test One (CTEST1)
Read Only

FMT3	FMT2	FMT1	FMT0	FFL3	FFL2	FFL1	FFL0
7	6	5	4	3	2	1	0
Default>>>							
1	1	1	1	0	0	0	0

Bits 7-4 FMT3-0 (Byte Empty in DMA FIFO)
Bits 3-0 FFL3-0 (Byte Full in DMA FIFO)

Register 1A (9A)
Chip Test Two (CTEST2)
Read/Write

DDIR	SIGP	CIO	CM	SRTCH	TEOP	DREQ	DACK
7	6	5	4	3	2	1	0
Default>>>							
0	0	X	X	0	0	0	1

Bit 7 DDIR (Data Transfer Direction)
Bit 6 SIGP (Signal Process)
Bit 5 CIO (Configured as I/O)
Bit 4 CM (Configured as Memory)
Bit 3 SRTCH (SCRATCHA/B Operation)
Bit 2 TEOP (SCSI True End of Process)
Bit 1 DREQ (Data Request Status)
Bit 0 DACK (Data Acknowledge Status)

Register 1B (9B)
Chip Test Three (CTEST3)
Read/Write

V3	V2	V1	V0	FLF	CLF	FM	RESW-RIE
7	6	5	4	3	2	1	0
Default>>>							
x	x	x	x	0	0	0	X0

Bits 7-4 V3-V0 (Chip revision level)
Bit 3 FLF (Flush DMA FIFO)
Bit 2 CLF (Clear DMA FIFO)
Bit 1 FM (Fetch Pin Mode)
Bit 0 ReservedWRIE (Write and Invalidate Enable)

Registers 1C-1F (9C-9F)
Temporary (TEMP)
Read/Write

Register 20 (A0)
DMA FIFO (DFIFO)
Read/Write

BO7	BO6	BO5	BO4	Bo3	BO2	BO1	BO0
7	6	5	4	3	2	1	0
Default>>>							
X	0	0	0	0	0	0	0

Bits 7-0 BO76BO0 (Byte offset counter)

Register 21 (A1)
Chip Test Four (CTEST4)
Read/Write

BDIS	ZMOD	ZSD	SRTM	MPEE	FBL2	FBL1	FBL0
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 BDIS (Burst Disable)
Bit 6 ZMOD (High Impedance Mode)
Bit 5 ZSD (SCSI Data High Impedance)
Bit 4 SRTM (Shadow Register Test Mode)
Bit 3 MPEE (Master Parity Error Enable)
Bits 2-0 FBL2-FBL0 (FIFO Byte Control)

Register 22 (A2)
Chip Test Five (CTEST5)
Read/Write

ADCK	BBCK	DFS	MASR	DDIR	BL2	BO9	BO8
7	6	5	4	3	2	1	0
Default>>>							
0	0	0X	0	0	X	X	X

Bit 7 ADCK (Clock Address Incrementor)
Bit 6 BBCK (Clock Byte Counter)
Bit 5 DFS (DMA FIFO Size)
Bit 4 MASR (Master Control for Set or Reset Pulses)
Bit 3 DDIR (DMA Direction)
Bit 2 BL2 (Burst Length bit 2)
Bits 1-0 BO9-8

Register 23 (A3)
Chip Test Six (CTEST6)
Read/Write

DF7	DF6	DF5	DF4	DF3	DF2	DF1	DF0
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bits 7-0 DF7-DF0 (DMA FIFO)

Registers 24-26 (A4-A6)
DMA Byte Counter (DBC)
Read/Write

Register 27 (A7)
DMA Command (DCMD)
Read/Write

Registers 28-2B (A8-AB)
DMA Next Address (DNAD)
Read/Write

Registers 2C-2F (AC-AF)
DMA SCRIPTS Pointer (DSP)
Read/Write

Registers 30-33 (B0-B3)
DMA SCRIPTS Pointer Save (DSPS)
Read/Write

Registers 34-37 (B4-B7)
Scratch Register A (SCRATCH A)
Read/Write

Register 38 (B8)
DMA Mode (DMODE)
Read/Write

BL1	BL0	SIOM	DIOM	ER	ERMP	BOF	MAN
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	X0	0	0

Bit 7-6 BL1-BL0 (Burst Length)
Bit 5 SIOM (Source I/O-Memory Enable)
Bit 4 DIOM (Destination I/O-Memory Enable)
Bit 3 ERL (Enable Read Line)
Bit 2 ERMP (Enable Read Multiple)
Bit 1 BOF (Burst Op Code Fetch Enable)
Bit 0 MAN (Manual Start Mode)

Register 39 (B9)
DMA Interrupt Enable (DIEN)
Read/Write

RES	MDPE	BF	ABRT	SSI	SIR	RES	IID
7	6	5	4	3	2	1	0
Default>>>							
X	0	0	0	0	0	X	0

Bit 7 Reserved
Bit 6 MDPE (Master Data Parity Error)
Bit 5 BF (Bus Fault)
Bit 4 ABRT (Aborted)
Bit 3 SSI (Single -step Interrupt)
Bit 2 SIR (SCRIPTS Interrupt Instruction Received
Bit 1 Reserved
Bit 0 IID (Illegal Instruction Detected)

Register 3A (BA)
Scratch Byte Register (SBR)
Read/Write

Register 3B (BB)
DMA Control (DCNTL)
Read/Write

CLSE	PFF	PFEN	SSM	IRQM	STD	IRQD	COM
7	6	5	4	3	2	1	0
Default>>>							
X0	X0	X0	0	0	0	X0	0

Bit 7 CLSE (Cache Line Size Enable)
Bit 6 PFF (Pre-fetch Flush)
Bit 5 PFEN (Pre-fetch Enable)
Bit 4 SSM (Single-step Mode)
Bit 3 IRQM (IRQ Mode)
Bit 2 STD (Start DMA Operation)
Bit 1 IRQD (IRQ Disable)
Bit 0 COM (53C700 Compatibility)

Register 3C-3F (BC-BF)
Adder Sum Output (ADDER)
Read Only

Register 40 (C0)
SCSI Interrupt Enable Zero (SIEN0)
Read/Write

M/A	CMP	SEL	RSL	SGE	UDC	RST	PAR
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 M/A (SCSI Phase Mismatch - Initiator Mode; SCSI ATN Condition - Target Mode)
Bit 6 CMP (Function Complete)
Bit 5 SEL (Selected)
Bit 4 RSL (Reselected)
Bit 3 SGE (SCSI Gross Error)
Bit 2 UDC (Unexpected Disconnect)
Bit 1 RST (SCSI Reset Condition)
Bit 0 PAR (SCSI Parity Error)

Register 41 (C1)
SCSI Interrupt Enable One (SIEN1)
Read/Write

RES	RES	RES	RES	RES	STO	GEN	HTH
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	X	0	0	0

Bits 7-3 Reserved
Bit 2 STO (Selection or Reselection Time-out)
Bit 1 GEN (General Purpose Timer Expired)
Bit 0 HTH (Handshake-to-Handshake Timer Expired)

Register 42 (C2)
SCSI Interrupt Status Zero (SIST0)
Read Only

M/A	CMP	SEL	RSL	SGE	UDC	RST	PAR
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 M/A (Initiator Mode: Phase Mismatch; Target Mode: SATN/ Active)
Bit 6 CMP (Function Complete)
Bit 5 SEL (Selected)
Bit 4 RSL (Reselected)
Bit 3 SGE (SCSI Gross Error)
Bit 2 UDC (Unexpected Disconnect)
Bit 1 RST (SCSI RST/ Received)
Bit 0 PAR (Parity Error)

Register 43 (C3)
SCSI Interrupt Status One (SIST1)
Read Only

RES	RES	RES	RES	RES	STO	GEN	HTH
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	X	0	0	0

Bits 7-3 Reserved
Bit 2 STO (Selection or Reselection Time-out)
Bit 1 GEN (General Purpose Timer Expired)
Bit 0 HTH (Handshake-to-Handshake Timer Expired)

Register 44 (C4)
SCSI Longitudinal Parity (SLPAR)
Read/Write

Register 45 (C5)
SCSI Wide Residue (SWIDE)
Read/Write

Register 46 (C6)
Memory Access Control (MACNTL)
Read/Write

TYP3	TYP2	TYP1	TYP0	DWR	DRD	PSCPT	SCPTS
7	6	5	4	3	2	1	0
Default>>>							
0	01	1	10	0	0	0	0

Bits 7-4 TYP3-0 (Chip Type)
Bit 3 DWR (DataWR)
Bit 2 DRD (DataRD)
Bit 1 PSCPT (Pointer SCRIPTS)
Bit 0 SCPTS (SCRIPTS)

Register 47 (C7)
General Purpose Pin Control (GPCNTL)
Read/Write

ME	FE	RES	GPIO4	GPIO3	GPIO2	GPIO1	GPIO0
7	6	5	4	3	2	1	0
Default>>>							
0	0	X	0	1	1	1	1

Bit 7 Master Enable
Bit 6 Fetch Enable
Bit 5 Reserved
Bits 4-2 GPIO4_EN–GPIO2_EN (GPIO Enable)
Bits 1-0 GPIO1_EN– GPIO0_EN (GPIO Enable)

Register 48 (C8)
SCSI Timer Zero (STIME0)
Read /Write

HTH	HTH	HTH	HRH	SEL	SEL	SEL	SEL
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bits 7-4 HTH (Handshake-to-Handshake Timer Period)
Bits 3-0 SEL (Selection Time-Out)

Register 49 (C9)
SCSI Timer One (STIME1)
Read/Write

RES	HTHBA	GENSF	HTHSF	GEN3	GEN2	GEN1	GEN0
7	6	5	4	3	2	1	0
Default>>>							
X	0X	0X	0X	0	0	0	0

Bit 7 Reserved
Bit 6 HTHBA (Handshake-to-Handshake Timer Bus Activity Enable)
Bit 5 GENSF (General Purpose Timer Scale Factor)
Bit 4 HTHSF (Handshake to Handshake Timer Scale Factor)
Bits 3-0 GEN3-0 (General Purpose Timer Period)

Register 4A (CA)
Response ID Zero (RESPID0)
Read/Write

Register 4B (CB)
Response ID One(RESPID1)
Read/Write

Register 4C (CC)
SCSI Test Zero (STEST0)
Read Only

SSAID3	SSAID2	SSAID1	SSAID0	SLT	ART	SOZ	SOM
7	6	5	4	3	2	1	0
Default>>>							
0X	0X	0X	0X	0	X	1	1

Bits 7-4 SSAID (SCSI Selected As ID)
Bit 3 SLT (Selection Response Logic Test)
Bit 2 ART (Arbitration Priority Encoder Test)
Bit 1 SOZ (SCSI Synchronous Offset Zero)
Bit 0 SOM (SCSI Synchronous Offset Maximum)

Register 4D (CD)
SCSI Test One (STEST1)
Read/Write

SCLK	SISO	RES	RES	RES	RES	RES	RES
7	6	5	4	3	2	1	0
Default>>>							
0	X0	X	X	X	X	X	X

Bit 7 SCLK
Bit 6 SISO (SCSI Isolation Mode)
Bits 5-0 Reserved

Register 4E (CE)
SCSI Test Two (STEST2)
Read/Write

SCE	ROF	DIF	SLB	SZM	AWS	EXT	LOW
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 SCE (SCSI Control Enable)
Bit 6 ROF (Reset SCSI Offset)
Bit 5 DIF (SCSI Differential Mode)
Bit 4 SLB (SCSI Loopback Mode)
Bit 3 SZM (SCSI High-Impedance Mode)
Bit 1 EXT (Extend SREQ/SACK Filtering)
Bit 0 LOW (SCSI Low level Mode)

Register 4F (CF)
SCSI Test Three (STEST3)
Read/Write

TE	STR	HSC	DSI	S16	TTM	CSF	STW
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 TE (TolerANT Enable)
Bit 6 STR (SCSI FIFO Test Read)
Bit 5 HSC (Halt SCSI Clock)
Bit 4 DSI (Disable Single Initiator Response)
Bit 3 S16 (16-bit System)
Bit 2 TTM (Timer Test Mode)
Bit 1 CSF (Clear SCSI FIFO)
Bit 0 STW (SCSI FIFO Test Write)

Register 50-51 (D0-D1)
SCSI Input Data Latch (SIDL)
Read Only

Registers 54-55 (D4-D5)
SCSI Output Data Latch (SODL)
Read/Write

Registers 58-59 (D8-D9)
SCSI Bus Data Lines (SBDL)
Read Only

Registers 5C-5F (DC-DF)
Scratch Register B (SCRATCHB)
(Read/Write)

Registers 60h-7Fh (E0h-FFh)
Scratch Registers C-J
(SCRATCHC-SCRATCHJ)
Read/Write

SYM53C860 Operating Registers

Register 00 (80)
SCSI Control Zero (SCNTL0)
Read/Write

ARB1	ARB0	START	WATN	EPC	RES	AAP	TRG
7	6	5	4	3	2	1	0
Default>>>							
1	1	0	0	0	X	0	0

Bit 7 ARB1 (Arbitration mode bit 1)
Bit 6 ARB0 (Arbitration mode bit 0)
Bit 5 START (Start sequence)
Bit 4 WATN (Select with SATN/ on a start sequence)
Bit 3 EPC (Enable parity checking)
Bit 2 Reserved
Bit 1 AAP (Assert SATN/ on parity error)
Bit 0 TRG (Target role)

Register 01 (81)
SCSI Control One (SCNTL1)
Read/Write

EXC	ADB	DHP	CON	RST	AESP	IARB	SST
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 EXC (Extra clock cycle of data setup)
Bit 6 ADB (Assert SCSI data bus)
Bit 5 DHP (Disable Halt on Parity Error or ATN) (Target Only)
Bit 4 CON (Connected)
Bit 3 RST (Assert SCSI RST/ signal)
Bit 2 AESP (Assert even SCSI parity (force bad parity))
Bit 1 IARB (Immediate Arbitration)
Bit 0 SST (Start SCSI Transfer)

Register 02 (82)
SCSI Control Two (SCNTL2)
Read/Write

SDU	RES	RES	RES	RES	RES	RES	RES
7	6	5	4	3	2	1	0
Default>>>							
0	X	X	X	X	X	X	X

Bit 7 SDU (SCSI Disconnect Unexpected)
Bits 6-0 Reserved

Register 03 (83)
SCSI Control Three (SCNTL3)
Read/Write

ULTRA	SCF2	SCF1	SCF0	RES	CCF2	CCF1	CCF0
7	6	5	4	3	2	1	0
Default>>>							
X	0	0	0	X	0	0	0

Bit 7 ULTRA (Ultra Enable)
Bits 6-4 SCF2-0 (Synchronous Clock
Bit 3 Reserved
Bits 2-0 CCF2-0 (Clock Conversion Factor)

Register 04 (84)
SCSI Chip ID (SCID)
Read/Write

RES	RRE	SRE	RES	RES	ENC2	ENC1	ENC0
7	6	5	4	3	2	1	0
Default>>>							
X	0	0	X	X	0	0	0

Bit 7 Reserved
Bit 6 RRE (Enable Response to Reselection)
Bit 5 SRE (Enable Response to Selection)
Bit 4-3 Reserved
Bits 2-0 Encoded Chip SCSI ID, bits 2-0

Register 05 (85)
SCSI Transfer (SXFER)
Read/Write

TP2	TP1	TP0	RES	MO3	MO2	MO1	MO0
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	X	0	0	0	0

Bits 7-5 TP2-0 (SCSI Synchronous Transfer Period)
Bit 4 Reserved
Bits 3-0 MO3-MO0 (Max SCSI synchronous offset)

Register 06 (86)
SCSI Destination ID (SDID)
Read/Write

RES	RES	RES	RES	RES	ENC2	ENC1	ENC0
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	X	0	0	0

Bits 7-3 Reserved
Bits 2-0 Encoded destination SCSI ID

Register 07 (87)
General Purpose (GPREG)
Read/Write

RES	RES	RES	RES	RES	RES	GPIO1	GPIO0
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	X	X	0	0

Bits 7-2 Reserved
Bits 1-0 GPIO1-GPIO0 (General Purpose)

Register 08 (88)
SCSI First Byte Received (SFBR)
Read/Write

1B7	1B6	1B5	1B4	1B3	1B2	1B1	1B0
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Register 09 (89)
SCSI Output Control Latch (SOCL)
Read /Write

REQ	ACK	BSY	SEL	ATN	MSG	C/D	I/O
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 REQ(Assert SCSI REQ/ signal)
Bit 6 ACK(Assert SCSI ACK/ signal)
Bit 5 BSY(Assert SCSI BSY/ signal)
Bit 4 SEL(Assert SCSI SEL/ signal)
Bit 3 ATN(Assert SCSI ATN/ signal)
Bit 2 MSG(Assert SCSI MSG/ signal)
Bit 1 C/D(Assert SCSI C_D/ signal)
Bit 0 I/O(Assert SCSI I_O/ signal)

Register 0A (8A)
SCSI Selector ID (SSID)
Read Only

VAL	RES	RES	RES	RES	ENID2	ENID1	ENID0
7	6	5	4	3	2	1	0
Default>>>							
0	X	X	X	X	0	0	0

Bit 7 VAL (SCSI Valid Bit)
Bits 6-3 Reserved
Bits 2-0 Encoded Destination SCSI ID

Register 0B (8B)
SCSI Bus Control Lines (SBCL)
Read Only

REQ	ACK	BSY	SEL	ATN	MSG	C/D	I/O
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	X	X	X	X

Bit 7 REQ (SREQ/ status)
Bit 6 ACK (SACK/ status)
Bit 5 BSY (SBSY/ status)
Bit 4 SEL (SSEL/ status)
Bit 3 ATN SATN/ status)
Bit 2 MSG (SMSG/ status)
Bit 1 C/D (SC_D/ status)
Bit 0 I/O (SI_O/ status)

Register 0C (8C)
DMA Status (DSTAT)
Read Only

DFE	MDPE	BF	ABRT	SSI	SIR	RES	IID
7	6	5	4	3	2	1	0
Default>>>							
1	0	0	0	0	0	X	0

Bit 7 DFE (DMA FIFO empty)
Bit 6 MDPE (Master Data Parity Error)
Bit 5 BF (Bus fault)
Bit 4 ABRT (Aborted)
Bit 3 SSI (Single step interrupt)
Bit 2 SIR (SCRIPTS interrupt instruction received)
Bit 1 Reserved
Bit 0 IID (Illegal instruction detected)

Register 0D (8D)
SCSI Status Zero (SSTAT0)
Read Only

ILF	ORF	OLF	AIP	LOA	WOA	RST	SDP0/
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 ILF (SIDL full)
Bit 6 ORF (SODR full)
Bit 5 OLF (SODL full)
Bit 4 AIP (Arbitration in progress)
Bit 3 LOA (Lost arbitration)
Bit 2 WOA (Won arbitration)
Bit 1 RST/ (SCSI RST/ signal)
Bit 0 SDP/ (SCSI SDP/ parity signal)

Register 0E (8E)
SCSI Status One (SSTAT1)
Read Only

FF3	FF2	FF1	FF0	SDP0L	MSG	C/D	I/O
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	X	X	X	X

Bits 7-4 FF3-FF0 (FIFO flags)
Bit 3 SDPL (Latched SCSI parity)
Bit 2 MSG (SCSI MSG/ signal)
Bit 1 C/D (SCSI C_D/ signal)
Bit 0 I/O (SCSI I_O/ signal)

Register 0F (8F)
SCSI Status Two (SSTAT2)
(Read Only)

RES	RES	RES	RES	RES	RES	LDSC	RES
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	X	X	1	X

Bits 7-2 Reserved
Bit 1 LDSC (Last Disconnect)
Bit 0 Reserved

Registers 10-13 (90-93)
Data Structure Address (DSA)
Read/Write

Register 14 (94)
Interrupt Status (ISTAT)
(Read/Write)

ABRT	SRST	SIGP	SEM	CON	INTF	SIP	DIP
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 ABRT (Abort operation)
Bit 6 SRST (Software reset)
Bit 5 SIGP (Signal process)
Bit 4 SEM (Semaphore)
Bit 3 CON (Connected)
Bit 2 INTF (Interrupt on the Fly)
Bit 1 SIP (SCSI interrupt pending)
Bit 0 DIP (DMA interrupt pending)

Register 18 (98)
Chip Test Zero (CTEST0)
Read/Write

Register 19 (99)
Chip Test One (CTEST1)
Read Only

FMT3	FMT2	FMT1	FMT0	FFL3	FFL2	FFL1	FFL0
7	6	5	4	3	2	1	0
Default>>>							
1	1	1	1	0	0	0	0

Bits 7-4 FMT3-0 (Byte empty in DMA FIFO)
Bits 3-0 FFL3-0 (Byte full in DMA FIFO)

Register 1A (9A)
Chip Test Two (CTEST2)
Read Only

DDIR	SIGP	CIO	CM	RES	TEOP	DREQ	DACK
7	6	5	4	3	2	1	0
Default>>>							
0	0	X	X	0	0	0	1

Bit 7 DDIR (Data transfer direction)
Bit 6 SIGP (Signal process)
Bit 5 CIO (Configured as I/O)
Bit 4 CM (Configured as memory)
Bit 3 Reserved
Bit 2 TEOP (SCSI true end of process)
Bit 1 DREQ (Data request status)
Bit 0 DACK (Data acknowledge status)

Register 1B (9B)
Chip Test Three (CTEST3)
Read/Write

V3	V2	V1	V0	FLF	CLF	FM	WRIE
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	0	0	0	0

Bits 7-4 V3-V0 (Chip revision level)
Bit 3 FLF (Flush DMA FIFO)
Bit 2 CLF (Clear DMA FIFO)
Bit 1 FM (Fetch pin mode)
Bit 0 WRIE (Write and Invalidate Enable)

Registers 1C-1F (9C-9F)
Temporary (TEMP)
Read/Write

Register 20 (A0)
DMA FIFO (DFIFO)
Read/Write

RES	BO6	BO5	BO4	Bo3	BO2	BO1	BO0
7	6	5	4	3	2	1	0
Default>>>							
X	0	0	0	0	0	0	0

Bit 7 Reserved
Bits 6-0 BO6-BO0 (Byte offset counter)

Register 21 (A1)
Chip Test Four (CTEST4)
Read/Write

BDIS	ZMOD	ZSD	SRTM	MPEE	FBL2	FBL1	FBL0
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 BDIS (Burst Disable)
Bit 6 ZMOD (High impedance mode)
Bit 5 ZSD (SCSI Data High Impedance)
Bit 4 SRTM (Shadow Register Test Mode)
Bit 3 MPEE (Master Parity Error Enable)
Bits 2-0 FBL2-FBL0 (FIFO byte control)

Register 22 (A2)
Chip Test Five (CTEST5)
Read/Write

ADCK	BBCK	RES	MASR	DDIR	RES	RES	RES
7	6	5	4	3	2	1	0
Default>>>							
0	0	X	0	0	X	X	X

Bit 7 ADCK (Clock address incrementor)
Bit 6 BBCK (Clock byte counter)
Bit 5 Reserved
Bit 4 MASR (Master control for set or reset pulses)
Bit 3 DDIR (DMA direction)
Bits 2-0 Reserved

Register 23 (A3)
Chip Test Six (CTEST6)
Read/Write

DF7	DF6	DF5	DF4	DF3	DF2	DF1	DF0
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bits 7-0 DF7-DF0 (DMA FIFO)

Registers 24-26 (A4-A6)
DMA Byte Counter (DBC)
Read/Write

Register 27 (A7)
DMA Command (DCMD)
Read/Write

Registers 28-2B (A8-AB)
DMA Next Address (DNAD)
Read/Write

Registers 2C-2F (AC-AF)
DMA SCRIPTS Pointer (DSP)
Read/Write

Registers 30-33 (B0-B3)
DMA SCRIPTS Pointer Save (DSPS)
Read/Write

Registers 34-37 (B4-B7)
Scratch Register A (SCRATCH A)
Read/Write

Register 38 (B8)
DMA Mode (DMODE)
Read/Write

BL1	BL0	SIOM	DIOM	RESERL	RESERMP	BOF	MAN
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7-6 BL1-BL0 (Burst length)
Bit 5 SIOM (Source I/O-Memory Enable)
Bit 4 DIOM (Destination I/O-Memory Enable)
Bit 3 ERL (Enable Read Line)
Bit 2 ERMP (Enable Read Multiple)
Bit 1 BOF (Burst Op Code Fetch Enable)
Bit 0 MAN (Manual Start Mode)

Register 39 (B9)
DMA Interrupt Enable (DIEN)
Read/Write

RES	MDPE	BF	ABRT	SSI	SIR	RES	IID
7	6	5	4	3	2	1	0
Default>>>							
X	0	0	0	0	0	X	0

Bit 7 Reserved
Bit 6 MDPE (Master Data Parity Error)
Bit 5 BF (Bus fault)
Bit 4 ABRT (Aborted)
Bit 3 SSI (Single step interrupt)
Bit 2 SIR (SCRIPTS interrupt instruction received
Bit 1 Reserved
Bit 0 IID (Illegal instruction detected)

Register 3A (BA)
Scratch Byte Register (SBR)
Read/Write

Register 3B (BB)
DMA Control (DCNTL)
Read/Write

CLSE	PFF	PFEN	SSM	IRQM	STD	IRQD	COM
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 CLSE (Cache Line Size Enable)
Bit 6 PFF (Pre-Fetch Flush)
Bit 5 PFEN (Pre-fetch Enable)
Bit 4 SSM (Single-step mode)
Bit 3 IRQM (IRQ Mode)
Bit 2 STD (Start DMA operation)
Bit 1 IRQD (IRQ Disable)
Bit 0 COM (53C700 compatibility)

Register 3C-3F (BC-BF)
Adder Sum Output (ADDER)
Read Only

Register 40 (C0)
SCSI Interrupt Enable Zero (SIEN0)
Read/Write

M/A	CMP	SEL	RSL	SGE	UDC	RST	PAR
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 M/A (SCSI Phase Mismatch - Initiator Mode; SCSI ATN Condition - Target Mode)
Bit 6 CMP (Function Complete)
Bit 5 SEL (Selected)
Bit 4 RSL (Reselected)
Bit 3 SGE (SCSI Gross Error)
Bit 2 UDC (Unexpected Disconnect)
Bit 1 RST (SCSI Reset Condition)
Bit 0 PAR (SCSI Parity Error)

Register 41 (C1)
SCSI Interrupt Enable One (SIEN1)
Read/Write

RES	RES	RES	RES	RES	STO	GEN	HTH
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	X	0	0	0

Bits 7-3 Reserved
Bit 2 STO (Selection or Reselection Time-out)
Bit 1 GEN (General Purpose Timer Expired)
Bit 0 HTH (Handshake to Handshake timer Expired)

Register 42 (C2)
SCSI Interrupt Status Zero (SIST0)
Read Only

M/A	CMP	SEL	RSL	SGE	UDC	RST	PAR
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 M/A (Initiator Mode: Phase Mismatch; Target Mode: SATN/ Active)
Bit 6 CMP (Function Complete)
Bit 5 SEL (Selected)
Bit 4 RSL (Reselected)
Bit 3 SGE (SCSI Gross Error)
Bit 2 UDC (Unexpected Disconnect)
Bit 1 RST (SCSI RST/ Received)
Bit 0 PAR (Parity Error)

Register 43 (C3)
SCSI Interrupt Status One (SIST1)
Read Only

RES	RES	RES	RES	RES	STO	GEN	HTH
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	X	0	0	0

Bits 7-3 Reserved
Bit 2 STO (Selection or Reselection Time-out)
Bit 1 GEN (General Purpose Timer Expired)
Bit 0 HTH (Handshake-to-Handshake Timer Expired)

Register 44 (C4)
SCSI Longitudinal Parity (SLPAR)
Read/Write

Register 46 (C6)
Memory Access Control (MACNTL)
Read/Write

TYP3	TYP2	TYP1	TYP0	DWR	DRD	PSCPT	SCPTS
7	6	5	4	3	2	1	0
Default>>>							
0	1	0	1	0	0	0	0

Bits 7-4 TYP3-0 (Chip Type)
Bit 3 DWR (DataWR)
Bit 2 DRD (DataRD)
Bit 1 PSCPT (Pointer SCRIPTS)
Bit 0 SCPTS (SCRIPTS)

Register 47 (C7)
General Purpose Pin Control (GPCNTL)
Read/Write

ME	FE	RES	RES	RES	RES	GPIO1	GPIO0
7	6	5	4	3	2	1	0
Default>>>							
0	0	X	0	1	1	1	1

Bit 7 Master Enable
Bit 6 Fetch Enable
Bit 5 -2 Reserved
Bits 1-0 GPIO1_EN– GPIO0_EN (GPIO Enable)

Register 48 (C8)
SCSI Timer Zero (STIME0)
Read /Write

HTH	HTH	HTH	HRH	SEL	SEL	SEL	SEL
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bits 7-4 HTH (Handshake-to-Handshake Timer Period)
Bits 3-0 SEL (Selection Time-Out)

Register 49 (C9)
SCSI Timer One (STIME1)
Read/Write

RES	RES	RES	RES	GEN3	GEN2	GEN1	GEN0
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	0	0	0	0

Bits 7-4 Reserved
Bits 3-0 GEN3-0 (General Purpose Timer Period)

Register 4A (CA)
Response ID (RESPID)
Read/Write

Register 4C (CC)
SCSI Test Zero (STEST0)
Read Only

RES	SSAID2	SSAID1	SSAID0	SLT	ART	SOZ	SOM
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	0	X	1	1

Bit 7 Reserved
Bits 6-4 SSAID (SCSI Selected As ID)
Bit 3 SLT (Selection response logic test)
Bit 2 ART (Arbitration Priority Encoder Test)
Bit 1 SOZ (SCSI Synchronous Offset Zero)
Bit 0 SOM (SCSI Synchronous Offset Maximum)

Register 4D (CD)
SCSI Test One (STEST1)
Read/Write

SCLK	SISO	RES	RES	RES	RES	RES	RES
7	6	5	4	3	2	1	0
Default>>>							
0	0	X	X	X	X	X	X

Bit 7 SCLK
Bit 6 SISO (SCSI Isolation Mode)
Bits 5-0 Reserved

Register 4E (CE)
SCSI Test Two (STEST2)
Read/Write

SCE	ROF	RES	SLB	SZM	RES	EXT	LOW
7	6	5	4	3	2	1	0
Default>>>							
0	0	X	0	0	X	0	0

Bit 7 SCE (SCSI Control Enable)
Bit 6 ROF (Reset SCSI Offset)
Bit 5 Reserved
Bit 4 SLB (SCSI Loopback Mode)
Bit 3 SZM (SCSI High-Impedance Mode)
Bit 2 Reserved
Bit 1 EXT(Extend SREQ/SACK filtering)
Bit 0 LOW (SCSI Low level Mode)

Register 4F (CF)
SCSI Test Three (STEST3)
Read/Write

TE	STR	HSC	DSI	RES	TTM	CSF	STW
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	X	0	0	0

Bit 7 TE (TolerANT Enable)
Bit 6 STR (SCSI FIFO Test Read)
Bit 5 HSC (Halt SCSI Clock)
Bit 4 DSI (Disable Single Initiator Response)
Bit 3 Reserved
Bit 2 TTM (Timer Test Mode)
Bit 1 CSF (Clear SCSI FIFO)
Bit 0 STW (SCSI FIFO Test Write)

Register 50 (D0)
SCSI Input Data Latch (SIDL)
Read Only

Registers 54 (D4)
SCSI Output Data Latch (SODL)
Read/Write

Registers 58 (D8)
SCSI Bus Data Lines (SBDL)
Read Only

Registers 5C-5F (DC-DF)
Scratch Register B (SCRATCHB)
(Read/Write)

SYM53C875 Operating Registers

Register 00 (80)
SCSI Control Zero (SCNTL0)
Read/Write

ARB1	ARB0	START	WATN	EPC	RES	AAP	TRG
7	6	5	4	3	2	1	0
Default>>>							
1	1	0	0	0	X	0	0

Bit 7 ARB1 (Arbitration Mode bit 1)
Bit 6 ARB0 (Arbitration Mode bit 0)
Bit 5 START (Start Sequence)
Bit 4 WATN (Select with SATN/ on a Start Sequence)
Bit 3 EPC (Enable Parity Checking)
Bit 2 Reserved
Bit 1 AAP (Assert SATN/ on Parity Error)
Bit 0 TRG (Target Mode)

Register 01 (81)
SCSI Control One (SCNTL1)
Read/Write

EXC	ADB	DHP	CON	RST	AESP	IARB	SST
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 EXC (Extra Clock Cycle of Data Setup)
Bit 6 ADB (Assert SCSI Data Bus)
Bit 5 DHP (Disable Halt on Parity Error or ATN) (Target Only)
Bit 4 CON (Connected)
Bit 3 RST (Assert SCSI RST/ Signal)
Bit 2 AESP (Assert Even SCSI Parity (force bad parity))
Bit 1 IARB (Immediate Arbitration)
Bit 0 SST (Start SCSI Transfer)

Register 02 (82)
SCSI Control Two (SCNTL2)
Read/Write

SDU	CHM	SLPMD	SLPHBEN	WSS	VUE0	VUE1	WSR
7	6	5	4	3	2	1	0
Default>>>							
0	0	X0	X0	0	0X	0X	0

Bit 7 SDU (SCSI Disconnect Unexpected)
Bit 6 CHM (Chained Mode)
Bit 5 SLPMD (SLPAR Mode Bit)
Bit 4 SLPHBEN (SLPAR High Byte Enable)
Bit 3 WSS (Wide SCSI Send)
Bit 2 VUE0 (Vendor Unique Enhancements bit 0)
Bit 1 VUE1 (Vendor Unique Enhancements bit 1)
Bit 0 WSR (Wide SCSI Receive)

Register 03 (83)
SCSI Control Three (SCNTL3)
Read/Write

F20	SCF2	SCF1	SCF0	EWS	CCF2	CCF1	CCF0
7	6	5	4	3	2	1	0
Default>>>							
X0	0	0	0	0	0	0	0

Bit 7 F20 (Fast-20 Enable)
Bits 6-4 SCF2-0 (Synchronous Clock Conversion Factor)
Bit 3 EWS (Enable Wide SCSI)
Bits 2-0 CCF2-0 (Clock Conversion Factor)

Register 04 (84)
SCSI Chip ID (SCID)
Read/Write

RES	RRE	SRE	RES	ENC3	ENC2	ENC1	ENC0
7	6	5	4	3	2	1	0
Default>>>							
X	0	0	X	0	0	0	0

Bit 7 Reserved
Bit 6 RRE (Enable Response to Reselection)
Bit 5 SRE (Enable Response to Selection)
Bit 4 Reserved
Bits 3-0 Encoded Chip SCSI ID, bits 3-0

Register 05 (85)
SCSI Transfer (SXFER)
Read/Write

TP2	TP1	TP0	MO4	MO3	MO2	MO1	MO0
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	X0	0	0	0	0

Bits 7-5 TP2-0 (SCSI Synchronous Transfer Period)
Bits 3-0 MO3-MO0 (Max SCSI Synchronous Offset)

Register 06 (86)
SCSI Destination ID (SDID)
Read/Write

RES	RES	RES	RES	ENC3	ENC2	ENC1	ENC0
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	0	0	0	0

Bits 7-4 Reserved
Bits 3-0 Encoded Destination SCSI ID

Register 07 (87)
General Purpose (GPREG)
Read/Write

RES	RES	RES	GPIO4	GPIO3	GPIO2	GPIO1	GPIO0
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	0	X	X	X	X

Bits 7-5 Reserved
Bits 4-0 GPIO4-GPIO0 (General Purpose)

Register 08 (88)
SCSI First Byte Received (SFBR)
Read/Write

1B7	1B6	1B5	1B4	1B3	1B2	1B1	1B0
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Register 09 (89)
SCSI Output Control Latch (SOCL)
Read /Write

REQ	ACK	BSY	SEL	ATN	MSG	C/D	I/O
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 REQ(Assert SCSI REQ/ Signal)
Bit 6 ACK(Assert SCSI ACK/ Signal)
Bit 5 BSY(Assert SCSI BSY/ Signal)
Bit 4 SEL(Assert SCSI SEL/ Signal)
Bit 3 ATN(Assert SCSI ATN/ Signal)
Bit 2 MSG(Assert SCSI MSG/ Signal)
Bit 1 C/D(Assert SCSI C_D/ Signal)
Bit 0 I/O(Assert SCSI I_O/ Signal)

Register 0A (09)
SCSI Selector ID (SSID)
Read Only

VAL	RES	RES	RES	ENID3	ENID2	ENID1	ENID0
7	6	5	4	3	2	1	0
Default>>>							
0	X	X	X	0	0	0	0

Bit 7 VAL (SCSI Valid)
Bits 6-4 Reserved
Bits 3-0 Encoded Destination SCSI ID

Register 0B (8B)
SCSI Bus Control Lines (SBCL)
Read Only

REQ	ACK	BSY	SEL	ATN	MSG	C/D	I/O
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	X	X	X	X

Bit 7 REQ (SREQ/ Status)
Bit 6 ACK (SACK/ Status)
Bit 5 BSY (SBSY/ Status)
Bit 4 SEL (SSEL/ Status)
Bit 3 ATN (SATN/ Status)
Bit 2 MSG (SMSG/ Status)
Bit 1 C/D (SC_D/ Status)
Bit 0 I/O (SI_O/ Status)

Register 0C (8C)
DMA Status (DSTAT)
Read Only

DFE	MDPE	BF	ABRT	SSI	SIR	EBPI	IID
7	6	5	4	3	2	1	0
Default>>>							
1	0	0	0	0	0	X	0

Bit 7 DFE (DMA FIFO Empty)
Bit 6 MDPE (Master Data Parity Error)
Bit 5 BF (Bus Fault)
Bit 4 ABRT (Aborted)
Bit 3 SSI (Single Step Interrupt)
Bit 2 SIR (SCRIPTS Interrupt Instruction Received)
Bit 1 EBPI (Extended Byte Parity Error Interrupt) (53C875N only)
Bit 0 IID (Illegal Instruction Detected)

Register 0D (8D)
SCSI Status Zero (SSTAT0)
Read Only

ILF	ORF	OLF	AIP	LOA	WOA	RST	SDP0/
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 ILF (SIDL Least Significant Byte Full)
Bit 6 ORF (SODR Least Significant Byte Full)
Bit 5 OLF (SODL Least Significant Byte Full)
Bit 4 AIP (Arbitration in Progress)
Bit 3 LOA (Lost Arbitration)
Bit 2 WOA (Won Arbitration)
Bit 1 RST/ (SCSI RST/ Signal)
Bit 0 SDP0/ (SCSI SDP0/ Parity Signal)

Register 0E (8E)
SCSI Status One (SSTAT1)
Read Only

FF3	FF2	FF1	FF0	SDP0L	MSG	C/D	I/O
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	X	X	X	X

Bits 7-4 FF3-FF0 (FIFO Flags
Bit 3 SDP0L (Latched SCSI Parity)
Bit 2 MSG (SCSI MSG/ Signal)
Bit 1 C/D (SCSI C_D/ Signal)
Bit 0 I/O (SCSI I_O/ Signal)

Register 0F (8F)
SCSI Status Two (SSTAT2)
(Read Only)

ILF1	ORF1	OLF1	FF4	SPL1	RES	LDSC	SDP1
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	X	X	1	X

Bit 7 ILF1 (SIDL Most Significant Byte Full)
Bit 6 ORF1 (SODR Most Significant Byte Full)
Bit 5 OLF1 (SODL Most Significant Byte Full)
Bit 4 FF4 (FIFO Flags bit 4)
Bit 3 SPL1(Latched SCSI parity for SD15-8)
Bit 2 DIFFSENSE SENSE
Bit 1 LDSC (Last Disconnect)
Bit 0 SDP1 (SCSI SDP1 Signal)

Registers 10-13 (90-93)
Data Structure Address (DSA)
Read/Write

Register 14 (94)
Interrupt Status (ISTAT)
(Read/Write)

ABRT	SRST	SIGP	SEM	CON	INTF	SIP	DIP
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 ABRT (Abort Operation)
Bit 6 SRST (Software Reset)
Bit 5 SIGP (Signal Process)
Bit 4 SEM (Semaphore)
Bit 3 CON (Connected)
Bit 2 INTF (Interrupt on the Fly)
Bit 1 SIP (SCSI Interrupt Pending)
Bit 0 DIP (DMA Interrupt Pending)

Register 18 (98)
Chip Test Zero (CTEST0)
Read/Write

Register 19 (99)
Chip Test One (CTEST1)
Read Only

FMT3	FMT2	FMT1	FMT0	FFL3	FFL2	FFL1	FFL0
7	6	5	4	3	2	1	0
Default>>>							
1	1	1	1	0	0	0	0

Bits 7-4 FMT3-0 (Byte Empty in DMA FIFO)
Bits 3-0 FFL3-0 (Byte Full in DMA FIFO)

Register 1A (9A)
Chip Test Two (CTEST2)
Read/Write

DDIR	SIGP	CIO	CM	SRTCH	TEOP	DREQ	DACK
7	6	5	4	3	2	1	0
Default>>>							
0	0	X	X	0	0	0	1

Bit 7 DDIR (Data Transfer Direction)
Bit 6 SIGP (Signal Process)
Bit 5 CIO (Configured as I/O)
Bit 4 CM (Configured as Memory)
Bit 3 SRTCH (SCRATCHA/B Operation)
Bit 2 TEOP (SCSI True End of Process)
Bit 1 DREQ (Data Request Status)
Bit 0 DACK (Data Acknowledge Status)

Register 1B (9B)
Chip Test Three (CTEST3)
Read/Write

V3	V2	V1	V0	FLF	CLF	FM	WRIE
7	6	5	4	3	2	1	0
Default>>>							
x	x	x	x	0	0	0	X0

Bits 7-4 V3-V0 (Chip revision level)
Bit 3 FLF (Flush DMA FIFO)
Bit 2 CLF (Clear DMA FIFO)
Bit 1 FM (Fetch Pin Mode)
Bit 0 WRIE (Write and Invalidate Enable)

Registers 1C-1F (9C-9F)
Temporary (TEMP)
Read/Write

Register 20 (A0)
DMA FIFO (DFIFO)
Read/Write

BO7	BO6	BO5	BO4	Bo3	BO2	BO1	BO0
7	6	5	4	3	2	1	0
Default>>>							
X	0	0	0	0	0	0	0

Bits 7-0 BO7-BO0 (Byte offset counter)

Register 21 (A1)
Chip Test Four (CTEST4)
Read/Write

BDIS	ZMOD	ZSD	SRTM	MPEE	FBL2	FBL1	FBL0
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 BDIS (Burst Disable)
Bit 6 ZMOD (High Impedance Mode)
Bit 5 ZSD (SCSI Data High Impedance)
Bit 4 SRTM (Shadow Register Test Mode)
Bit 3 MPEE (Master Parity Error Enable)
Bits 2-0 FBL2-FBL0 (FIFO Byte Control)

Register 22 (A2)
Chip Test Five (CTEST5)
Read/Write

ADCK	BBCK	DFS	MASR	DDIR	BL2	BO9	BO8
7	6	5	4	3	2	1	0
Default>>>							
0	0	0X	0	0	X	X	X

Bit 7 ADCK (Clock Address Incrementor)
Bit 6 BBCK (Clock Byte Counter)
Bit 5 DFS (DMA FIFO Size)
Bit 4 MASR (Master Control for Set or Reset Pulses)
Bit 3 DDIR (DMA Direction)
Bit 2 BL2 (Burst Length bit 2)
Bits 1-0 BO9-8

Register 23 (A3)
Chip Test Six (CTEST6)
Read/Write

DF7	DF6	DF5	DF4	DF3	DF2	DF1	DF0
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bits 7-0 DF7-DF0 (DMA FIFO)

Registers 24-26 (A4-A6)
DMA Byte Counter (DBC)
Read/Write

Register 27 (A7)
DMA Command (DCMD)
Read/Write

Registers 28-2B (A8-AB)
DMA Next Address (DNAD)
Read/Write

Registers 2C-2F (AC-AF)
DMA SCRIPTS Pointer (DSP)
Read/Write

Registers 30-33 (B0-B3)
DMA SCRIPTS Pointer Save (DSPS)
Read/Write

Registers 34-37 (B4-B7)
Scratch Register A (SCRATCH A)
Read/Write

Register 38 (B8)
DMA Mode (DMODE)
Read/Write

BL1	BL0	SIOM	DIOM	ER	ERMP	BOF	MAN
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	X0	0	0

Bit 7-6 BL1-BL0 (Burst Length)
Bit 5 SIOM (Source I/O-Memory Enable)
Bit 4 DIOM (Destination I/O-Memory Enable)
Bit 3 ERL (Enable Read Line)
Bit 2 ERMP (Enable Read Multiple)
Bit 1 BOF (Burst Op Code Fetch Enable)
Bit 0 MAN (Manual Start Mode)

Register 39 (B9)
DMA Interrupt Enable (DIEN)
Read/Write

RES	MDPE	BF	ABRT	SSI	SIR	EBPE	IID
7	6	5	4	3	2	1	0
Default>>>							
X	0	0	0	0	0	X	0

Bit 7 Reserved
Bit 6 MDPE (Master Data Parity Error)
Bit 5 BF (Bus Fault)
Bit 4 ABRT (Aborted)
Bit 3 SSI (Single -step Interrupt)
Bit 2 SIR (SCRIPTS Interrupt Instruction Received
Bit 1 EBPE (Extended Byte Parity Enable) (SYM53C875N only)
Bit 0 IID (Illegal Instruction Detected)

Register 3A (BA)
Scratch Byte Register (SBR)
Read/Write

Register 3B (BB)
DMA Control (DCNTL)
Read/Write

CLSE	PFF	PFEN	SSM	IRQM	STD	IRQD	COM
7	6	5	4	3	2	1	0
Default>>>							
X0	X0	X0	0	0	0	X0	0

Bit 7 CLSE (Cache Line Size Enable)
Bit 6 PFF (Pre-fetch Flush)
Bit 5 PFEN (Pre-fetch Enable)
Bit 4 SSM (Single-step Mode)
Bit 3 IRQM (IRQ Mode)
Bit 2 STD (Start DMA Operation)
Bit 1 IRQD (IRQ Disable)
Bit 0 COM (53C700 Compatibility)

Register 3C-3F (BC-BF)
Adder Sum Output (ADDER)
Read Only

Register 40 (C0)
SCSI Interrupt Enable Zero (SIEN0)
Read/Write

M/A	CMP	SEL	RSL	SGE	UDC	RST	PAR
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 M/A (SCSI Phase Mismatch - Initiator Mode; SCSI ATN Condition - Target Mode)
Bit 6 CMP (Function Complete)
Bit 5 SEL (Selected)
Bit 4 RSL (Reselected)
Bit 3 SGE (SCSI Gross Error)
Bit 2 UDC (Unexpected Disconnect)
Bit 1 RST (SCSI Reset Condition)
Bit 0 PAR (SCSI Parity Error)

Register 41 (C1)
SCSI Interrupt Enable One (SIEN1)
Read/Write

RES 7	RES 6	RES 5	RES 4	RES 3	STO 2	GEN 1	HTH 0
Default>>>							
X	X	X	X	X	0	0	0

Bits 7-3 Reserved
Bit 2 STO (Selection or Reselection Time-out)
Bit 1 GEN (General Purpose Timer Expired)
Bit 0 HTH (Handshake-to-Handshake Timer Expired)

Register 42 (C2)
SCSI Interrupt Status Zero (SIST0)
Read Only

M/A 7	CMP 6	SEL 5	RSL 4	SGE 3	UDC 2	RST 1	PAR 0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 M/A (Initiator Mode: Phase Mismatch; Target Mode: SATN/ Active)
Bit 6 CMP (Function Complete)
Bit 5 SEL (Selected)
Bit 4 RSL (Reselected)
Bit 3 SGE (SCSI Gross Error)
Bit 2 UDC (Unexpected Disconnect)
Bit 1 RST (SCSI RST/ Received)
Bit 0 PAR (Parity Error)

Register 43 (C3)
SCSI Interrupt Status One (SIST1)
Read Only

RES 7	RES 6	RES 5	RES 4	RES 3	STO 2	GEN 1	HTH 0
Default>>>							
X	X	X	X	X	0	0	0

Bits 7-3 Reserved
Bit 2 STO (Selection or Reselection Time-out)
Bit 1 GEN (General Purpose Timer Expired)
Bit 0 HTH (Handshake-to-Handshake Timer Expired)

Register 44 (C4)
SCSI Longitudinal Parity (SLPAR)
Read/Write

Register 45 (C5)
SCSI Wide Residue (SWIDE)
Read/Write

Register 46 (C6)
Memory Access Control (MACNTL)
Read/Write

TYP3 7	TYP2 6	TYP1 5	TYP0 4	DWR 3	DRD 2	PSCPT 1	SCPTS 0
Default>>>							
0	01	1	10	0	0	0	0

Bits 7-4 TYP3-0 (Chip Type)
Bit 3 DWR (DataWR)
Bit 2 DRD (DataRD)
Bit 1 PSCPT (Pointer SCRIPTS)
Bit 0 SCPTS (SCRIPTS)

Register 47 (C7)
General Purpose Pin Control (GPCNTL)
Read/Write

ME 7	FE 6	RES 5	GPIO4 4	GPIO3 3	GPIO2 2	GPIO1 1	GPIO0 0
Default>>>							
0	0	X	0	1	1	1	1

Bit 7 Master Enable
Bit 6 Fetch Enable
Bit 5 Reserved
Bits 4-2 GPIO4_EN–GPIO2_EN (GPIO Enable)
Bits 1-0 GPIO1_EN– GPIO0_EN (GPIO Enable)

Register 48 (C8)
SCSI Timer Zero (STIME0)
Read /Write

HTH 7	HTH 6	HTH 5	HRH 4	SEL 3	SEL 2	SEL 1	SEL 0
Default>>>							
0	0	0	0	0	0	0	0

Bits 7-4 HTH (Handshake-to-Handshake Timer Period)
Bits 3-0 SEL (Selection Time-Out)

Register 49 (C9)
SCSI Timer One (STIME1)
Read/Write

RES 7	HTHBA 6	GENSF 5	HTHSF 4	GEN3 3	GEN2 2	GEN1 1	GEN0 0
Default>>>							
X	0X	0X	0X	0	0	0	0

Bit 7 Reserved
Bit 6 HTHBA (Handshake-to-Handshake Timer Bus Activity Enable)
Bit 5 GENSF (General Purpose Timer Scale Factor)
Bit 4 HTHSF (Handshake to Handshake Timer Scale Factor)
Bits 3-0 GEN3-0 (General Purpose Timer Period)

Register 4A (CA)
Response ID Zero (RESPID0)
Read/Write

Register 4B (CB)
Response ID One(RESPID1)
Read/Write

Register 4C (CC)
SCSI Test Zero (STEST0)
Read Only

SSAID3 7	SSAID2 6	SSAID1 5	SSAID0 4	SLT 3	ART 2	SOZ 1	SOM 0
Default>>>							
0X	0X	0X	0X	0	X	1	1

Bits 7-4 SSAID (SCSI Selected As ID)
Bit 3 SLT (Selection Response Logic Test)
Bit 2 ART (Arbitration Priority Encoder Test)
Bit 1 SOZ (SCSI Synchronous Offset Zero)
Bit 0 SOM (SCSI Synchronous Offset Maximum)

Register 4D (CD)
SCSI Test One (STEST1)
Read/Write

SCLK	SISO	RES	RES	RES	RES	RES	RES
7	6	5	4	3	2	1	0
Default>>>							
0	X0	X	X	X	X	X	X

Bit 7 SCLK
Bit 6 SISO (SCSI Isolation Mode)
Bits 5-4 Reserved
Bit 3 SCLK Doubler Enable (DBLEN)
Bit 2 SCLK Doubler Select (DBLSEL)
Bits1-0 Reserved

Register 4E (CE)
SCSI Test Two (STEST2)
Read/Write

SCE	ROF	DIF	SLB	SZM	AWS	EXT	LOW
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 SCE (SCSI Control Enable)
Bit 6 ROF (Reset SCSI Offset)
Bit 5 DIF (SCSI Differential Mode)
Bit 4 SLB (SCSI Loopback Mode)
Bit 3 SZM (SCSI High-Impedance Mode)
Bit 1 EXT (Extend SREQ/SACK Filtering)
Bit 0 LOW (SCSI Low level Mode)

Register 4F (CF)
SCSI Test Three (STEST3)
Read/Write

TE	STR	HSC	DSI	S16	TTM	CSF	STW
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 TE (TolerANT Enable)
Bit 6 STR (SCSI FIFO Test Read)
Bit 5 HSC (Halt SCSI Clock)
Bit 4 DSI (Disable Single Initiator Response)
Bit 3 S16 (16-bit System)
Bit 2 TTM (Timer Test Mode)
Bit 1 CSF (Clear SCSI FIFO)
Bit 0 STW (SCSI FIFO Test Write)

Register 50-51 (D0-D1)
SCSI Input Data Latch (SIDL)
Read Only

Registers 54-55 (D4-D5)
SCSI Output Data Latch (SODL)
Read/Write

Registers 58-59 (D8-D9)
SCSI Bus Data Lines (SBDL)
Read Only

Registers 5C-5F (DC-DF)
Scratch Register B (SCRATCHB)
(Read/Write)

Registers 60h-7Fh (E0h-FFh)
Scratch Registers C-J
(SCRATCHC-SCRATCHJ)
Read/Write

SYM53C876 Operating Registers

Register 00h
SCSI Control Zero (SCNTL0)
Read/Write

ARB1	ARB0	START	WATN	EPC	RES	AAP	TRG
7	6	5	4	3	2	1	0
Default >>>							
1	1	0	0	0	X	0	0

Bit 7 ARB1 (Arbitration Mode bit 1)
Bit 6 ARB0 (Arbitration Mode bit 0)
Bit 5 START (Start Sequence)
Bit 4 WATN (Select with SATN/ on a Start Sequence)
Bit 3 EPC (Enable Parity Checking)
Bit 2 Reserved
Bit 1 AAP (Assert SATN/ on Parity Error)
Bit 0 TRG (Target Mode)

Register 01h
SCSI Control One (SCNTL1)
Read/Write

EXC	ADB	DHP	CON	RST	AESP	IARB	SST
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bit 7 EXC (Extra Clock Cycle of Data Setup)
Bit 6 ADB (Assert SCSI Data Bus)
Bit 5 DHP (Disable Halt on Parity Error or ATN) (Target Only)
Bit 4 CON (Connected)
Bit 3 RST (Assert SCSI RST/ Signal)
Bit 2 AESP (Assert Even SCSI Parity (force bad parity))
Bit 1 IARB (Immediate Arbitration)
Bit 0 SST (Start SCSI Transfer)

Register 02h
SCSI Control Two (SCNTL2)
Read/Write

SDU	CHM	SLPMD	SLPH-BEN	WSS	VUE0	VUE1	WSR
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	X	0

Bit 7 SDU (SCSI Disconnect Unexpected)
Bit 6 CHM (Chained Mode)
Bit 5 SLPMD (SLPAR Mode Bit)
Bit 4 SLPHBEN (SLPAR High Byte Enable)
Bit 3 WSS (Wide SCSI Send)
Bit 2 VUE0 (Vendor Unique Enhancement bit 0)
Bit 1 VUE1 (Vendor Unique Enhancement bit 1)
Bit 0 WSR (Wide SCSI Receive)

Register 03h
SCSI Control Three (SCNTL3)
Read/Write

USE	SCF2	SCF1	SCF0	EWS	CCF2	CCF1	CCF0
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bit 7 USE (Ultra SCSI Enable)
Bits 6-4 SCF2-0 (Synchronous Clock Conversion Factor)
Bit 3 EWS (Enable Wide SCSI)
Bits 2-0 CCF2-0 (Clock Conversion Factor)

Register 04h
SCSI Chip ID (SCID)
Read/Write

RES	RRE	SRE	RES	ENC3	ENC2	ENC1	ENC0
7	6	5	4	3	2	1	0
Default >>>							
X	0	0	X	0	0	0	0

Bit 7 Reserved
Bit 6 RRE (Enable Response to Reselection)
Bit 5 SRE (Enable Response to Selection)
Bit 4 Reserved
Bits 3-0 Encoded Chip SCSI ID, bits 3-0

Register 05h
SCSI Transfer (SXFER)
Read/Write

TP2	TP1	TP0	MO4	MO3	MO2	MO1	MO0
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	X	0	0	0	0

Bits 7-5 TP2-0 (SCSI Synchronous Transfer Period)
Bits 4-0 MO4-MO0 (Max SCSI Synchronous Offset)

Register 06h
SCSI Destination ID (SDID)
Read/Write

RES	RES	RES	RES	ENC3	ENC2	ENC1	ENC0
7	6	5	4	3	2	1	0
Default >>>							
X	X	X	X	0	0	0	0

Bits 7-4 Reserved
Bits 3-0 Encoded Destination SCSI ID

Register 07h
General Purpose (GPREG)
Read/Write

RES	RES	RES	GPIO4	GPIO3	GPIO2	GPIO1	GPIO0
7	6	5	4	3	2	1	0
Default >>>							
X	X	X	0	X	X	X	X

Bits 7-5, 3 Reserved
Bits 4, 2-0 GPIO4-GPIO0 (General Purpose)

Register 08h
SCSI First Byte Received (SFBR)
Read/Write

1B7	1B6	1B5	1B4	1B3	1B2	1B1	1B0
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Register 09h
SCSI Output Control Latch (SOCL)
Read/Write

REQ	ACK	BSY	SEL	ATN	MSG	C/D	I/O
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bit 7 REQ(Assert SCSI REQ/ Signal)
Bit 6 ACK(Assert SCSI ACK/ Signal)
Bit 5 BSY(Assert SCSI BSY/ Signal)
Bit 4 SEL(Assert SCSI SEL/ Signal)
Bit 3 ATN(Assert SCSI ATN/ Signal)
Bit 2 MSG(Assert SCSI MSG/ Signal)
Bit 1 C/D(Assert SCSI C_D/ Signal)
Bit 0 I/O(Assert SCSI I_O/ Signal)

Register 0Ah
SCSI Selector ID (SSID)
Read Only

VAL	RES	RES	RES	ENID3	ENID2	ENID1	ENID0
7	6	5	4	3	2	1	0
Default >>>							
0	X	X	X	0	0	0	0

Bit 7 VAL (SCSI Valid)
Bits 6-4 Reserved
Bits 3-0 Encoded Destination SCSI ID

Register 0Bh
SCSI Bus Control Lines (SBCL)
Read Only

REQ	ACK	BSY	SEL	ATN	MSG	C/D	I/O
7	6	5	4	3	2	1	0
Default >>>							
X	X	X	X	X	X	X	X

Bit 7 REQ (SREQ/ Status)
Bit 6 ACK (SACK/ Status)
Bit 5 BSY (SBSY/ Status)
Bit 4 SEL (SSEL/ Status)
Bit 3 ATN (SATN/ Status)
Bit 2 MSG (SMSG/ Status)
Bit 1 C/D (SC_D/ Status)
Bit 0 I/O (SI_O/ Status)

Register 0Ch
DMA Status (DSTAT)
Read Only

DFE	MDPE	BF	ABRT	SSI	SIR	RES	IID
7	6	5	4	3	2	1	0
Default >>>							
1	0	0	0	0	0	X	0

Bit 7 DFE (DMA FIFO Empty)
Bit 6 MDPE (Master Data Parity Error)
Bit 5 BF (Bus Fault)
Bit 4 ABRT (Aborted)
Bit 3 SSI (Single Step Interrupt)
Bit 2 SIR (SCRIPTS Interrupt Instruction Received)
Bit 1 Reserved
Bit 0 IID (Illegal Instruction Detected)

Register 0Dh
SCSI Status Zero (SSTAT0)
Read Only

ILF	ORF	OLF	AIP	LOA	WOA	RST/	SDP0/
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bit 7 ILF (SIDL Least Significant Byte Full)
Bit 6 ORF (SODR Least Significant Byte Full)
Bit 5 OLF (SODL Least Significant Byte Full)
Bit 4 AIP (Arbitration in Progress)
Bit 3 LOA (Lost Arbitration)
Bit 2 WOA (Won Arbitration)
Bit 1 RST/ (SCSI RST/ Signal)
Bit 0 SDP0/ (SCSI SDP0/ Parity Signal)

Register 0Eh
SCSI Status One (SSTAT1)
Read Only

FF3	FF2	FF1	FF0	SDP0L	MSG	C/D	I/O
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	X	X	X	X

Bits 7-4 FF3-FF0 (FIFO Flags)
Bit 3 SDP0L (Latched SCSI Parity)
Bit 2 MSG (SCSI MSG/ Signal)
Bit 1 C/D (SCSI C_D/ Signal)
Bit 0 I/O (SCSI I_O/ Signal)

Register 0Fh
SCSI Status Two (SSTAT2)
Read Only

ILF1	ORF1	OLF1	FF4	SPL1	DIFF	LDSC	SDP1
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	X	X	1	X

Bit 7 ILF1 (SIDL Most Significant Byte Full)
Bit 6 ORF1 (SODR Most Significant Byte Full)
Bit 5 OLF1 (SODL Most Significant Byte Full)
Bit 4 FF4 (FIFO Flags bit 4)
Bit 3 SPL1(Latched SCSI parity for SD15-8)
Bit 2 DIFFSENSE Sense
Bit 1 LDSC (Last Disconnect)
Bit 0 SDP1 (SCSI SDP1 Signal)

Registers 10h-13h
Data Structure Address (DSA)
Read/Write

Register 14h
Interrupt Status (ISTAT)
Read/Write

ABRT	SRST	SIGP	SEM	CON	INTF	SIP	DIP
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bit 7 ABRT (Abort Operation)
Bit 6 SRST (Software Reset)
Bit 5 SIGP (Signal Process)
Bit 4 SEM (Semaphore)
Bit 3 CON (Connected)
Bit 2 INTF (Interrupt on the Fly)
Bit 1 SIP (SCSI Interrupt Pending)
Bit 0 DIP (DMA Interrupt Pending)

Register 19h
Chip Test One (CTEST1)
Read Only

FMT3	FMT2	FMT1	FMT0	FFL3	FFL2	FFL1	FFL0
7	6	5	4	3	2	1	0
Default >>>							
1	1	1	1	0	0	0	0

Bits 7-4 FMT3-0 (Byte Empty in DMA FIFO)
Bits 3-0 FFL3-0 (Byte Full in DMA FIFO)

Register 1Ah
Chip Test Two (CTEST2)
Read Only

DDIR	SIGP	CIO	CM	SRTCH	TEOP	DREQ	DACK
7	6	5	4	3	2	1	0
Default >>>							
0	0	X	X	0	0	0	1

Bit 7 DDIR (Data Transfer Direction)
Bit 6 SIGP (Signal Process)
Bit 5 CIO (Configured as I/O)
Bit 4 CM (Configured as Memory)
Bit 3 SRTCH (SCRATCHA/B Operation)
Bit 2 TEOP (SCSI True End of Process)
Bit 1 DREQ (Data Request Status)
Bit 0 DACK (Data Acknowledge Status)

Register 1Bh
Chip Test Three (CTEST3)
Read/Write

V3	V2	V1	V0	FLF	CLF	FM	WRIE
7	6	5	4	3	2	1	0
Default >>>							
X	X	X	X	0	0	0	0

Bits 7-4 V3-V0 (Chip Revision Level)
Bit 3 FLF (Flush DMA FIFO)
Bit 2 CLF (Clear DMA FIFO)
Bit 1 FM (Fetch Pin Mode)
Bit 0 WRIE (Write and Invalidate Enable)

Registers 1Ch-1Fh
Temporary (TEMP)
Read/Write

Register 20h
DMA FIFO (DFIFO)
Read/Write

BO7	BO6	BO5	BO4	B03	BO2	BO1	BO0
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bits 7-0 BO7-BO0 (Byte offset counter)

Register 21h
Chip Test Four (CTEST4)
Read/Write

BDIS	ZMOD	ZSD	SRTM	MPEE	FBL2	FBL1	FBL0
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bit 7 BDIS (Burst Disable)
Bit 6 ZMOD (High Impedance Mode)
Bit 5 ZSD (SCSI Data High Impedance)
Bit 4 SRTM (Shadow Register Test Mode)
Bit 3 MPEE (Master Parity Error Enable)
Bits 2-0 FBL2-FBL0 (FIFO Byte Control)

Register 22h
Chip Test Five (CTEST5)
Read/Write

ADCK	BBCK	DFS	MASR	DDIR	BL2	BO9	BO8
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bit 7 ADCK (Clock Address Incrementor)
Bit 6 BBCK (Clock Byte Counter)
Bit 5 DFS (DMA FIFO Size)
Bit 4 MASR (Master Control for Set or Reset Pulses)
Bit 3 DDIR (DMA Direction)
Bit 2 BL2 (Burst Length bit 2)
Bits 1-0 BO9-BO8 (DMA FIFO Byte Offset Counter, bits 9-8)

Register 23h
Chip Test Six (CTEST6)
Read/Write

DF7	DF6	DF5	DF4	DF3	DF2	DF1	DF0
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bits 7-0 DF7-DF0 (DMA FIFO)

Registers 24h-26h
DMA Byte Counter (DBC)
Read/Write

Register 27h
DMA Command (DCMD)
Read/Write

Registers 28h-2Bh
DMA Next Address (DNAD)
Read/Write

Registers 2Ch-2Fh
DMA SCRIPTS Pointer (DSP)
Read/Write

Registers 30h-33h
DMA SCRIPTS Pointer Save (DSPS)
Read/Write

Registers 34h
Scratch Register A (SCRATCHA)
Read/Write

Register 38h
DMA Mode (DMODE)
Read/Write

BL1	BL0	SIOM	DIOM	ER	ERMP	BOF	MAN
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bit 7-6 BL1-BL0 (Burst Length)
Bit 5 SIOM (Source I/O-Memory Enable)
Bit 4 DIOM (Destination I/O-Memory Enable)
Bit 3 ERL (Enable Read Line)
Bit 2 ERMP (Enable Read Multiple)
Bit 1 BOF (Burst Op Code Fetch Enable)
Bit 0 MAN (Manual Start Mode)

Register 39h
DMA Interrupt Enable (DIEN)
Read/Write

RES	MDPE	BF	ABRT	SSI	SIR	RES	IID
7	6	5	4	3	2	1	0
Default >>>							
X	0	0	0	0	0	X	0

Bit 7 Reserved
Bit 6 MDPE (Master Data Parity Error)
Bit 5 BF (Bus Fault)
Bit 4 ABRT (Aborted)
Bit 3 SSI (Single -step Interrupt)
Bit 2 SIR (SCRIPTS Interrupt Instruction Received
Bit 1 Reserved
Bit 0 IID (Illegal Instruction Detected)

Register 3Ah
Scratch Byte Register (SBR)
Read/Write

Register 3Bh
DMA Control (DCNTL)
Read/Write

CLSE	PFF	PFEN	SSM	INTM	STD	INTD	COM
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bit 7 CLSE (Cache Line Size Enable)
Bit 6 PFF (Pre-fetch Flush)
Bit 5 PFEN (Pre-fetch Enable)
Bit 4 SSM (Single-step Mode)
Bit 3 INTM (INTA Mode)
Bit 1 IRQD (INTA, INTB Disable)
Bit 0 COM (53C700 Compatibility)

Register 3Ch-3Fh
Adder Sum Output (ADDER)
Read Only

Register 40h
SCSI Interrupt Enable Zero (SIEN0)
Read/Write

M/A	CMP	SEL	RSL	SGE	UDC	RST	PAR
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bit 7 M/A (SCSI Phase Mismatch - Initiator Mode; SCSI ATN Condition - Target Mode)
Bit 6 CMP (Function Complete)
Bit 5 SEL (Selected)
Bit 4 RSL (Reselected)
Bit 3 SGE (SCSI Gross Error)
Bit 2 UDC (Unexpected Disconnect)
Bit 1 RST (SCSI Reset Condition)
Bit 0 PAR (SCSI Parity Error)

Register 41h
SCSI Interrupt Enable One (SIEN1)
Read/Write

RES	RES	RES	RES	RES	STO	GEN	HTH
7	6	5	4	3	2	1	0
Default >>>							
X	X	X	X	X	0	0	0

Bits 7-3 Reserved
Bit 2 STO (Selection or Reselection Time-out)
Bit 1 GEN (General Purpose Timer Expired)
Bit 0 HTH (Handshake-to-Handshake Timer Expired)

Register 42h
SCSI Interrupt Status Zero (SIST0)
Read Only

M/A	CMP	SEL	RSL	SGE	UDC	RST	PAR
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bit 7 M/A (Initiator Mode: Phase Mismatch; Target Mode: SATN/ Active)
Bit 6 CMP (Function Complete)
Bit 5 SEL (Selected)
Bit 4 RSL (Reselected)
Bit 3 SGE (SCSI Gross Error)
Bit 2 UDC (Unexpected Disconnect)
Bit 1 RST (SCSI RST/ Received)
Bit 0 PAR (Parity Error)

Register 43h
SCSI Interrupt Status One (SIST1)
Read Only

RES	RES	RES	RES	RES	STO	GEN	HTH
7	6	5	4	3	2	1	0
Default >>>							
X	X	X	X	0	0	0	0

Bits 7-4 Reserved
Bit 2 STO (Selection or Reselection Time-out)
Bit 1 GEN (General Purpose Timer Expired)
Bit 0 HTH (Handshake-to-Handshake Timer Expired)

Register 44h
SCSI Longitudinal Parity (SLPAR)
Read/Write

Register 45h
SCSI Wide Residue (SWIDE)
Read/Write

Register 46h
Memory Access Control (MACNTL)
Read/Write

TYP3	TYP2	TYP1	TYP0	RES	RES	RES	RES
7	6	5	4	3	2	1	0
Default >>>							
0	1	1	1	X	X	X	X

Bits 7-4 TYP3-0 (Chip Type)
Bit 3-0 Reserved

Register 47h
General Purpose Pin Control (GPCNTL)
Read/Write

ME	FE	RES	GPIO4	GPIO3	GPIO2	GPIO1	GPIO0
7	6	5	4	3	2	1	0
Default >>>							
0	0	X	0	1	1	1	1

Bit 7 Master Enable
Bit 6 Fetch Enable
Bits 5 Reserved
Bits 4, 2 GPIO4_EN-GPIO2_EN (GPIO Enable)
Bits 1-0 GPIO1_EN-GPIO0_EN (GPIO Enable)

Register 48h
SCSI Timer Zero (STIME0)
Read/Write

HTH7	HTH6	HTH5	HRH4	SEL	SEL	SEL	SEL
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bits 7-4 HTH (Handshake-to-Handshake Timer Period)
Bits 3-0 SEL (Selection Time-Out)

Register 49h
SCSI Timer One (STIME1)
Read/Write

RES	HTHBA	GENSF	HTHSF	GEN3	GEN2	GEN1	GEN0
7	6	5	4	3	2	1	0
Default >>>							
X	0	0	0	0	0	0	0

Bit 7 **Reserved**
Bit 6 **HTHBA (Handshake-to-Handshake Timer Bus Activity Enable)**
Bit 5 **GENSF (General Purpose Timer Scale Factor)**
Bit 4 **HTHSF (Handshake to Handshake Timer Scale Factor)**
Bits 3-0 **GEN3-0 (General Purpose Timer Period)**

Register 4Ah
Response ID Zero (RESPID0)
Read/Write

ID	ID	ID	ID	ID	ID	ID	ID
7	6	5	4	3	2	1	0
Default >>>							
X	X	X	X	X	X	X	X

Register 4Bh
Response ID One (RESPID1)
Read/Write

ID	ID	ID	ID	ID	ID	ID	ID
15	14	13	12	11	10	9	8
Default >>>							
X	X	X	X	X	X	X	X

Register 4Ch
SCSI Test Zero (STEST0)
Read Only

SSAID3	SSAID2	SSAID1	SSAID0	SLT	ART	SOZ	SOM
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	X	1	1

Bits 7-4 **SSAID3-0 (SCSI Selected As ID)**
Bit 3 **SLT (Selection Response Logic Test)**
Bit 2 **ART (Arbitration Priority Encoder Test)**
Bit 1 **SOZ (SCSI Synchronous Offset Zero)**
Bit 0 **SOM (SCSI Synchronous Offset Maximum)**

Register 4Dh
SCSI Test One (STEST1)
Read/Write

SCLK	ISO	RES	RES	DBLEN	DBLSEL	RES	RES
7	6	5	4	3	2	1	0
Default >>>							
0	0	X	X	0	0	X	X

Bit 7 **SCLK**
Bit 6 **ISO_MODE (SCSI Isolation Mode)**
Bit 5 **Reserved**
Bit 4 **Reserved**
Bit 3 **DBLEN (SCLK Doubler Enable)**
Bit 2 **DBLSEL (SCLK Doubler Select)**
Bits 1-0 **Reserved**

Register 4Eh
SCSI Test Two (STEST2)
Read/Write

SCE	ROF	DIF	SLB	SZM	AWS	EXT	LOW
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bit 7 **SCE (SCSI Control Enable)**
Bit 6 **ROF (Reset SCSI Offset)**
Bit 5 **DIF**
Bit 4 **SLB (SCSI Loopback Mode)**
Bit 3 **SZM (SCSI High-Impedance Mode)**
Bit 2 **AWS (Always Wide SCSI)**
Bit 1 **EXT (Extend SREQ/SACK Filtering)**
Bit 0 **LOW (SCSI Low level Mode)**

Register 4Fh
SCSI Test Three (STEST3)
Read/Write

TE	STR	HSC	DSI	CHECKHI	TTM	CSF	STW
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	X	0	0	0

Bit 7 **TE (TolerANT Enable)**
Bit 6 **STR (SCSI FIFO Test Read)**
Bit 5 **HSC (Halt SCSI Clock)**
Bit 4 **DSI (Disable Single Initiator Response)**
Bit 3 **CHECKHI (Check High Parity)**
Bit 2 **TTM (Timer Test Mode)**
Bit 1 **CSF (Clear SCSI FIFO)**
Bit 0 **STW (SCSI FIFO Test Write)**

Register 50h-51h
SCSI Input Data Latch (SIDL)
Read Only

Registers 54h-55h
SCSI Output Data Latch (SODL)
Read/Write

Registers 58h-59h
SCSI Bus Data Lines (SBDL)
Read Only

Registers 5Ch-5Fh
Scratch Register B (SCRATCHB)
Read/Write

Registers 60h-7Fh
Scratch Registers C-J
(SCRATCHC-SCRATCHJ)
Read/Write

SYM53C885 SCSI Register Summary

Register 00h
SCSI Control Zero (SCNTL0)
Read/Write

ARB1	ARB0	START	WATN	EPC	RES	AAP	TRG
7	6	5	4	3	2	1	0
Default >>>							
1	1	0	0	0	X	0	0

Bit 7 ARB1 (Arbitration Mode bit 1)
Bit 6 ARB0 (Arbitration Mode bit 0)
Bit 5 START (Start Sequence)
Bit 4 WATN (Select with SATN/ on a Start Sequence)
Bit 3 EPC (Enable Parity Checking)
Bit 2 Reserved
Bit 1 AAP (Assert SATN/ on Parity Error)
Bit 0 TRG (Target Mode)

Register 01h
SCSI Control One (SCNTL1)
Read/Write

EXC	ADB	DHP	CON	RST	AESP	IARB	SST
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bit 7 EXC (Extra Clock Cycle of Data Setup)
Bit 6 ADB (Assert SCSI Data Bus)
Bit 5 DHP (Disable Halt on Parity Error or ATN) (Target Only)
Bit 4 CON (Connected)
Bit 3 RST (Assert SCSI RST/ Signal)
Bit 2 AESP (Assert Even SCSI Parity (force bad parity))
Bit 1 IARB (Immediate Arbitration)
Bit 0 SST (Start SCSI Transfer)

Register 02h
SCSI Control Two (SCNTL2)
Read/Write

SDU	CHM	SLPMD	SLPH-BEN	WSS	VUE0	VUE1	WSR
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	X	0

Bit 7 SDU (SCSI Disconnect Unexpected)
Bit 6 CHM (Chained Mode)
Bit 5 SLPMD (SLPAR Mode Bit)
Bit 4 SLPHBEN (SLPAR High Byte Enable)
Bit 3 WSS (Wide SCSI Send)
Bit 2 VUE0 (Vendor Unique Enhancement bit 0)
Bit 1 VUE1 (Vendor Unique Enhancement bit 1)
Bit 0 WSR (Wide SCSI Receive)

Register 03h
SCSI Control Three (SCNTL3)
Read/Write

USE	SCF2	SCF1	SCF0	EWS	CCF2	CCF1	CCF0
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bit 7 USE (Ultra SCSI Enable)
Bits 6-4 SCF2-0 (Synchronous Clock Conversion Factor)
Bit 3 EWS (Enable Wide SCSI)
Bits 2-0 CCF2-0 (Clock Conversion Factor)

Register 04h
SCSI Chip ID (SCID)
Read/Write

RES	RRE	SRE	RES	ENC3	ENC2	ENC1	ENC0
7	6	5	4	3	2	1	0
Default >>>							
X	0	0	X	0	0	0	0

Bit 7 Reserved
Bit 6 RRE (Enable Response to Reselection)
Bit 5 SRE (Enable Response to Selection)
Bit 4 Reserved
Bits 3-0 Encoded Chip SCSI ID bits 3-0

Register 05h
SCSI Transfer (SXFER)
Read/Write

TP2	TP1	TP0	MO4	MO3	MO2	MO1	MO0
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	X	0	0	0	0

Bits 7-5 TP2-0 (SCSI Synchronous Transfer Period)
Bits 4-0 MO4-MO0 (Max SCSI Synchronous Offset)

Register 06h
SCSI Destination ID (SDID)
Read/Write

RES	RES	RES	RES	ENC3	ENC2	ENC1	ENC0
7	6	5	4	3	2	1	0
Default >>>							
X	X	X	X	0	0	0	0

Bits 7-4 Reserved
Bits 3-0 Encoded Destination SCSI ID

Register 07h
General Purpose (GPREG)
Read/Write

RES	RES	RES	GPIO4	RES	GPIO2	GPIO1	GPIO0
7	6	5	4	3	2	1	0
Default >>>							
X	X	X	0	X	X	X	X

Bits 7-3 Reserved
Bits 2-0 GPIO4-GPIO0 (General Purpose)

Register 08h
SCSI First Byte Received (SFBR)
Read/Write

1B7	1B6	1B5	1B4	1B3	1B2	1B1	1B0
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Register 09h
SCSI Output Control Latch (SOCL)
Read/Write

REQ	ACK	BSY	SEL	ATN	MSG	C/D	I/O
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bit 7 REQ(Assert SCSI REQ/ Signal)
Bit 6 ACK(Assert SCSI ACK/ Signal)
Bit 5 BSY(Assert SCSI BSY/ Signal)
Bit 4 SEL(Assert SCSI SEL/ Signal)
Bit 3 ATN(Assert SCSI ATN/ Signal)
Bit 2 MSG(Assert SCSI MSG/ Signal)
Bit 1 C/D(Assert SCSI C_D/ Signal)
Bit 0 I/O(Assert SCSI I_O/ Signal)

Register 0Ah
SCSI Selector ID (SSID)
Read Only

VAL	RES	RES	RES	ENID3	ENID2	ENID1	ENID0
7	6	5	4	3	2	1	0
Default >>>							
0	X	X	X	0	0	0	0

Bit 7 VAL (SCSI Valid)
Bits 6-4 Reserved
Bits 3-0 Encoded Destination SCSI ID

Register 0Bh
SCSI Bus Control Lines (SBCL)
Read Only

REQ	ACK	BSY	SEL	ATN	MSG	C/D	I/O
7	6	5	4	3	2	1	0
Default >>>							
X	X	X	X	X	X	X	X

Bit 7 REQ (SREQ/ Status)
Bit 6 ACK (SACK/ Status)
Bit 5 BSY (SBSY/ Status)
Bit 4 SEL (SSEL/ Status)
Bit 3 ATN (SATN/ Status)
Bit 2 MSG (SMSG/ Status)
Bit 1 C/D (SC_D/ Status)
Bit 0 I/O (SI_O/ Status)

Register 0Ch
DMA Status (DSTAT)
Read Only

DFE	MDPE	BF	ABRT	SSI	SIR	RES	IID
7	6	5	4	3	2	1	0
Default >>>							
1	0	0	0	0	0	X	0

Bit 7 DFE (DMA FIFO Empty)
Bit 6 MDPE (Master Data Parity Error)
Bit 5 BF (Bus Fault)
Bit 4 ABRT (Aborted)
Bit 3 SSI (Single Step Interrupt)
Bit 2 SIR (SCRIPTS Interrupt Instruction Received)
Bit 1 Reserved
Bit 0 IID (Illegal Instruction Detected)

Register 0Dh
SCSI Status Zero (SSTAT0)
Read Only

ILF	ORF	OLF	AIP	LOA	WOA	RST/	SDP0/
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bit 7 ILF (SIDL Least Significant Byte Full)
Bit 6 ORF (SODR Least Significant Byte Full)
Bit 5 OLF (SODL Least Significant Byte Full)
Bit 4 AIP (Arbitration in Progress)
Bit 3 LOA (Lost Arbitration)
Bit 2 WOA (Won Arbitration)
Bit 1 RST/ (SCSI RST/ Signal)
Bit 0 SDP0/ (SCSI SDP0/ Parity Signal)

Register 0Eh
SCSI Status One (SSTAT1)
Read Only

FF3	FF2	FF1	FF0	SDP0L	MSG	C/D	I/O
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	X	X	X	X

Bits 7-4 FF3-FF0 (FIFO Flags)
Bit 3 SDP0L (Latched SCSI Parity)
Bit 2 MSG (SCSI MSG/ Signal)
Bit 1 C/D (SCSI C_D/ Signal)
Bit 0 I/O (SCSI I_O/ Signal)

Register 0Fh
SCSI Status Two (SSTAT2)
Read Only

ILF1	ORF1	OLF1	FF4	SPL1	RES	LDSC	SDP1
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	X	X	1	X

Bit 7 ILF1 (SIDL Most Significant Byte Full)
Bit 6 ORF1 (SODR Most Significant Byte Full)
Bit 5 OLF1 (SODL Most Significant Byte Full)
Bit 4 FF4 (FIFO Flags bit 4)
Bit 3 SPL1(Latched SCSI parity for SD15-8)
Bit 2 Reserved
Bit 1 LDSC (Last Disconnect)
Bit 0 SDP1 (SCSI SDP1 Signal)

Registers 10h-13h
Data Structure Address (DSA)
Read/Write

Register 14h
Interrupt Status (ISTAT)
Read/Write

ABRT	SRST	SIGP	SEM	CON	INTF	SIP	DIP
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bit 7 ABRT (Abort Operation)
Bit 6 SRST (Software Reset)
Bit 5 SIGP (Signal Process)
Bit 4 SEM (Semaphore)
Bit 3 CON (Connected)
Bit 2 INTF (Interrupt on the Fly)
Bit 1 SIP (SCSI Interrupt Pending)
Bit 0 DIP (DMA Interrupt Pending)

Register 18h
Chip Test Zero (CTEST0)
Read/Write

RES	RES	RES	CM	SM	AP2	AP1	AP0
7	6	5	4	3	2	1	0
Default >>>							
X	X	X	0	0	0	0	0

Bits 7-5 Reserved
Bit 4 CM (Coma Mode)
Bit 3 SM (Snooze Mode)
Bits 2-0 AP2-0 (Arbitration Priority 2-0)

Register 19h
Chip Test One (CTEST1)
Read Only

FMT3	FMT2	FMT1	FMT0	FFL3	FFL2	FFL1	FFL0
7	6	5	4	3	2	1	0
Default >>>							
1	1	1	1	0	0	0	0

Bits 7-4 FMT3-0 (Byte Empty in DMA FIFO)
Bits 3-0 FFL3-0 (Byte Full in DMA FIFO)

Register 1Ah
Chip Test Two (CTEST2)
Read Only

DDIR	SIGP	CIO	CM	SRTCH	TEOP	DREQ	DACK
7	6	5	4	3	2	1	0
Default >>>							
0	0	X	X	0	0	0	1

Bit 7 DDIR (Data Transfer Direction)
Bit 6 SIGP (Signal Process)
Bit 5 CIO (Configured as I/O)
Bit 4 CM (Configured as Memory)
Bit 3 SRTCH (SCRATCHA/B Operation)
Bit 2 TEOP (SCSI True End of Process)
Bit 1 DREQ (Data Request Status)
Bit 0 DACK (Data Acknowledge Status)

Register 1Bh
Chip Test Three (CTEST3)
Read/Write

V3	V2	V1	V0	FLF	CLF	FM	WRIE
7	6	5	4	3	2	1	0
Default >>>							
x	x	x	x	0	0	0	0

Bits 7-4 V3-V0 (Chip revision level)
Bit 3 FLF (Flush DMA FIFO)
Bit 2 CLF (Clear DMA FIFO)
Bit 1 FM (Fetch Pin Mode)
Bit 0 WRIE (Write and Invalidate Enable)

Registers 1Ch-1Fh
Temporary (TEMP)
Read/Write

Register 20h
DMA FIFO (DFIFO)
Read/Write

BO7	BO6	BO5	BO4	B03	BO2	BO1	BO0
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bits 7-0 BO7-BO0 (Byte offset counter)

Register 21h
Chip Test Four (CTEST4)
Read/Write

BDIS	ZMOD	ZSD	SRTM	MPEE	FBL2	FBL1	FBL0
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bit 7 BDIS (Burst Disable)
Bit 6 ZMOD (High Impedance Mode)
Bit 5 ZSD (SCSI Data High Impedance)
Bit 4 SRTM (Shadow Register Test Mode)
Bit 3 MPEE (Master Parity Error Enable)
Bits 2-0 FBL2-FBL0 (FIFO Byte Control)

Register 22h
Chip Test Five (CTEST5)
Read/Write

ADCK	BBCK	DFS	MASR	DDIR	BL2	BO9	BO8
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bit 7 ADCK (Clock Address Incrementor)
Bit 6 BBCK (Clock Byte Counter)
Bit 5 DFS (DMA FIFO Size)
Bit 4 MASR (Master Control for Set or Reset Pulses)
Bit 3 DDIR (DMA Direction)
Bit 2 BL2 (Burst Length bit 2)
Bits 1-0 BO9-BO8 (DMA FIFO Byte Offset Counter bits 9-8)

Register 23h
Chip Test Six (CTEST6)
Read/Write

DF7	DF6	DF5	DF4	DF3	DF2	DF1	DF0
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bits 7-0 DF7-DF0 (DMA FIFO)

Registers 24h-26h
DMA Byte Counter (DBC)
Read/Write

Register 27h
DMA Command (DCMD)
Read/Write

Registers 28h-2Bh
DMA Next Address (DNAD)
Read/Write

Registers 2Ch-2Fh
DMA SCRIPTS Pointer (DSP)
Read/Write

Registers 30h-33h
DMA SCRIPTS Pointer Save (DSPS)
Read/Write

Registers 34h
Scratch Register A (SCRATCHA)
Read/Write

Register 38h
DMA Mode (DMODE)
Read/Write

BL1	BL0	SIOM	DIOM	ER	ERMP	BOF	MAN
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bit 7-6 BL1-BL0 (Burst Length)
Bit 5 SIOM (Source I/O-Memory Enable)
Bit 4 DIOM (Destination I/O-Memory Enable)
Bit 3 ERL (Enable Read Line)
Bit 2 ERMP (Enable Read Multiple)
Bit 1 BOF (Burst Op Code Fetch Enable)
Bit 0 MAN (Manual Start Mode)

Register 39h
DMA Interrupt Enable (DIEN)
Read/Write

RES	MDPE	BF	ABRT	SSI	SIR	RES	IID
7	6	5	4	3	2	1	0
Default >>>							
X	0	0	0	0	0	X	0

Bit 7 Reserved
Bit 6 MDPE (Master Data Parity Error)
Bit 5 BF (Bus Fault)
Bit 4 ABRT (Aborted)
Bit 3 SSI (Single -Step Interrupt)
Bit 2 SIR (SCRIPTS Interrupt Instruction Received
Bit 1 Reserved
Bit 0 IID (Illegal Instruction Detected)

Register 3Ah
Scratch Byte Register (SBR)
Read/Write

Register 3Bh
DMA Control (DCNTL)
Read/Write

CLSE	PFF	PFEN	SSM	INTM	STD	INTD	COM
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bit 7 CLSE (Cache Line Size Enable)
Bit 6 PFF (Pre-fetch Flush)
Bit 5 PFEN (Pre-fetch Enable)
Bit 4 SSM (Single-step Mode)
Bit 3 INTM (INTA Mode)
Bit 2 STD (Start DMA Operation)
Bit 1 INTD (INTA Disable)
Bit 0 COM (53C700 Compatibility)

Register 3Ch-3Fh
Adder Sum Output (ADDER)
Read Only

Register 40h
SCSI Interrupt Enable Zero (SIEN0)
Read/Write

M/A	CMP	SEL	RSL	SGE	UDC	RST	PAR
7	6	5	4	3	2	1	0
Default >>>							
0	0	0	0	0	0	0	0

Bit 7 M/A (SCSI Phase Mismatch - Initiator Mode; SCSI ATN Condition - Target Mode)
Bit 6 CMP (Function Complete)
Bit 5 SEL (Selected)
Bit 4 RSL (Reselected)
Bit 3 SGE (SCSI Gross Error)
Bit 2 UDC (Unexpected Disconnect)
Bit 1 RST (SCSI Reset Condition)
Bit 0 PAR (SCSI Parity Error)

Register 41h
SCSI Interrupt Enable One (SIEN1)
Read/Write

RES 7	RES 6	RES 5	RES 4	WI 3	STO 2	GEN 1	HTH 0
Default >>>							
X	X	X	X	0	0	0	0

Bits 7-4 Reserved
Bit 3 WI (Wakeup)
Bit 2 STO (Selection or Reselection Time-out)
Bit 1 GEN (General Purpose Timer Expired)
Bit 0 HTH (Handshake-to-Handshake Timer Expired)

Register 42h
SCSI Interrupt Status Zero (SIST0)
Read Only

M/A 7	CMP 6	SEL 5	RSL 4	SGE 3	UDC 2	RST 1	PAR 0
Default >>>							
0	0	0	0	0	0	0	0

Bit 7 M/A (Initiator Mode: Phase Mismatch; Target Mode: SATN/ Active)
Bit 6 CMP (Function Complete)
Bit 5 SEL (Selected)
Bit 4 RSL (Reselected)
Bit 3 SGE (SCSI Gross Error)
Bit 2 UDC (Unexpected Disconnect)
Bit 1 RST (SCSI RST/ Received)
Bit 0 PAR (Parity Error)

Register 43h
SCSI Interrupt Status One (SIST1)
Read Only

RES 7	RES 6	RES 5	RES 4	WI 3	STO 2	GEN 1	HTH 0
Default >>>							
X	X	X	X	0	0	0	0

Bits 7-4 Reserved
Bit 3 WI (Wakeup)
Bit 2 STO (Selection or Reselection Time-out)
Bit 1 GEN (General Purpose Timer Expired)
Bit 0 HTH (Handshake-to-Handshake Timer Expired)

Register 45h
SCSI Wide Residue (SWIDE)
Read/Write

Register 46h
Memory Access Control (MACNTL)
Read/Write

TYP3 7	TYP2 6	TYP1 5	TYP0 4	DWR 3	DRD 2	PSCPT 1	SCPTS 0
Default >>>							
1	0	1	0	0	0	0	0

Bits 7-4 TYP3-0 (Chip Type)
Bit 3 DWR (DataWR)
Bit 2 DRD (DataRD)
Bit 1 PSCPT (Pointer SCRIPTS)
Bit 0 SCPTS (SCRIPTS)

Register 47h
General Purpose Pin Control (GPCNTL)
Read/Write

ME 7	FE 6	RES 5	GPIO4 4	RES 3	GPIO2 2	GPIO1 1	GPIO0 0
Default >>>							
0	0	X	0	1	1	1	1

Bit 7 Master Enable
Bit 6 Fetch Enable
Bits 5
3Reserved
Bits 4
2GPIO4_EN/GPIO2_EN (GPIO Enable)
Bits 1-0 GPIO1_EN– GPIO0_EN (GPIO Enable)

Register 48h
SCSI Timer Zero (STIME0)
Read/Write

HTH7 7	HTH6 6	HTH5 5	HRH4 4	SEL 3	SEL 2	SEL 1	SEL 0
Default >>>							
0	0	0	0	0	0	0	0

Bits 7-4 HTH (Handshake-to-Handshake Timer Period)
Bits 3-0 SEL (Selection Time-Out)

Register 49h
SCSI Timer One (STIME1)
Read/Write

RES 7	HTHBA 6	GENSF 5	HTHSF 4	GEN3 3	GEN2 2	GEN1 1	GEN0 0
Default >>>							
X	0	0	0	0	0	0	0

Bit 7 Reserved
Bit 6 HTHBA (Handshake-to-Handshake Timer Bus Activity Enable)
Bit 5 GENSF (General Purpose Timer Scale Factor)
Bit 4 HTHSF (Handshake to Handshake Timer Scale Factor)
Bits 3-0 GEN3-0 (General Purpose Timer Period)

Register 4Ah
Response ID Zero (RESPID0)
Read/Write

ID 7	ID 6	ID 5	ID 4	ID 3	ID 2	ID 1	ID 0
Default >>>							
X	X	X	X	X	X	X	X

Register 4B
Response ID One(RESPID1)
Read/Write

ID 15	ID 14	ID 13	ID 12	ID 11	ID 10	ID 9	ID 8
Default >>>							
X	X	X	X	X	X	X	X

Register 4Ch
SCSI Test Zero (STEST0)
Read Only

SSAID3 7	SSAID2 6	SSAID1 5	SSAID0 4	SLT 3	ART 2	SOZ 1	SOM 0
Default >>>							
0	0	0	0	0	X	1	1

Bits 7-4 SSAID3-0 (SCSI Selected As ID)
Bit 3 SLT (Selection Response Logic Test)
Bit 2 ART (Arbitration Priority Encoder Test)
Bit 1 SOZ (SCSI Synchronous Offset Zero)
Bit 0 SOM (SCSI Synchronous Offset Maximum)

Register 4Dh
SCSI Test One (STEST1)
Read/Write

RES 7	SISO 6	RES 5	RES 4	DBLEN 3	DBLSEL 2	RES 1	RES 0
Default >>>							
0	0	X	X	0	0	X	X

Bit 7 Reserved
Bit 6 SISO (SCSI Isolation Mode)
Bits 5-4 Reserved
Bit 3 DBLEN (SCLK Doubler Enable)
Bit 2 DBLSEL (SCLK Doubler Select)
Bits 1-0 Reserved

Register 4Eh
SCSI Test Two (STEST2)
Read/Write

SCE 7	ROF 6	RES 5	SLB 4	SZM 3	AWS 2	EXT 1	LOW 0
Default >>>							
0	0	0	0	0	0	0	0

Bit 7 SCE (SCSI Control Enable)
Bit 6 ROF (Reset SCSI Offset)
Bit 5 Reserved
Bit 4 SLB (SCSI Loopback Mode)
Bit 3 SZM (SCSI High-Impedance Mode)
Bit 2 AWS (Always Wide SCSI)
Bit 1 EXT (Extend SREQ/SACK Filtering)
Bit 0 LOW (SCSI Low level Mode)

Register 4Fh
SCSI Test Three (STEST3)
Read/Write

TE 7	STR 6	HSC 5	DSI 4	RES 3	TTM 2	CSF 1	STW 0
Default >>>							
0	0	0	0	X	0	0	0

Bit 7 TE (TolerANT Enable)
Bit 6 STR (SCSI FIFO Test Read)
Bit 5 HSC (Halt SCSI Clock)
Bit 4 DSI (Disable Single Initiator Response)
Bit 3 Reserved
Bit 2 TTM (Timer Test Mode)
Bit 1 CSF (Clear SCSI FIFO)
Bit 0 STW (SCSI FIFO Test Write)

Register 50h-51h
SCSI Input Data Latch (SIDL)
Read Only

Registers 54h-55h
SCSI Output Data Latch (SODL)
Read/Write

Registers 58h-59h
SCSI Bus Data Lines (SBDL)
Read Only

Registers 5Ch-5Fh
Scratch Register B
(SCRATCHB)
Read/Write

Registers 60h-7Fh
Scratch Registers C-J
(SCRATCHC-SCRATCHJ)
Read/Write

SYM53C895 Operating Registers

Register 00 (80)
SCSI Control Zero (SCNTL0)
Read/Write

ARB1	ARB0	START	WATN	EPC	RES	AAP	TRG
7	6	5	4	3	2	1	0
Default>>>							
1	1	0	0	0	X	0	0

Bit 7 ARB1 (Arbitration Mode bit 1)
Bit 6 ARB0 (Arbitration Mode bit 0)
Bit 5 START (Start Sequence)
Bit 4 WATN (Select with SATN/ on a Start Sequence)
Bit 3 EPC (Enable Parity Checking)
Bit 2 Reserved
Bit 1 AAP (Assert SATN/ on Parity Error)
Bit 0 TRG (Target Mode)

Register 01 (81)
SCSI Control One (SCNTL1)
Read/Write

EXC	ADB	DHP	CON	RST	AESP	IARB	SST
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 EXC (Extra Clock Cycle of Data Setup)
Bit 6 ADB (Assert SCSI Data Bus)
Bit 5 DHP (Disable Halt on Parity Error or ATN) (Target Only)
Bit 4 CON (Connected)
Bit 3 RST (Assert SCSI RST/ Signal)
Bit 2 AESP (Assert Even SCSI Parity (force bad parity))
Bit 1 IARB (Immediate Arbitration)
Bit 0 SST (Start SCSI Transfer)

Register 02 (82)
SCSI Control Two (SCNTL2)
Read/Write

SDU	CHM	SLPMD	SLPHBEN	WSS	VUE0	VUE1	WSR
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 SDU (SCSI Disconnect Unexpected)
Bit 6 CHM (Chained Mode)
Bit 5 SLPMD (SLPAR Mode Bit)
Bit 4 SLPHBEN (SLPAR High Byte Enable)
Bit 3 WSS (Wide SCSI Send)
Bit 2 VUE0 (Vendor Unique Enhancements bit 0)
Bit 1 VUE1 (Vendor Unique Enhancements bit 1)
Bit 0 WSR (Wide SCSI Receive)

Register 03 (83)
SCSI Control Three (SCNTL3)
Read/Write

ULTRA	SCF2	SCF1	SCF0	EWS	CCF2	CCF1	CCF0
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 ULTRA (Ultra Enable)
Bits 6-4 SCF2-0 (Synchronous Clock Conversion Factor))
Bit 3 EWS (Enable Wide SCSI)
Bits 2-0 CCF2-0 (Clock Conversion Factor)

Register 04 (84)
SCSI Chip ID (SCID)
Read/Write

RES	RRE	SRE	RES	ENC3	ENC2	ENC1	ENC0
7	6	5	4	3	2	1	0
Default>>>							
X	0	0	X	0	0	0	0

Bit 7 Reserved
Bit 6 RRE (Enable Response to Reselection)
Bit 5 SRE (Enable Response to Selection)
Bit 4 Reserved
Bits 3-0 Encoded Chip SCSI ID, bits 3-0

Register 05 (85)
SCSI Transfer (SXFER)
Read/Write

TP2	TP1	TP0	MO4	MO3	MO2	MO1	MO0
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bits 7-5 TP2-0 (SCSI Synchronous Transfer Period)
Bits 4-0 MO4-MO0 (Max SCSI Synchronous Offset)

Register 06 (86)
SCSI Destination ID (SDID)
Read/Write

RES	RES	RES	RES	ENC3	ENC2	ENC1	ENC0
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	0	0	0	0

Bits 7-4 Reserved
Bits 3-0 Encoded Destination SCSI ID

Rexgister 07 (87)
General Purpose (GPREG)
Read/Write

RES	RES	RES	GPIO4	GPIO3	GPIO2	GPIO1	GPIO0
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	0	X	X	X	X

Bits 7-5 Reserved
Bits 4-0 GPIO4-GPIO0 (General Purpose)

Register 08 (88)
SCSI First Byte Received (SFBR)
Read/Write

1B7	1B6	1B5	1B4	1B3	1B2	1B1	1B0
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Register 09 (89)
SCSI Output Control Latch (SOCL)
Read /Write

REQ	ACK	BSY	SEL	ATN	MSG	C/D	I/O
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 REQ(Assert SCSI REQ/ Signal)
Bit 6 ACK(Assert SCSI ACK/ Signal)
Bit 5 BSY(Assert SCSI BSY/ Signal)
Bit 4 SEL(Assert SCSI SEL/ Signal)
Bit 3 ATN(Assert SCSI ATN/ Signal)
Bit 2 MSG(Assert SCSI MSG/ Signal)
Bit 1 C/D(Assert SCSI C_D/ Signal)
Bit 0 I/O(Assert SCSI I_O/ Signal)

Register 0A (8A)
SCSI Selector ID (SSID)
Read Only

VAL	RES	RES	RES	ENID3	ENID2	ENID1	ENID0
7	6	5	4	3	2	1	0
Default>>>							
0	X	X	X	0	0	0	0

Bit 7 VAL (SCSI Valid)
Bits 6-4 Reserved
Bits 3-0 Encoded Destination SCSI ID

Register 0B (8B)
SCSI Bus Control Lines (SBCL)
Read Only

REQ	ACK	BSY	SEL	ATN	MSG	C/D	I/O
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	X	X	X	X	X

Bit 7 REQ (SREQ/ Status)
Bit 6 ACK (SACK/ Status)
Bit 5 BSY (SBSY/ Status)
Bit 4 SEL (SSEL/ Status)
Bit 3 ATN (SATN/ Status)
Bit 2 MSG (SMSG/ Status)
Bit 1 C/D (SC_D/ Status)
Bit 0 I/O (SI_O/ Status)

Register 0C (8C)
DMA Status (DSTAT)
Read Only

DFE	MDPE	BF	ABRT	SSI	SIR	RES	IID
7	6	5	4	3	2	1	0
Default>>>							
1	0	0	0	0	0	X	0

Bit 7 DFE (DMA FIFO Empty)
Bit 6 MDPE (Master Data Parity Error)
Bit 5 BF (Bus Fault)
Bit 4 ABRT (Aborted)
Bit 3 SSI (Single Step Interrupt)
Bit 2 SIR (SCRIPTS Interrupt Instruction Received)
Bit 1 Reserved
Bit 0 IID (Illegal Instruction Detected)

Register 0D (8D)
SCSI Status Zero (SSTAT0)
Read Only

ILF	ORF	OLF	AIP	LOA	WOA	RST	SDP0/
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 ILF (SIDL Least Significant Byte Full)
Bit 6 ORF (SODR Least Significant Byte Full)
Bit 5 OLF (SODL Least Significant Byte Full)
Bit 4 AIP (Arbitration in Progress)
Bit 3 LOA (Lost Arbitration)
Bit 2 WOA (Won Arbitration)
Bit 1 RST/ (SCSI RST/ Signal)
Bit 0 SDP0/ (SCSI SDP0/ Parity Signal)

Register 0E (8E)
SCSI Status One (SSTAT1)
Read Only

FF3	FF2	FF1	FF0	SDP0L	MSG	C/D	I/O
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	X	X	X	X

Bits 7-4 FF3-FF0 (FIFO Flags)
Bit 3 SDP0L (Latched SCSI Parity)
Bit 2 MSG (SCSI MSG/ Signal)
Bit 1 C/D (SCSI C_D/ Signal)
Bit 0 I/O (SCSI I_O/ Signal)

Register 0F (8F)
SCSI Status Two (SSTAT2)
(Read Only)

ILF1	ORF1	OLF1	FF4	SPL1	DM	LDSC	SDP1
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	X	X	1	X

Bit 7 ILF1 (SIDL Most Significant Byte Full)
Bit 6 ORF1 (SODR Most Significant Byte Full)
Bit 5 OLF1 (SODL Most Significant Byte Full)
Bit 4 FF4 (FIFO Flags bit 4)
Bit 3 SPL1(Latched SCSI parity for SD15-8)
Bit 2 DM (DIFFSENS Mismatch)
Bit 1 LDSC (Last Disconnect)
Bit 0 SDP1 (SCSI SDP1 Signal)

Registers 10-13 (90-93)
Data Structure Address (DSA)
Read/Write

Register 14 (94)
Interrupt Status (ISTAT)
(Read/Write)

ABRT	SRST	SIGP	SEM	CON	INTF	SIP	DIP
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 ABRT (Abort Operation)
Bit 6 SRST (Software Reset)
Bit 5 SIGP (Signal Process)
Bit 4 SEM (Semaphore)
Bit 3 CON (Connected)
Bit 2 INTF (Interrupt on the Fly)
Bit 1 SIP (SCSI Interrupt Pending)
Bit 0 DIP (DMA Interrupt Pending)

Register 18 (98)
Chip Test Zero (CTEST0)
Read/Write

Register 19 (99)
Chip Test One (CTEST1)
Read Only

FMT3	FMT2	FMT1	FMT0	FFL3	FFL2	FFL1	FFL0
7	6	5	4	3	2	1	0
Default>>>							
1	1	1	1	0	0	0	0

Bits 7-4 FMT3-0 (Byte Empty in DMA FIFO)
Bits 3-0 FFL3-0 (Byte Full in DMA FIFO)

Register 1A (9A)
Chip Test Two (CTEST2)
Read/Write

DDIR	SIGP	CIO	CM	SRTCH	TEOP	DREQ	DACK
7	6	5	4	3	2	1	0
Default>>>							
0	0	X	X	0	0	0	1

Bit 7 DDIR (Data Transfer Direction)
Bit 6 SIGP (Signal Process)
Bit 5 CIO (Configured as I/O)
Bit 4 CM (Configured as Memory)
Bit 3 SRTCH (SCRATCHA/B Operation)
Bit 2 TEOP (SCSI True End of Process)
Bit 1 DREQ (Data Request Status)
Bit 0 DACK (Data Acknowledge Status)

Register 1B (9B)
Chip Test Three (CTEST3)
Read/Write

V3	V2	V1	V0	FLF	CLF	FM	WRIE
7	6	5	4	3	2	1	0
Default>>>							
x	x	x	x	0	0	0	0

Bits 7-4 V3-V0 (Chip revision level)
Bit 3 FLF (Flush DMA FIFO)
Bit 2 CLF (Clear DMA FIFO)
Bit 1 FM (Fetch Pin Mode)
Bit 0 WRIE (Write and Invalidate Enable)

Registers 1C-1F (9C-9F)
Temporary (TEMP)
Read/Write

Register 20 (A0)
DMA FIFO (DFIFO)
Read/Write

BO7	BO6	BO5	BO4	Bo3	BO2	BO1	BO0
7	6	5	4	3	2	1	0
Default>>>							
X	0	0	0	0	0	0	0

Bits 7-0 BO7-BO0 (Byte offset counter)

Register 21 (A1)
Chip Test Four (CTEST4)
Read/Write

BDIS	ZMOD	ZSD	SRTM	MPEE	FBL2	FBL1	FBL0
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 BDIS (Burst Disable)
Bit 6 ZMOD (High Impedance Mode)
Bit 5 ZSD (SCSI Data High Impedance)
Bit 4 SRTM (Shadow Register Test Mode)
Bit 3 MPEE (Master Parity Error Enable)
Bits 2-0 FBL2-FBL0 (FIFO Byte Control

Register 22 (A2)
Chip Test Five (CTEST5)
Read/Write

ADCK	BBCK	DFS	MASR	DDIR	BL2	BO9	BO8
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	X	X	X

Bit 7 ADCK (Clock Address Incrementor)
Bit 6 BBCK (Clock Byte Counter)
Bit 5 DFS (DMA FIFO Size)
Bit 4 MASR (Master Control for Set or Reset Pulses)
Bit 3 DDIR (DMA Direction)
Bit 2 BL2 (Burst Length bit 2)
Bits 1-0 BO9-8

Register 23 (A3)
Chip Test Six (CTEST6)
Read/Write

DF7	DF6	DF5	DF4	DF3	DF2	DF1	DF0
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bits 7-0 DF7-DF0 (DMA FIFO)

Registers 24-26 (A4-A6)
DMA Byte Counter (DBC)
Read/Write

Register 27 (A7)
DMA Command (DCMD)
Read/Write

Registers 28-2B (A8-AB)
DMA Next Address (DNAD)
Read/Write

Registers 2C-2F (AC-AF)
DMA SCRIPTS Pointer (DSP)
Read/Write

Registers 30-33 (B0-B3)
DMA SCRIPTS Pointer Save (DSPS)
Read/Write

Registers 34-37 (B4-B7)
Scratch Register A (SCRATCH A)
Read/Write

Register 38 (B8)
DMA Mode (DMODE)
Read/Write

BL1	BL0	SIOM	DIOM	ER	ERMP	BOF	MAN
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7-6 BL1-BL0 (Burst Length)
Bit 5 SIOM (Source I/O-Memory Enable)
Bit 4 DIOM (Destination I/O-Memory Enable)
Bit 3 ERL (Enable Read Line)
Bit 2 ERMP (Enable Read Multiple)
Bit 1 BOF (Burst Op Code Fetch Enable)
Bit 0 MAN (Manual Start Mode)

Register 39 (B9)
DMA Interrupt Enable (DIEN)
Read/Write

RES	MDPE	BF	ABRT	SSI	SIR	RES	IID
7	6	5	4	3	2	1	0
Default>>>							
X	0	0	0	0	0	X	0

Bit 7 Reserved
Bit 6 MDPE (Master Data Parity Error)
Bit 5 BF (Bus Fault)
Bit 4 ABRT (Aborted)
Bit 3 SSI (Single -step Interrupt)
Bit 2 SIR (SCRIPTS Interrupt Instruction Received
Bit 1 RES
Bit 0 IID (Illegal Instruction Detected)

Register 3A (BA)
Scratch Byte Register (SBR)
Read/Write

Register 3B (BB)
DMA Control (DCNTL)
Read/Write

CLSE	PFF	PFEN	SSM	IRQM	STD	IRQD	COM
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 CLSE (Cache Line Size Enable)
Bit 6 PFF (Pre-fetch Flush)
Bit 5 PFEN (Pre-fetch Enable)
Bit 4 SSM (Single-step Mode)
Bit 3 IRQM (IRQ Mode)
Bit 2 STD (Start DMA Operation)
Bit 1 IRQD (IRQ Disable)
Bit 0 COM (53C700 Compatibility)

Register 3C-3F (BC-BF)
Adder Sum Output (ADDER)
Read Only

Register 40 (C0)
SCSI Interrupt Enable Zero (SIEN0)
Read/Write

M/A	CMP	SEL	RSL	SGE	UDC	RST	PAR
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 M/A (SCSI Phase Mismatch - Initiator Mode; SCSI ATN Condition - Target Mode)
Bit 6 CMP (Function Complete)
Bit 5 SEL (Selected)
Bit 4 RSL (Reselected)
Bit 3 SGE (SCSI Gross Error)
Bit 2 UDC (Unexpected Disconnect)
Bit 1 RST (SCSI Reset Condition)
Bit 0 PAR (SCSI Parity Error)

Register 41 (C1)
SCSI Interrupt Enable One (SIEN1)
Read/Write

RES	RES	RES	SBMC	RES	STO	GEN	HTH
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	0	X	0	0	0

Bits 7-5 Reserved
Bit 4 SBMC (SCSI Bus Mode Change)
Bit 3 Reserved
Bit 2 STO (Selection or Reselection Time-out)
Bit 1 GEN (General Purpose Timer Expired)
Bit 0 HTH (Handshake-to-Handshake Timer Expired)

Register 42 (C2)
SCSI Interrupt Status Zero (SIST0)
Read Only

M/A	CMP	SEL	RSL	SGE	UDC	RST	PAR
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 M/A (Initiator Mode: Phase Mismatch; Target Mode: SATN/ Active)
Bit 6 CMP (Function Complete)
Bit 5 SEL (Selected)
Bit 4 RSL (Reselected)
Bit 3 SGE (SCSI Gross Error)
Bit 2 UDC (Unexpected Disconnect)
Bit 1 RST (SCSI RST/ Received)
Bit 0 PAR (Parity Error)

Register 43 (C3)
SCSI Interrupt Status One (SIST1)
Read Only

RES	RES	RES	SBMC	RES	STO	GEN	HTH
7	6	5	4	3	2	1	0
Default>>>							
X	X	X	0	X	0	0	0

Bits 7-5 Reserved
Bit 4 SBMC (SCSI Bus Mode Change)
Bit 3 Reserved
Bit 2 STO (Selection or Reselection Time-out)
Bit 1 GEN (General Purpose Timer Expired)
Bit 0 HTH (Handshake-to-Handshake Timer Expired)

Register 44 (C4)
SCSI Longitudinal Parity (SLPAR)
Read/Write

Register 45 (C5)
SCSI Wide Residue (SWIDE)
Read/Write

Register 46 (C6)
Memory Access Control (MACNTL)
Read/Write

TYP3	TYP2	TYP1	TYP0	DWR	DRD	PSCPT	SCPTS
7	6	5	4	3	2	1	0
Default>>>							
1	1	0	1	0	0	0	0

Bits 7-4 TYP3-0 (Chip Type)
Bit 3 DWR (DataWR)
Bit 2 DRD (DataRD)
Bit 1 PSCPT (Pointer SCRIPTS)
Bit 0 SCPTS (SCRIPTS)

Register 47 (C7)
General Purpose Pin Control (GPCNTL)
Read/Write

ME	FE	RES	GPIO4	GPIO3	GPIO2	GPIO1	GPIO0
7	6	5	4	3	2	1	0
Default>>>							
0	0	X	0	1	1	1	1

Bit 7 Master Enable
Bit 6 Fetch Enable
Bit 5 Reserved
Bits 4-2 GPIO4_EN–GPIO2_EN (GPIO Enable)
Bits 1-0 GPIO1_EN– GPIO0_EN (GPIO Enable)

Register 48 (C8)
SCSI Timer Zero (STIME0)
Read /Write

HTH	HTH	HTH	HRH	SEL	SEL	SEL	SEL
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bits 7-4 HTH (Handshake-to-Handshake Timer Period)
Bits 3-0 SEL (Selection Time-Out)

Register 49 (C9)
SCSI Timer One (STIME1)
Read/Write

RES	HTHBA	GENSF	HTHSF	GEN3	GEN2	GEN1	GEN0
7	6	5	4	3	2	1	0
Default>>>							
X	0	0	0	0	0	0	0

Bit 7 Reserved
Bit 6 HTHBA (Handshake-to-Handshake Timer Bus Activity Enable)
Bit 5 GENSF (General Purpose Timer Scale Factor)
Bit 4 HTHSF (Handshake to Handshake Timer Scale Factor)
Bits 3-0 GEN3-0 (General Purpose Timer Period)

Register 4A (CA)
Response ID Zero (RESPID0)
Read/Write

Register 4B (CB)
Response ID One(RESPID1)
Read/Write

.Register 4C (CC)
SCSI Test Zero (STEST0)
Read Only

SSAID3	SSAID2	SSAID1	SSAID0	SLT	ART	SOZ	SOM
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	X	1	1

Bits 7-4 SSAID (SCSI Selected As ID)
Bit 3 SLT (Selection Response Logic Test)
Bit 2 ART (Arbitration Priority Encoder Test)
Bit 1 SOZ (SCSI Synchronous Offset Zero)
Bit 0 SOM (SCSI Synchronous Offset Maximum)

Register 4D (CD)
SCSI Test One (STEST1)
Read/Write

SCLK	SISO	RES	RES	QEN	QSEL	RES	RES
7	6	5	4	3	2	1	0
Default>>>							
0	0	X	X	0	0	X	X

Bit 7 SCLK
Bit 6 SISO (SCSI Isolation Mode)
Bits 5-4 Reserved
Bit 3 QEN (SCLK Quadrupler Enable)
Bit 2 QSEL (SCLK Quadrupler Select)
Bits1-0 Reserved

Register 4E (CE)
SCSI Test Two (STEST2)
Read/Write

SCE	ROF	DIF	SLB	SZM	AWS	EXT	LOW
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 SCE (SCSI Control Enable)
Bit 6 ROF (Reset SCSI Offset)
Bit 5 DIF (SCSI Differential Mode)
Bit 4 SLB (SCSI Loopback Mode)
Bit 3 SZM (SCSI High-Impedance Mode)
Bit 2 AWS (Always Wide SCSI)
Bit 1 EXT (Extend SREQ/SACK Filtering)
Bit 0 LOW (SCSI Low level Mode)

Register 4F (CF)
SCSI Test Three (STEST3)
Read/Write

TE	STR	HSC	DSI	S16	TTM	CSF	STW
7	6	5	4	3	2	1	0
Default>>>							
0	0	0	0	0	0	0	0

Bit 7 TE (TolerANT Enable)
Bit 6 STR (SCSI FIFO Test Read)
Bit 5 HSC (Halt SCSI Clock)
Bit 4 DSI (Disable Single Initiator Response)
Bit 3 S16 (16-bit System)
Bit 2 TTM (Timer Test Mode)
Bit 1 CSF (Clear SCSI FIFO)
Bit 0 STW (SCSI FIFO Test Write)

Register 50-51 (D0-D1)
SCSI Input Data Latch (SIDL)
Read Only

Register 52 (D2)
SCSI Test 4 (STEST4)
Read Only

SMODE		LOCK	RES	RES	RES	RES	RES
7	6	5	4	3	2	1	0
Default>>>							
X	X	0	X	X	X	X	X

Bit 7-6 SMODE (SCSI Mode)
Bit 5 LOCK (Frequency Lock)
Bits 4-0 Reserved

Registers 54-55 (D4-D5)
SCSI Output Data Latch (SODL)
Read/Write

Registers 58-59 (D8-D9)
SCSI Bus Data Lines (SBDL)
Read Only

Registers 5C-5F (DC-DF)
Scratch Register B (SCRATCHB)
(Read/Write)

Registers 60h-7Fh (E0h-FFh)
Scratch Registers C-J
(SCRATCHC-SCRATCHJ)
Read/Write

Appendix C

Multi-Threaded SCRIPTS Example

```
;*****************************************************************
; 53C810 MULTI THREAD EXAMPLE
;*****************************************************************
; ABSOLUTE declarations

ABSOLUTE SCSI_id       = 0
ABSOLUTE MATCH_SCSI_ID = 0x81

; Messages
ABSOLUTE  CMD_COMPLETE_  = 0x00
ABSOLUTE  EXTEND_MSG_ = 0x01
ABSOLUTE  SAVE_DATAPTR_ = 0x02
ABSOLUTE  DISCONNECT_                = 0x04
ABSOLUTE  MSG_REJECT_ = 0x07

; Interrupt codes
ABSOLUTE  error_not_cmd_phase        = 0x01
ABSOLUTE  error_not_data_in_phase    = 0x02
ABSOLUTE  error_not_data_out_phase    = 0x03
ABSOLUTE  error_not_msg_in_phase     = 0x04
ABSOLUTE  error_not_msg_out_phase     = 0x05
ABSOLUTE  error_not_status_phase      = 0x06
ABSOLUTE  error_unexpected_phase      = 0x07
ABSOLUTE  error_jump_not_taken        = 0x10
ABSOLUTE  error_not_cmd_complete      = 0x20
ABSOLUTE  error_not_extended_msg      = 0x21
ABSOLUTE  io_complete                 = 0x0A
ABSOLUTE  setup_SXFER                = 0x888
ABSOLUTE  reselect_id_error           = 0x999
ABSOLUTE  select_error                = 0xfff
;*****************************************************************
; TABLE declarations for Table Indirect offsets in bytes
Table Table_Indirect                                    \
SCSI_ID=ID{0x00,0x00,0x00,0x00}, \
identify_msg_buf = {0xc0},                               \
synch_msgi_buf  = 5{??},                                 \
cmd_buf=12{??},                                          \
status_buf = 1{??},                                       \
msg_in_buf = 1{??},                                       \
data_buf = 512{??}

;*****************************************************************
; ENTRY declarations
ENTRY multi_thread
ENTRY to_decisions
```

```
ENTRY id_msg_out
ENTRY msg_in_phase
 ENTRY cmd_phase
ENTRY data_in_phase
ENTRY data_out_phase
ENTRY status_phase
ENTRY disconnected
ENTRY entry0
ENTRY entry1
ENTRY entry2
ENTRY io_request0
ENTRY io_request1
ENTRY io_request2
ENTRY schedule_NOP

; Scheduler SCRIPT code
scheduler:

entry0:
 ;Initialize DSA register with table base address for using table
 ;indirect addressing
 MOVE MEMORY 4, PATCH_addr_of_table0_ptr, PATCH_chip_physaddr+DSA
 ;Initilize address for changing jump to nop after starting new I/O
 ;(after SELECT instruction in main SCRIPT code)
 MOVE MEMORY 4,
PATCH_SCRIPTphysaddr+io_request0,PATCH_SCRIPTphysaddr+schedule_NOP+8
io_request0:
 JUMP REL(multi_thread)

entry1:
 MOVE MEMORY 4, PATCH_addr_of_table1_ptr, PATCH_chip_physaddr+DSA
 MOVE MEMORY 4,
PATCH_SCRIPTphysaddr+io_request1,PATCH_SCRIPTphysaddr+schedule_NOP+8
io_request1:
 JUMP REL(multi_thread)

entry2:
 MOVE MEMORY 4, PATCH_addr_of_table2_ptr, PATCH_chip_physaddr+DSA
 MOVE MEMORY 4,
PATCH_SCRIPTphysaddr+io_request2,PATCH_SCRIPTphysaddr+schedule_NOP+8
io_request2:
 JUMP REL(multi_thread)

 JUMP REL(wait_for_reselect)


;*********************************************************************
; main SCRIPT code
multi_thread:
 SELECT ATN FROM SCSI_id, REL(wait_for_reselect)

;Change jump to nop in scheduler after starting new I/O
;the destination address is initialized from scheduler SCRIPT
```

```
schedule_NOP:
 MOVE MEMORY 4, PATCH_nop_physaddr, PATCH_place_hold_addr

 JUMP REL(to_decisions), WHEN NOT MSG_OUT

id_msg_out:
 MOVE FROM identify_msg_buf, WHEN MSG_OUT
 JUMP REL(to_decisions), WHEN NOT CMD

cmd_phase:
 CLEAR ATN
 MOVE FROM cmd_buf, WHEN CMD
 JUMP REL(to_decisions), WHEN NOT DATA_IN

data_in_phase:
 MOVE FROM data_buf, WHEN DATA_IN
 JUMP REL(status_phase), WHEN STATUS
 JUMP REL(to_decisions)

data_out_phase:
 MOVE FROM data_buf, WHEN DATA_OUT
 JUMP REL(to_decisions), WHEN NOT STATUS

status_phase:
 MOVE FROM status_buf, WHEN STATUS
 JUMP REL(to_decisions), WHEN NOT MSG_IN
msg_in_phase:
 MOVE FROM msg_in_buf, WHEN MSG_IN
 JUMP REL(disconnected), IF DISCONNECT_
 JUMP REL(msg_in_phase), WHEN SAVE_DATAPTR_ ;compare data, wait for
phase
 INT error_not_cmd_complete, IF NOT 0x00
 CLEAR ACK
 MOVE SCNTL2 & 0x7F TO SCNTL2
 WAIT DISCONNECT
 INT io_complete
disconnected:
 MOVE SCNTL2 & 0x7F TO SCNTL2
 WAIT DISCONNECT
 JUMP REL(wait_for_reselect)


to_decisions:
 JUMP REL(msg_in_phase),         WHEN MSG_IN
 JUMP REL(cmd_phase),            IF CMD
 JUMP REL(data_in_phase),        IF DATA_IN
 JUMP REL(data_out_phase),       IF DATA_OUT
 JUMP REL(status_phase),         IF STATUS
 INT error_unexpected_phase


;Reselect SCRIPT code
wait_for_reselect:
```

```
 WAIT RESELECT REL(CPU_set_SIGP)

SCSI_id_jump_table:
 MOVE SSID TO SFBR
 JUMP REL(id_0), IF 0x00 ;
 JUMP REL(id_1), IF 0x01
 JUMP REL(id_2), IF 0x02
 INT reselect_id_error

id_0:
 MOVE MEMORY 4, PATCH_addr_of_table0_ptr, PATCH_chip_physaddr+DSA
 ;initialize SXFER for synchronous transfers from table
 MOVE MEMORY 1,PATCH_addr_of_table0+2,PATCH_chip_physaddr+SXFER
 MOVE MEMORY 1,PATCH_addr_of_table0,PATCH_chip_physaddr+SCNTL3
 ; This will set up the clock dividers as defined in the SCNTL3 register
 MOVE FROM identify_msg_buf, WHEN MSG_IN
 CLEAR ACK
 JUMP REL(to_decisions)

id_1:
 MOVE MEMORY 4, PATCH_addr_of_table1_ptr, PATCH_chip_physaddr+DSA
 ;initialize SXFER for synchronous transfers from table
 MOVE MEMORY 1, PATCH_addr_of_table1+2,PATCH_chip_physaddr+SXFER
 MOVE MEMORY 1,PATCH_addr_of_table1,PATCH_chip_physaddr+SCNTL3
 ; This will set up the clock dividers as defined in the SCNTL3 register
 MOVE FROM identify_msg_buf, WHEN MSG_IN
 CLEAR ACK
 JUMP REL(to_decisions)

id_2:
 MOVE MEMORY 4, PATCH_addr_of_table2_ptr, PATCH_chip_physaddr+DSA
 ;initialize SXFER for synchronous transfers from table
 MOVE MEMORY 1,PATCH_addr_of_table2+2,PATCH_chip_physaddr+SXFER
 MOVE MEMORY 1,PATCH_addr_of_table2,PATCH_chip_physaddr+SCNTL3
 ; This will set up the clock dividers as defined in the SCNTL3 register
 MOVE FROM identify_msg_buf, WHEN MSG_IN
 CLEAR ACK
 JUMP REL(to_decisions)


CPU_set_SIGP:
 JUMP scheduler
```

Index

S

T

U

W

Symbios Logic
Sales Locations

For literature on any Symbios Logic product or service,
call our hotline toll-free
1-800-856-3093

North American Sales Locations

Western Sales Area
1731 Technology Drive, Suite 610
San Jose, CA 95110
(408) 441-1080

3300 Irvine Avenue, Suite 255
Newport Beach, CA 92660
(714) 474-7095

Eastern Sales Area
8000 Townline Avenue, Suite 209
Bloomington, MN 55438-1000
(612) 941-7075

12377 Merit Dr., Suite 400
Dallas, TX 75251
(972) 503-3205

92 Montvale Avenue, Suite 3500
Stoneham, MA 02180-3623
(617) 438-0043

30 Mansell Court, Suite 220
Roswell, GA 30076
(404) 641-8001

International Sales Locations

European Sales Headquarters
Westendstrasse 193\III
80686 Muenchen
Germany
011-49-89-547470-0

Asia/Pacific Sales Headquarters
Marina Square, #02-256
No. 6 Raffles Boulevard
Singapore, 0103
011-65-3376323

Printed in the U.S.A.
J25972I
0897-3MH