

# ***9100A Series***

## **9100A-017 Vector Output I/O Module**

P/N 855437

May 1989

©1989, John Fluke Mfg. Co., Inc.  
All rights reserved. Litho in U.S.A.

**FLUKE**

## **LIMITED WARRANTY**

John Fluke Mfg.Co., Inc. (Fluke) warrants your 9100A-017 Vector Output I/O Module to be free from defects in material and workmanship under normal use and service for one (1) year from the date of shipment. Software and firmware products are provided "AS IS." We do not warrant that software or firmware products will be error free, operated without interruption or that all errors will be corrected. This warranty extends to you if you are the original purchaser and does not apply to fuses, batteries, or any product which, in our sole opinion, has been subject to misuse, alteration, or abnormal conditions of operation or handling.

To obtain warranty service, contact a Fluke Service Center or send the product, with the description of the difficulty, postage prepaid, to the nearest Fluke Service Center. Fluke assumes no risk for damage in transit.

Fluke will, at our option, repair or replace the defective product free of charge. However, if we determine that the failure was caused by misuse, alteration, or abnormal condition of operation or handling, you will be billed for the repair. The repaired product will be returned to you, transportation prepaid.

THIS WARRANTY IS EXCLUSIVE AND IS IN LIEU OF ALL OTHER WARRANTIES, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO ANY IMPLIED WARRANTY OF MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OR USE. FLUKE WILL NOT BE LIABLE FOR ANY SPECIAL, INDIRECT, INCIDENTAL, OR CONSEQUENTIAL DAMAGES OR LOSS WHETHER IN CONTRACT, TORT, OR OTHERWISE.

# Contents

---

Section	Title	Page
<b>1.</b>	<b>Introduction .....</b>	<b>1-1</b>
1-1.	THE VECTOR OUTPUT I/O MODULE .....	1-1
1-2.	HIGH SPEED PARALLEL STIMULUS .....	1-1
1-3.	BUS EMULATION .....	1-2
1-4.	TEST VECTOR PATTERN GENERATION .....	1-2
1-5.	USER OPERATIONS .....	1-3
<b>2.</b>	<b>Specifications .....</b>	<b>2-1</b>
2-1.	SPECIFICATIONS .....	2-1
<b>3.</b>	<b>Vector Output Operation .....</b>	<b>3-1</b>
3-1.	INTRODUCTION .....	3-1
3-2.	CLOCK SELECTION .....	3-1
3-3.	VECTOR LOADING .....	3-2
3-4.	VECTOR DRIVING .....	3-2
3-5.	LOOP CONTROL .....	3-2
3-6.	HANDSHAKING .....	3-2
3-7.	STOPPING VECTOR DRIVING .....	3-3
3-8.	TRISTATE CONTROL .....	3-4
3-9.	COMPLETION STATUS .....	3-5
<b>4.</b>	<b>Hardware .....</b>	<b>4-1</b>
4-1.	PHYSICAL DESCRIPTION .....	4-1
4-2.	CONNECTING THE MODULE TO THE MAINFRAME .....	4-5
4-3.	PERFORMING THE MODULE SELF TEST .....	4-6
4-4.	CALIBRATING THE MODULE .....	4-6
4-5.	I/O Module to External and Capture .....	4-7
4-6.	I/O Module to Pod .....	4-8

Section	Title	Page
4-7.	CONNECTING THE MODULE TO THE UUT .....	4-9
4-8.	Using the Clip Modules .....	4-9
4-9.	Connecting the External Lines .....	4-10
4-10.	Using the Card Edge Fixture Kit .....	4-10
4-11.	SYNCHRONIZING THE MODULE TO THE UUT .....	4-11
4-12.	External Sync Mode .....	4-11
4-13.	Pod Sync Mode .....	4-13
4-14.	Freerun Sync Mode .....	4-14
4-15.	Internal Sync Mode .....	4-14
4-16.	Capture Mode .....	4-15
4-17.	SYNCHRONIZING MULTIPLE MODULE VECTORS .....	4-16
<b>5.</b>	<b>TL/1 Functions .....</b>	<b>5-1</b>
5-1.	INTRODUCTION .....	5-1
5-2.	CLOCKFREQ .....	5-1
5-3.	DRIVEPOLL .....	5-1
5-4.	EDGEOUTPUT .....	5-2
5-5.	ENABLEOUTPUT .....	5-3
5-6.	STROBEOUTCLOCK .....	5-3
5-7.	SYNCOUTPUT .....	5-3
5-8.	VECTORDRIVE .....	5-3
5-9.	VECTORLOAD .....	5-3
<b>6.</b>	<b>Using the Vector Editor .....</b>	<b>6-1</b>
6-1.	INTRODUCTION .....	6-1
6-2.	CREATING A NEW VECTOR FILE .....	6-1
6-3.	SOFTKEY OPERATIONS .....	6-2
6-4.	WORKING WITH THE VECTOR FILE .....	6-4
6-5.	The Groups Line .....	6-4
6-6.	The Display Line .....	6-5
6-7.	Comment Lines .....	6-6
6-8.	Data Fields .....	6-6
6-9.	Control Phrases in Data Fields .....	6-11
<b>7.</b>	<b>Applications .....</b>	<b>7-1</b>
7-1.	USING THE VECTOR OUTPUT I/O MODULE .....	7-1
7-2.	BUS EMULATION .....	7-2

Section	Title	Page
7-3.	TESTING THE IBM CGA VIDEO CARD .....	7-4
7-4.	Fixturing the CGA Interface .....	7-5
7-5.	Fixturing Considerations .....	7-6
7-6.	Vector File Programming .....	7-6
7-9.	Vector File Generation .....	7-10
7-10.	CGA Write/Read Program Example .....	7-11
7-11.	68000 BUS VIDEO APPLICATION .....	7-12
7-12.	Testing The Video Board .....	7-12
7-13.	Fixturing the Video Board Interface .....	7-13
7-14.	Determining Bus Cycles .....	7-14
7-15.	Determining the Write Cycle .....	7-14
7-16.	Determining the Read Cycle .....	7-16
7-17.	WAIT APPLICATION .....	7-18
7-18.	TIMING SETS .....	7-19
7-19.	TL/1 VECTOR FILE GENERATION .....	7-21

**Appendices**

<b>A.</b>	<b>New TL/1 Commands .....</b>	<b>A-1</b>
<b>B.</b>	<b>I/O Module Clip/Pin Mapping .....</b>	<b>B-1</b>
<b>C.</b>	<b>Keypad Reference Changes .....</b>	<b>C-1</b>



# Tables

---

Table	Title	Page
2-1.	Vector Output I/O Module Specifications .....	2-1
7-1.	CGA to Vector Output I/O Module Mapping .....	7-5
7-2.	Video Board to Vector Output I/O Module Mapping .....	7-13





# Figures

---

Figure	Title	Page
3-1.	WAIT Input Timing Diagram .....	3-3
3-2.	$\overline{\text{TRISTATE}}$ Input Timing Diagram .....	3-4
4-1.	Vector Output I/O Module Block Diagram .....	4-4
4-2.	Vector Output I/O Module Output Section .....	4-4
4-3.	Input Section Block Diagram .....	4-5
4-4.	Pod Sync Input/Output Section Timing Diagram .....	4-13
4-5.	Capture Clock Timing Diagram .....	4-15
5-1.	Vector I/O Control .....	5-2
7-1.	MC6845 Cycle Timing Diagram .....	7-7
7-2.	SCN2674 Controller Timing Diagram .....	7-15
7-3.	Using WAIT to Check for a Level (Method 1) .....	7-19
7-4.	Using WAIT to Check for a Level (Method 2) .....	7-19
7-5.	Connecting Two Modules for Timing Sets .....	7-20



# Section 1

## Introduction

---

### **THE VECTOR OUTPUT I/O MODULE**

**1.1.**

The Fluke 9100A-017 Vector Output I/O Module adds high speed test stimulus capabilities to the 9100A Digital Test System and 9105A Digital Test Station (referred to in this manual as the Mainframe). The 9100A-017 can generate vectors (parallel patterns) with clock speeds up to 25 MHz for functional testing and troubleshooting. This speed improves card edge stimulus and bus emulation performance for both microprocessor and non-microprocessor bus-based cards.

### **HIGH SPEED PARALLEL STIMULUS**

**1.2.**

The Vector Output I/O Module allows you to drive vectors (parallel patterns) with four different clock modes. Vectors can be clocked out at up to 25 MHz using an external clock source. Each 40-pin module also provides a selectable internal clock generator of 1, 5, 10, or 20 MHz. The clock can be qualified by external lines that control start, stop, and enable. A handshaking line allows additional synchronization between the module and the unit under test (UUT) bus. Each pin can be tri-stated individually for any number of vectors by program control in a vector file. An external tri-state input allows all 40 pins to be tri-stated simultaneously.

Parallel vectors may be up to 8192 bits deep. With 40 pins per module, four modules may be used together for a total width of

160 pins. The ability to loop on a set of vectors from 1 to 65536 times makes the maximum effective depth of the test vector 536,870,912 patterns.

The Vector Output I/O Module input response capabilities are identical to the 9100-003 Parallel I/O Module. Responses may be gathered at clock speeds up to 10 MHz for signature analysis and frequency/count measurement.

## **BUS EMULATION**

**1.3.**

The high speed vector generation allows the module to emulate many common busses. An external input, WAIT, can be used to synchronize the output vectors to bus cycle events. Separate drive and receive clocks allows bus cycles to be simulated with many drive patterns, and data input can be sampled on command.

## **TEST VECTOR PATTERN GENERATION**

**1.4.**

The generation of the test vector patterns can be accomplished in two ways: using the 9100A Editor or downloading from another system.

The 9100A Editor has been enhanced to effectively enter test vectors for the 9100A-017. Vectors can be entered in user-programmable groups in hexadecimal notation, binary notation, or a mixture of both. The vector editor displays the vector information in the order set by the pin group assignments. Each vector is displayed in its position in the driving sequence.

Test vectors generated on a CAE workstation can be downloaded into the 9100A. Once they are in the 9100A, these vectors can then be translated to the 9100A vector format, making them suitable for use with the 9100A-017.

Several new Test Language (TL/1) functions have been added to make use of the new capabilities of the 9100A-017. Loading the vectors from the 9100A hard disk to each module during program execution requires a maximum of 3 seconds, with 1 second typical.

## USER OPERATIONS

### 1.5.

Users may access the new Vector Output I/O Module through functions provided in the Operator's Interface, TL/1, and the Editor.

From the Operator's Interface, all the input functions of the original 9100A-003 Parallel I/O Module are still available for the new Vector Output I/O Module. In addition, another soft key, CAPTURE, has been added to the SYNC I/O MOD operation. The user may now select CAPTURE as the input measurement clock. This clock synchronizes the vector output section with the input section (this clock is user-programmable and is generated by the vector file).

To create a vector file using the 9100A Editor, you need to edit a file of the new type, VECTOR, which has been added under each UUT directory. The file type should be specified as VECTOR when editing the file.

Eight new TL/1 built-in functions have been added to the 9100A Test Language (TL/1) to support the new Vector Output I/O Module. These functions are:

- Clockfreq
- Drivepoll
- Edgeoutput
- Enableoutput
- Strobeoutclock
- Syncoutput
- Vectordrive
- Vectorload

In addition, other TL/1 built-in functions have been modified for use with the new Vector Output I/O Module.

The Vector Output I/O Module is designed for high-speed output applications in which overdrive capabilities are not required. If overdrive capabilities are a requirement, use the 9100A-003 Parallel I/O Module.



# Section 2

# Specifications

---

## SPECIFICATIONS

## 2.1.

Table 2-1 contains the specifications for the Vector Output I/O Module.

*NOTE*

*Output specifications for Table 2-1 were obtained using the Y9100-102 Card Edge Interface Module into 10 LSTTL loads. Results may vary depending on the impedance, length, and shielding of the connector used. Output timing is measured at 50% of signal amplitude. Typical input signal transition times: clocks 5 ns, all others 12 ns.*

**Table 2-1. Vector Output I/O Module Specifications**

<b>VECTOR OUTPUT I/O MODULE OUTPUT SECTION (into 10 LS TTL loads with card edge connector attached):</b>	
Module Vector Size .....	8192 Vectors 40 Channels Wide
Maximum Vector Pattern (4 Modules).....	8192 Vectors 160 Channels Wide
Vector Looping .....	Up to 65536 Repetitions of One Vector Set
Output Logic Levels:	
High.....	3.7V minimum (6.0 mA Source)
Low.....	0.4V maximum (6.0 mA Sink)
INT CLK (internal dock) .....	1, 5, 10, or 20 MHz ( $\pm 100$ ppm).
DR CLK (external dock) .....	25 MHz Maximum. (This may be exceeded in some cases based upon application and hardware interfacing.)

Table 2-1. Vector Output I/O Module Specifications (cont)

Clock to Vector Out (tdel):	
INT CLK Out to Vector Out Delay .....	37 ns typical, 45 ns maximum.
DR CLK in to Vector Out Delay.....	50 ns typical, 58 ns maximum.
WAIT (Handshake) Setup Time (twsu) .....	42.5 ns maximum (35 ns typical)
	from WAIT acknowledgement until next dock cycle drives vector. If the setup time is not met, the next clock drives out the vector. Minimum WAIT pulse width is 10 ns.
Single Module Channel to Channel Skew * .....	6 ns Maximum (1 ns typical).
Module to Module Channel Skew * .....	10 ns Maximum (1 ns typical).
TRISTATE:	
Activation (txout) .....	Output source/sink released 25 ns <u>maximum</u> (20 ns typical) after <u>TRISTATE</u> goes low. Minimum <u>TRISTATE</u> pulse width is 10 ns.
Recovery (txsu).....	TRISTATE must go high no later than 5 ns after the rising edge of the INT CLK or no later than 10 ns after the clocked edge of DR CLK for the vector to be output by that dock, otherwise that vector is only driven internally and the output is held tri-stated, effectively skipping that vector.
Output Series Termination .....	33 Ohms
Capture Clock: **	
INT CLK .....	Capture Clock clocks 42.5 ns $\pm$ 5 ns after the falling edge of INT CLK.
DR CLK.....	Capture Clock clocks 55 ns $\pm$ 10 ns after non-clocking edge of DR CLK (approximate 50% duty cycle).
START, STOP, and ENABLE:	
START, STOP pulse width .....	10 ns minimum
INT CLK	
START Setup Time .....	30 ns minimum
STOP Setup Time .....	30 ns minimum
ENABLE Setup Time.....	25 ns minimum
ENABLE Hold Time.....	20 ns minimum
DR CLK	
START Setup Time .....	20 ns minimum
STOP Setup Time .....	20 ns minimum
ENABLE Setup Time.....	15 ns minimum
ENABLE Hold Time.....	35 ns minimum
Input Impedance:	
DR CLK.....	40 K $\Omega$ minimum, 35 pF maximum.
TRISTATE .....	40 K $\Omega$ minimum, 80 pF maximum.
WAIT .....	40 K $\Omega$ minimum, 50 pF maximum.
* Skew measurement assumes equal loading. Differences in capacitance may affect results.	
** Capture clock may be adjusted in approximate 15 ns steps by using the <i>setoffset</i> command (see the 9100 Series TL/1 Reference Manual).	



**Table 2-1. Vector Output I/O Module Specifications (cont)**

**VECTOR OUTPUT I/O MODULE INPUT SECTION:**

Input Impedance ..... 50 KΩ Minimum, 90 KΩ typical. 100 pF Maximum, 65 pF typical. \*

Operating Voltage Range..... -0.5V to +5.5V (all lines)

Input/Output Protection ..... +10V/-5V for one minute max. one line only (all lines).

Input Thresholds:

TTL	CMOS	LEVEL
5.0V	5.0V	Guaranteed HIGH
2.6V	3.4V	HIGH or INVALID
2.1V	2.9V	Guaranteed INVALID
1.0V	1.2V	LOW or INVALID
0.6V	0.8V	Guaranteed LOW
0.0V	0.0V	

CLOCK, START, STOP, and ENABLE Inputs:

Thresholds:

Logic LOW..... 0.8V maximum.

Logic HIGH..... 2.0V minimum.

Input Current ..... 125 μA. maximum

Input/Output Protection ..... +10V/-5V for one minute maximum, one line only.

Transition Counter:

Maximum Frequency..... 10 MHz minimum.

Maximum Count (Transition Mode) ..... 8388608 (23 bits) counts (+ overflow).

Frequency Accuracy (Frequency Mode) ..... ±250 ppm ±2 Hz

Stop Counter:

Maximum Frequency..... 10 MHz

Maximum Count ..... 65535 clocks

Clock:

Maximum Frequency..... 10 MHz

Minimum Pulse Width ..... 50ns

Timing for Synchronous Measurements:

Maximum Frequency of Clock..... 10 MHz

Maximum Frequency of Data ..... 5MHz

Data Setup Time ..... 30 ns minimum

Data Hold Time ..... 30 ns minimum

Minimum Pulse Width (Start/Stop/Enable/Clock) ..... 50ns

Start Edge Setup Time (before clock edge, for clock to be recognized) ..... 0 ns minimum

\* Input capacitance includes the Y9100A-102 Card Edge Interface Module.

**Table 2-1. Vector Output I/O Module Specifications (cont)**

Stop Edge Setup Time (before clock edge, for clock edge to not be recognized).....	5 ns minimum
Enable Setup Time (before clock edge, for clock edge to be recognized).....	0 ns minimum
Enable Hold Time (after clock edge, for clock edge to be recognized).....	10 ns minimum
Data Timing for Asynchronous Measurements:	
Maximum Frequency .....	10 MHz
Minimum Pulse Width (HIGH or LOW) .....	50 ns
Minimum Pulse Width (tri-state).....	150 ns
Data Compare Equal:	
Minimum Pulse Width of Data and Enable .....	75ns

**PHYSICAL SPECIFICATIONS**

## Operating Temperature:

5 to 27°C, 95% RH maximum (non-condensing).

27 to 40°C, RH decreasing linearly from 95% to 50% (non-condensing).

## Storage/Shipping Temperature:

20 to 60°C, 8% to 80% RH, (non-condensing).

# Section 3

## Vector Output Operation

---

### INTRODUCTION

3.1.

This section describes the general functions of the Vector Output I/O Module. Each of these functions are covered more fully in the Sections that follow.

### CLOCK SELECTION

3.2.

The Vector Output I/O Module can be synchronized to a number of different clock sources, both external and internal.

There are several methods of synchronizing the Vector Output I/O Module to the UUT. To synchronize vector driving with a UUT clock, the DR CLK input of the Vector Output I/O Module can be connected to a UUT clock source. The *syncoutput* command would then be set to “drclk”. To synchronize response gathering with a UUT clock, connect the Vector Output I/O Module CLOCK input (yellow lead) to the UUT clock source and set the *sync* command to “ext”.

To synchronize the UUT to the Vector Output I/O Module, connect the UUT clock input to INT CLK and set the *syncoutput* command to “intfreq”. Use the *clockfreq* command to select either 1, 5, 10, or 20 MHz as the clock rate. If the delay between INT CLK and the vector data output is too long, dedicate one output line in the 40-pin connector of the Vector Output I/O Module as the clock input to the UUT and program that line (in the vector file) to perform as a clock.

To synchronize response gathering, connect the Vector Output I/O Module input CLOCK line to the INT CLK output and set the sync command to “ext”. Another method would be to use the TL/1 *sync* “capture” mode. This method allows you to program clocks into the vector file to determine when data is gathered in the input section of the I/O Module.

## **VECTOR LOADING 3.3.**

The vector file to be output is loaded by the *vectorload* command into the designated module(s) and resides in the module until changed by a *writeword* or *writepin* command directed to that module, or by another *vectorload*.

## **VECTOR DRIVING 3.4.**

Vector driving has two modes of operation. In the first mode of operation, vector driving can begin immediately upon receipt of the proper start, enable, and clock signals. In the second mode of operation, driving begins after the execution of the TL/1 *arm* command and upon the receipt of the proper start, enable, and clock signals. The second mode permits the simultaneous starting of vector driving and response gathering. The vector number (in the current vector file) that the driving begins on can also be selected.

## **LOOP CONTROL 3.5.**

To drive a certain pattern of vectors more than once within a single file, use the LOOP and ENDLOOP commands within a vector file to control the number of times and the bounds of the pattern to be repeated. The number of loops must be set between 1 and 65536. At least 2 vectors must appear between the LOOP and ENDLOOP commands.

## **HANDSHAKING 3.6.**

The vector file WAIT command enhances synchronization of the Vector Output I/O module to the UUT. This command suspends vector driving until the programmed edge (+ or -) transition has occurred on the WAIT input pin (see Figure 3-1). Once the edge

transition has occurred, vector driving resumes on the next qualified clock. The edge polarity cannot be changed from one polarity to the other within the vector file.

## STOPPING VECTOR DRIVING

### 3.7.

Vector driving is stopped if the STOP statement is reached in a vector file, if the programmed edge on the external STOP line is received, or if a *writeword* or *writepin* command is performed during driving. Vector driving is suspended if the WAIT statement has been reached within the vector file, if the external ENABLE condition set by *enableoutput* is not satisfied, or if no clock for the appropriate *syncoutput* mode is received.

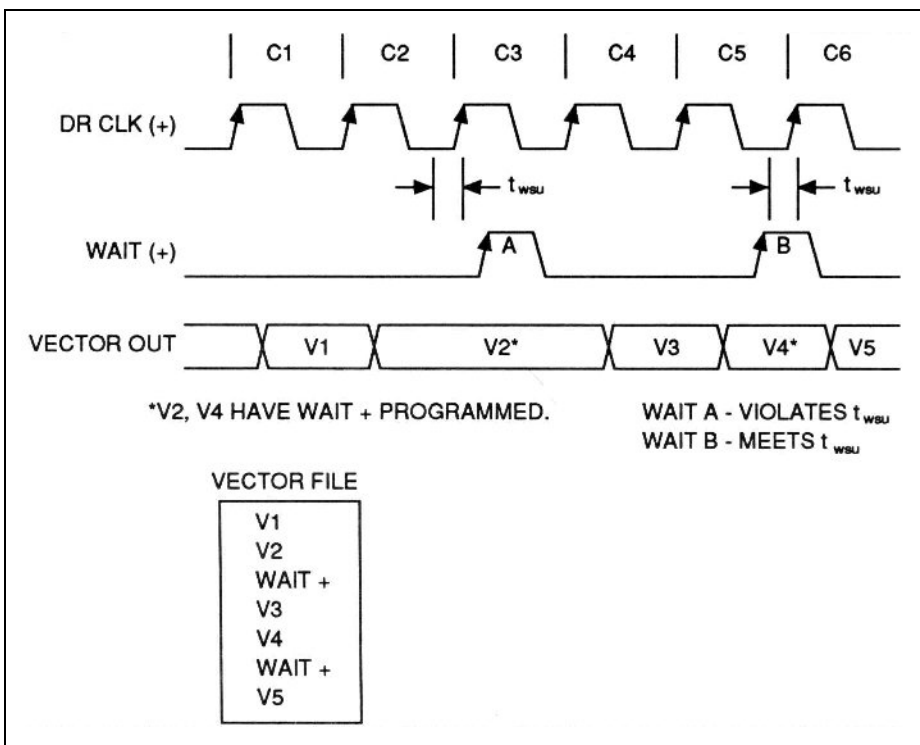


Figure 3-1. WAIT Input Timing Diagram

## TRISTATE CONTROL

### 3.8.

A low signal on the  $\overline{\text{TRISTATE}}$  input line of the Vector Output I/O Module asynchronously tri-states the 40-pin output of the module. The output of the module remains tri-stated after the removal of the  $\overline{\text{TRISTATE}}$  low signal until a *writeword* or *writopin* command has been executed or further vector driving has occurred.

If the  $\overline{\text{TRISTATE}}$  occurred during vector driving, the proper recovery time ( $t_{\text{XSU}}$ ) must be met to have the next vector driven out upon receipt of a clock (see Figure 3-2 and Table 2-1 in Section 2 of this manual). If the recovery time is not met, the output of the module remains tri-stated and the current vector is effectively skipped. To prevent vectors from being skipped, connect the external ENABLE line to the  $\overline{\text{TRISTATE}}$  pin and use the *enableoutput* mode “high”.

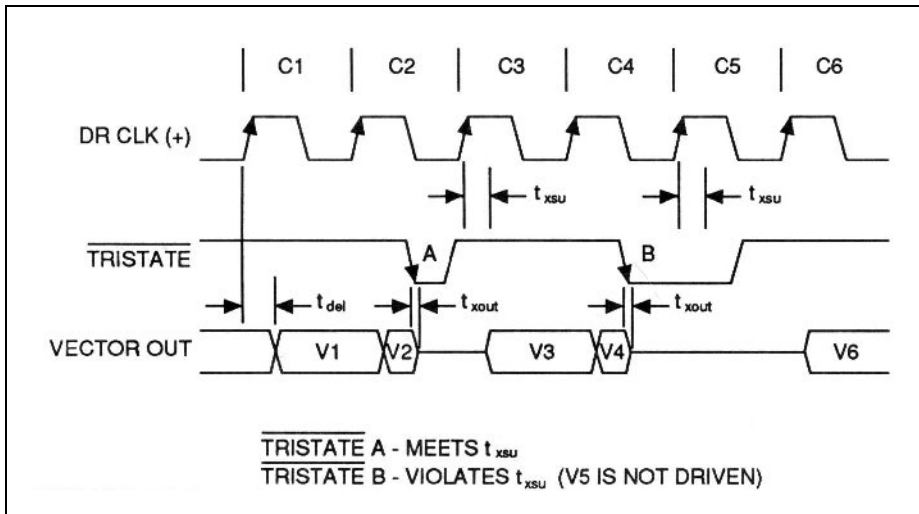


Figure 3-2.  $\overline{\text{TRISTATE}}$  Input Timing Diagram

One method of using the external tri-state feature would be when the Vector Output I/O Module is connected to a peripheral port on a bus of a UUT. The peripheral select line could be connected to the  $\overline{\text{TRISTATE}}$  input of the module. When the port is not in use, the peripheral select line would tri-state the module.

## COMPLETION STATUS

### 3.9.

The *drivepoll* command determines the completion status of vector driving. Two bits are returned from a *drivepoll* command: Bit 1 indicates if vector driving is currently suspended while awaiting the programmed WAIT edge (0 for waiting, 1 for not waiting) and Bit 0 indicates if the vector has been driven out until a STOP statement in the vector file has been reached (0 for not done, 1 for done).

When the TL/1 *syncoutput* mode is “intfreq” or “drclk”, vector driving is terminated but not completed if the programmed edge of the external STOP line is received. Other *syncoutput* modes are not affected by the STOP line.





# Section 4

## Hardware

---

### PHYSICAL DESCRIPTION

#### 4.1.

The 9100A-017 Vector Output I/O Module connects to the 9100A/9105A Mainframe through a round shielded cable, and connects to the UUT or fixture through a variety of I/O Module clip modules and measurement control lines.

All of the I/O Module clip modules used with the 9100A-003 I/O Module also work with the Vector Output I/O Module. I/O Module clip modules plug into the Vector Output I/O Module at one end and interface with the UUT or fixture at the other end. One or two I/O Module clip modules (depending on size) can plug into a Vector Output I/O Module at a time. Each I/O Module clip module contains a pushbutton (called a ready button) and a common clip. When the ready button is pressed, it signals the system to start a measurement cycle. The common clip should be connected to UUT common.

At one end of the Vector Output I/O Module case is a set of leads labeled CLOCK, START, STOP, ENABLE, and COMMON (the CLOCK line is for input only, all other lines are input and output). These lines are used for external synchronization of the module. External synchronization first requires a Start signal edge (either rising or falling) to be input on the START lead. Once this has been received, each active external clock edge (on the CLOCK lead), during which the Enable input (on the ENABLE lead) is asserted, triggers a measurement, until a Stop signal (on the STOP lead) occurs, the number of enabled clock edges specified has occurred, or a TL/1 *readout* command has

been executed. The COMMON lead should always be connected to the UUT except during self test and calibration.

On the right side of the Vector Output I/O Module is a 6-pin connector. These pins perform the following functions:

- DR CLK

A user-supplied clock that can be selected to drive out the vectors. This input can be clocked at up to 25 MHz (the signal must conform to the specifications listed in Section 2). Either a rising or falling edge can be selected as the clocking edge (using the *edgeoutput* command). This input can synchronize the vectors with a UUT, synchronize multiple modules to the same clock, or provide a custom clock speed for an application.

- TRISTATE

A user-supplied tri-state input signal. When an active low signal is present on this input, all 40 outputs of the Vector Output I/O Module are tri-stated immediately (see the specifications in Section 2 of this manual for activation time *txout*). TRISTATE must go high no later than the recovery time *txsu* after the clocking edge of INT CLK or DR CLK for the vector to be output. Otherwise, that vector is only driven internally and the output is held tri-stated, effectively skipping that vector.

- WAIT

A user-supplied handshake input signal used in association with the vector file WAIT command. This pin synchronizes vector output communication between the UUT and the Vector Output I/O Module. When a WAIT command occurs in a vector file, no new vectors are driven until the proper signal edge is received on this input. The WAIT command in the vector file determines which edge of the signal on this pin is considered valid (see Section 6 of this manual for more information on the vector editor WAIT command).

- COMMON

System common. This pin should be connected to the UUT if any of the other pins on this part of the Vector Output I/O Module are used.

- INT CLK

The internal clock specified by “intclk” in the *syncoutput* command is output on this line. The speed of the output signal on this pin is set to 1, 5, 10, or 20 MHz using the *clockfreq* command. The default value for *clockfreq* is 1 MHz. If a clock other than “intclk” is specified in the *syncoutput* command, the signal that is output from this pin remains the same as the last *clockfreq* command specified (or 1 MHz if at default). This output is a free-running clock.

- DCE

The Data Compare Equal output. When a pattern is present on the input pins that is equal to the pattern set using the *compare* command or the front panel SET I/O MOD x COMPARE command, this pin is asserted high. This pin can be used as hardware notification that a DCE has occurred.

The Vector Output I/O Module common line is fuse protected. If the line is accidentally connected to a UUT power supply when a pod is also connected to the UUT, the fuse will blow. If the fuse is blown, either a message appears on the operator's display or the TL/1 fault “iomod\_fuse\_blown” is invoked to indicate that the fuse needs replacing.

Figure 4-1 shows a block diagram overview of the Vector Output I/O Module. The output section of the module is further illustrated in Figure 4-2. A block diagram of one channel of the receive section of the module is shown in Figure 4-3.

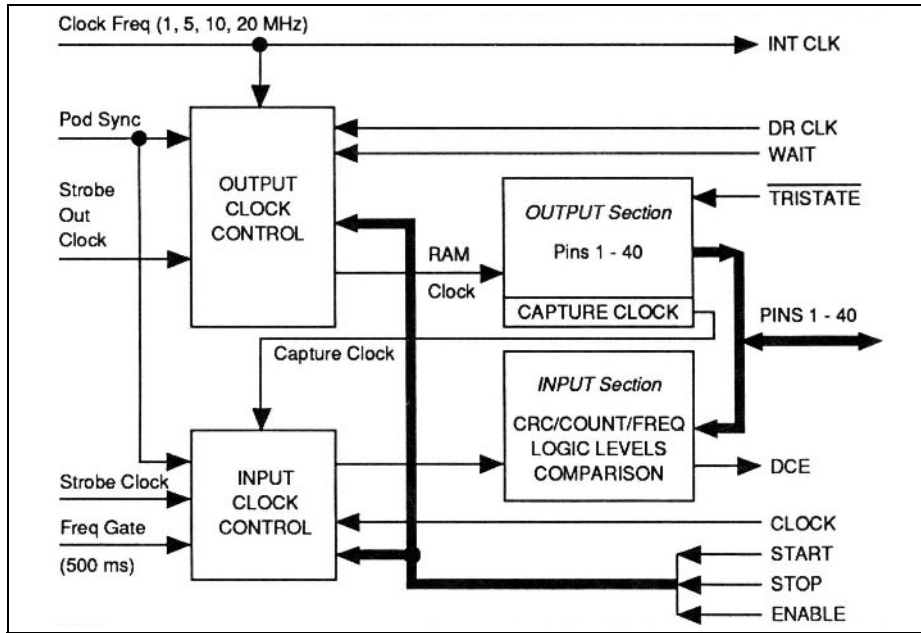


Figure 4-1. Vector Output I/O Module Block Diagram

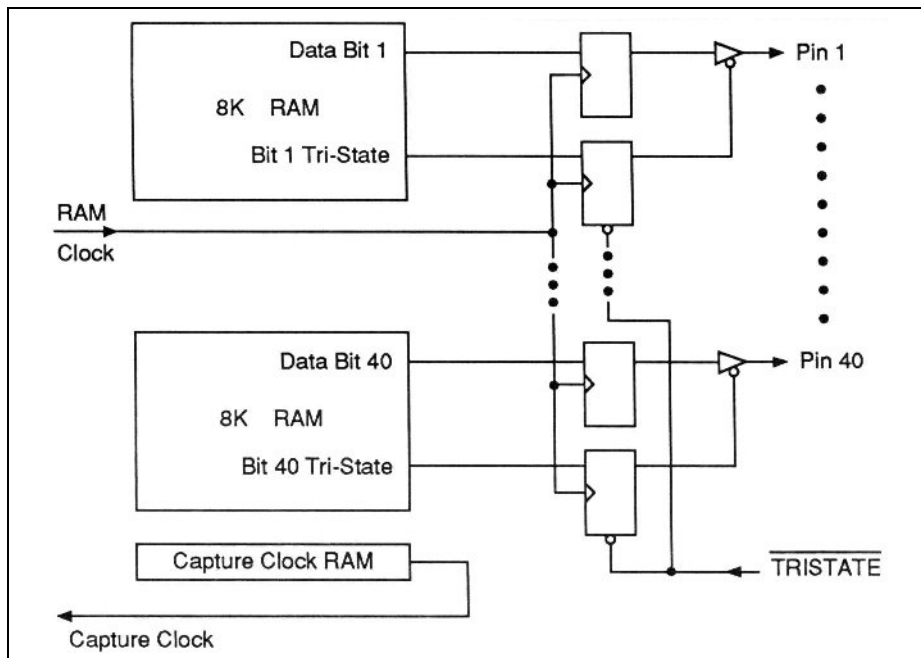


Figure 4-2. Vector Output I/O Module Output Section

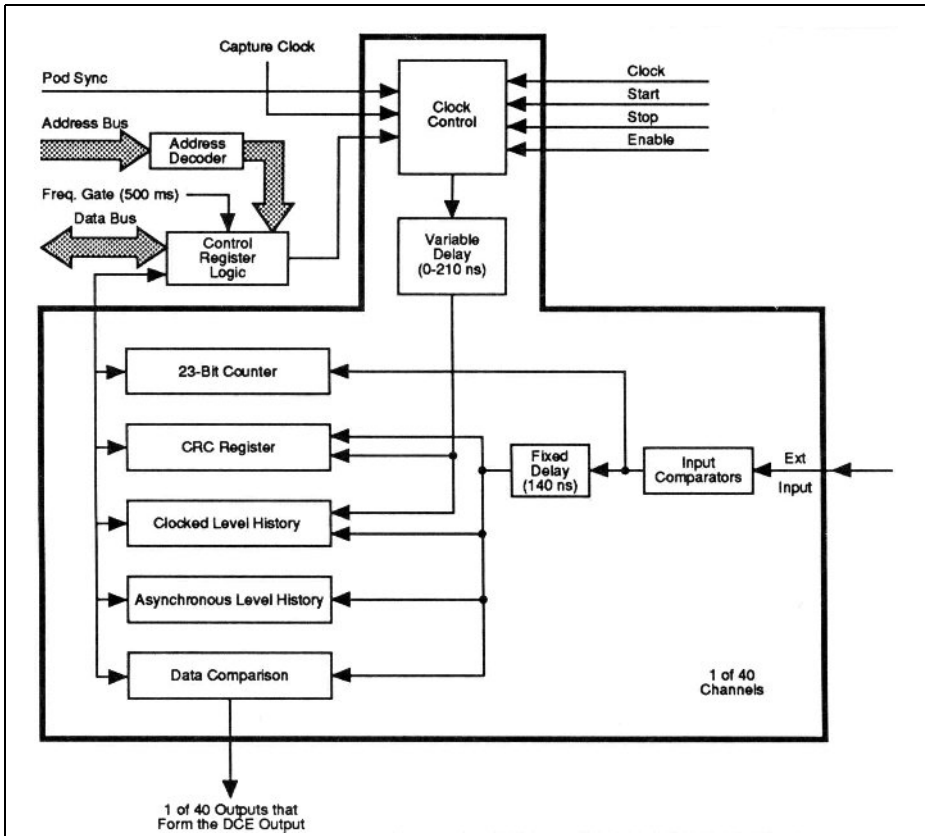


Figure 4-3. Input Section Block Diagram

## CONNECTING THE MODULE TO THE MAINFRAME 4.2.

Before performing the Vector Output I/O Module self test or calibration, connect the module to the 9100A/9105A Mainframe as follows:

1. Check that the Mainframe is OFF.
2. Connect the module's 37-pin connector to one of the four I/O Module connectors found on the back of the Mainframe.

## PERFORMING THE MODULE SELF TEST

### 4.3.

The Vector Output I/O Module self test determines if the selected module(s) is connected to the Mainframe and is communicating with the system. Perform the following steps to begin the self test:

1. Remove any clip modules from the Vector Output I/O Module.
2. Press the MAIN MENU key on the Mainframe keypad and use the left arrow key to move the cursor to the left-most field.
3. Press the SELFTEST softkey.
4. Move the cursor to the next field using the right arrow key and press the I/O MOD softkey.
5. Move the cursor to the right and select the module(s) to be tested by either entering a number between 1 and 4 or entering each of the numbers of the modules connected (i.e., 1234). For example, to self test Vector Output I/O Module 2, the display should read:

```
MAIN: SELFTEST I/O MOD 2
```

6. Press the ENTER key. If the self test fails, a failure message is displayed on the Mainframe. If this happens, check the Vector Output I/O Module connection and repeat the test, carefully following any instructions. If the self test passes, the display reads:

```
MAIN: SELFTEST COMPLETE
```

## CALIBRATING THE MODULE

### 4.4.

Calibration is the process of determining individual variations in the time delays between the clock path and the data path of the input section so the delays can be compensated.

Calibration should be done when the system is first set up and at regular intervals (at least monthly). Calibration is also necessary whenever devices attached to the system are changed or repaired.

Once the calibration is complete, the calibration data can be saved and restored when needed. Restoration should be done after each power-up or reset before testing or troubleshooting a UUT. For more information on saving and restoring calibration data, see the heading, *An Alternative to Calibration*, in Section 4 of the *9100 Series Technical User's Manual*.

## I/O Module to External and Capture

### 4.5.

This calibration determines the proper setting for the internal delay between the clock and data. This routine performs the necessary calibration for both the External and Capture sync modes. Perform the following steps to calibrate the module:

1. Press the MAIN MENU key and use the left arrow key to move the cursor to the left-most field.
2. Press the CAL softkey.
3. Move the cursor to the right and press the I/O MOD softkey.
4. Move the cursor to the right and press the EXT softkey. The Mainframe display reads:

```
MAIN: CAL I/O MOD TO EXT
```

5. Press the ENTER key. The display reads:

```
MAIN: CAL I/O MOD TO EXT
      INSTALL CAL HEADER IN DESIRED I/O MODULE
      PRESS BUTTON WHEN READY
```

6. Plug the calibration module into the Vector Output I/O Module to be calibrated.

7. Press the ready button on the calibration module. When the calibration is complete, the BUSY light on the Mainframe goes off and the display reads:

```
MAIN: CALIBRATION COMPLETE
```

## I/O Module to Pod

## 4.6.

This calibration calculates the proper settings for the Vector Output I/O Module's internal clock delay for use with the SYNC I/OMODTOEXT, SYNC I/O MOD TO CAPTURE, and SYNC I/O MOD TO POD command. When any of these commands are entered, the appropriate delay is selected. This calibration requires the use of the calibration module that is supplied with the Vector Output I/O Module. Use the following steps to perform the I/O Module to Pod calibration:

1. Connect a UUT to the pod.
2. Press the MAIN MENU key and use the left arrow key to move the cursor to the left-most field.
3. Press the CAL softkey.
4. Move the cursor to the right and press the I/O MOD softkey.
5. Move the cursor to the right and press the POD softkey.
6. Move the cursor to the right and press the desired softkey. For example, if you want to use the Mainframe in SYNC I/O MOD TO POD ADDR mode, press the ADDR softkey. The display then reads:

```
MAIN: CAL I/O MOD TO POD ADDR
```

7. Press the ENTER key. The display reads:

```
MAIN: CAL I/O MOD TO POD ADDR  
INSTALL CAL HEADER IN DESIRED I/O MODULE  
PRESS BUTON WHEN READY
```



8. Plug the calibration module into the Vector Output I/O Module and press the calibration module's ready button. (The calibration lead on the calibration module must be unconnected before pressing the ready button.)
9. After a few seconds, a pod-dependent message is displayed. For example, the display may read:

```
COMPLETED EXT CAL PRIOR TO CAL POD
NOW CONNECT CAL LEAD TO ~S1
PRESS BUTTON WHEN READY
```

10. Refer to a schematic of your UUT and locate the specified signal, in this case, ~S1. At a suitable point on the UUT, attach the calibration lead to this signal. Also attach the Vector Output I/O Module COMMON lead to common on the UUT.
  11. Press the calibration module's ready button. After several seconds, the display reads:
- ```
MAIN: CALIBRATION COMPLETE
```
12. Repeat steps 6 through 11 for each SYNC mode in which the pod is to be operated.

## **CONNECTING THE MODULE TO THE UUT** **4.7.**

### **Using the Clip Modules** **4.8.**

The same clip modules can be used with the 9100A-017 Vector Output I/O Module as are used with the 9100A-003 Parallel I/O Module. The clip modules therefore connect to the UUT in the same way as with the Parallel I/O Module. However, because the Vector Output I/O Module runs at a much greater speed, some of the following measures may need to be implemented:

- When receiving, all clip modules perform correctly for measuring stimuli. However, the longer the cable length used with the clip module, the more capacitance is added to the circuit. If this additional capacitance creates problems

with the UUT, try connecting the clip to another point in the circuit or provide a buffer between the UUT and the clip.

- When vector driving, capacitance associated with the clip modules may alter the driving specifications slightly (all driving specifications were obtained using the Y9100A-102 Card Edge Interface Module). As the output speed increases, there may be increasing waveform degradation. The flying lead set (John Fluke model number Y9100-20L) is susceptible to overshoot at higher speeds and should not be used in systems with system clocks that run at more than a few megahertz. Because the Vector Output I/O Module has no overdrive capability, care should be used in selecting where the clip is placed.

## Connecting the External Lines

### 4.9.

The DR CLK,  $\overline{\text{TRISTATE}}$ , WAIT, INT CLK, DCE, and COMMON lines may be connected to a UUT using cables with the module end terminated with a 0.64 mm (0.025 in.) square pin receptacle. As the signal speed increases, good quality cabling (such as ribbon cable, twisted pair, or coax cabling) should be utilized to preserve signal integrity on the DR CLK,  $\overline{\text{TRISTATE}}$ , WAIT, INT CLK and DCE lines. For some applications, a fixture may require buffering to maintain signal integrity.

## Using the Card Edge Fixture Kit

### 4.10.

The Card Edge Fixture Kit (John Fluke model number Y9100A-100) provides a simple means to interface the 9100 test system to boards with card-edge connectors. The kit enables up to four Vector Output I/O Modules to be connected to a card edge.

To provide easy connection from the UUT to the Card Edge Fixture Kit, the Card Edge Interface Module is also recommended. This module allows direct connection of all 40 pins from the Vector Output I/O Module, along with pins for the external START, STOP, CLOCK, ENABLE, and COMMON measurement control lines. The INT CLK, WAIT,  $\overline{\text{TRISTATE}}$ , DCE, DR CLK, and COMMON lines from the Vector Output

I/O Module may be substituted for any of the measurement control line pins on the fixture kit performance card. Application circuitry may also be added to the performance card in the fixture to provide buffering, dynamic RAM control circuitry, or for other purposes.

The Card Edge Fixture Kit manual (John Fluke part number 830349) contains further application and fixturing information that is not only useful for the fixture kit, but for general applications as well.

## **SYNCHRONIZING THE MODULE TO THE UUT**

### **4.11.**

Synchronization of the Vector Output I/O Module input section is accomplished either by pressing the Mainframe SYNC key or using the TL/1 sync command. The SYNC command allows you to specify the source of the clock signal used with the Vector Output I/O Module input section. For the Module, this clock gathers synchronous data.

For the Vector Output I/O Module, there are five sync modes:

- External Sync.
- Pod Sync.
- Freerun Sync.
- Internal Sync.
- Capture.

## **External Sync Mode**

### **4.12.**

The External Sync Mode qualifies the external CLOCK line with the external START, STOP, and ENABLE lines. Start, Stop, and Clock are edge-sensitive inputs from the UUT. Each can be made to respond to falling or rising edges. The sync period can also be programmed to end after a specified number of valid clock pulses, in which case, the Stop input is ignored. Enable is a level sensitive signal and can be specified as valid HIGH, LOW, or ALWAYS. When the sync enabling condition is set to ALWAYS, the Enable input is ignored. One additional mode allows the valid Pod Sync (POD) time to be used as the enable condition.

*NOTE*

*The enabling condition need not always depend on the specified state (HIGH, LOW, ALWAYS) of the Enable input. The enabling condition can also involve the state of the internal PodSync line.*

After an ARM I/O MOD command is entered and after a valid Start edge is detected, the measurement period begins. Start is recognized independently of the enabling condition, but data is only gathered after the enabling condition becomes true. Synchronous data is gathered after the same point, but only at the selected clock edges.

The data gathering period ends when the Stop condition becomes true: the selected Stop edge occurs, or a programmed number of clock edges completes. After the period ends, enter the appropriate SHOW I/O MOD command (found under the Mainframe I/O MOD key), which displays data gathered by the following:

- CRC signature register.  
  
Contains signature data clocked in at each enabled clock edge between Start and Stop.
- Clocked level history register.  
  
Contains all logic levels (1, 0, X) seen at each enabled clock edge between Start and Stop.
- Asynchronous level history register.  
  
Contains all logic levels (1, 0, X) seen while enabled between Start and Stop. An X level must be present for a fixed period before being recorded because an X should not be recorded during fast 0-to-1 or 1-to-0 transitions, which do pass briefly through the invalid (X) region.

- Transition counter.

Records each rising transition at the input, between Start and Stop, while enabled. A rising transition is either 0-to-1 or X-to-1.

For more information, see the sync command in Section 3 of the *TL/I Reference Manual* and the SYNC key in the *Technical User's Manual* keypad reference section.

## Pod Sync Mode

### 4.13.

This mode uses an internal pod signal as the clock. This signal (shown in Figure 4-4) is generated by the pod at each pod cycle for the selected addressing mode (i.e., READ, WRITE). Its generation can be made to depend on valid address, data, or other (pod-dependent) cycles. External lines (START, ENABLE, and CLOCK) are ignored.

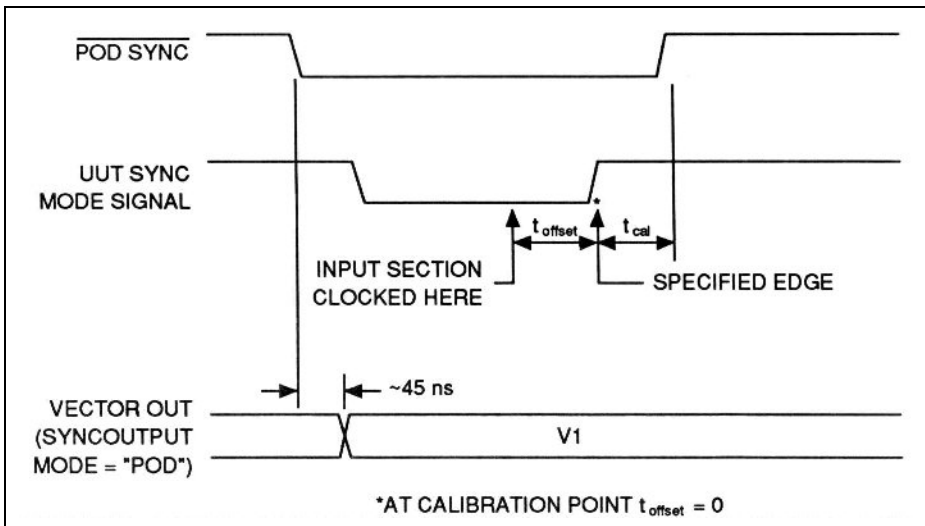


Figure 4-4. Pod Sync Input/Output Section Timing Diagram

Data is gathered after an ARM I/O MOD command. Synchronous data is gathered until a SHOW I/O MOD command, which displays data gathered by:

- CRC signature registers.

- Clocked level history registers.
- Asynchronous level history registers.
- Transition counter.

The data is gathered as described previously in External Sync Mode, except that the CRC signatures and clocked level history are clocked by the internal pod signal.

For more information, see the *sync* TL/1 command in Section 3 of the *TL/1 Reference Manual* and the SYNC key in the *Technical User's Manual* keypad reference section.

## **Freerun Sync Mode**

**4.14.**

This sync mode is more usefully invoked under TL/1 program control, rather from the keypad.

In this mode, the Vector Output I/O Module gathers asynchronous level history and transition count data between the TL/1 arm and readout commands. CRC signatures and clocked level histories are not gathered.

For more information, see the *sync* TL/1 command in Section 3 of the *TL/1 Reference Manual* and the SYNC key in the *Technical User's Manual* keypad reference section.

## **Internal Sync Mode**

**4.15.**

This sync mode uses a clock signal generated internally by a program. It is therefore more usefully invoked under TL/1 program control. The external START, ENABLE, and CLOCK lines are ignored.

For more information, see the *sync* TL/1 command in Section 3 of the *TL/1 Reference Manual* and the SYNC key in the *Technical User's Manual* keypad reference section.

## Capture Mode

### 4.16.

The capture sync mode synchronizes the response gathering hardware with the vector output automatically. When the vector files are edited, vectors can be programmed to clock the input section on a single vector, specific vectors, or all vectors. For more information, refer to Section 6 of this manual for the capture command description.

After a vector that has been programmed to capture is driven, the receive hardware is clocked (see Figure 4-5). When *syncoutput* mode is “intfreq” or “drclk”, the receive hardware is clocked at about the midpoint of a single vector period (for example, when a 1 MHz clock is used, a vector period is 1000 ns and clocking occurs 500 ns after the vector is driven). (For “drclk”, a 50% clock duty cycle is assumed.)

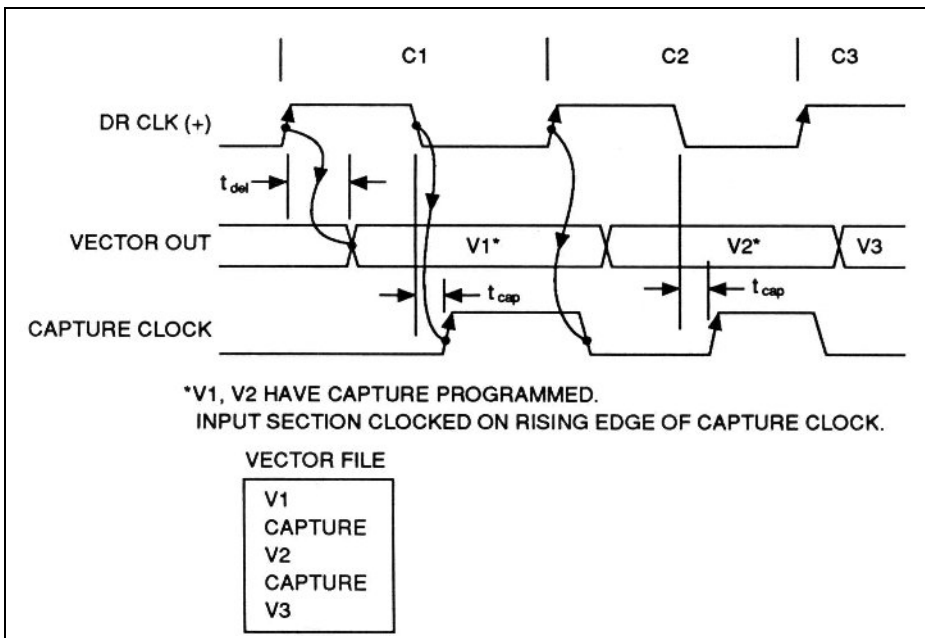


Figure 4-5. Capture Clock Timing Diagram

For all *syncoutput* modes, capture clock performs without any user consideration. When the *syncoutput* mode is “drclk”, the clock cycle should be close to a 50% duty cycle to get receive clocking in the center of the driven vector. A 50% duty cycle

becomes critical as clock speeds increase above 5 Mhz. This duty cycle ensures that the the minimum clock pulse width of 50 ns is maintained. If the clock does not have a duty cycle close to 50%, the external sync mode is recommended instead of the capture mode for high speeds.

When capture clock is used, performing the external calibration procedure (described earlier in this section) is recommended. If a UUT has considerable propagation delay, the TL/1 *setoffset* command can adjust for this. (The sync mode must be set to “capture” prior to using the *setoffset* command.)

## **SYNCHRONIZING MULTIPLE MODULE VECTORS 4.17.**

To synchronize the output vectors of multiple modules, the same clock source must be applied to the DR CLK input of all the modules that are to be synchronized. The INT CLK clock (selectable at 1, 5, 10, or 20 MHz) may be used as the clock source by connecting it to the DR CLK input of all modules. An external clock at any frequency may also be used (external clock signals must meet the specifications listed in Section 2).

As the speed of the clock increases, high-speed signal techniques must be used to ensure signal integrity. A connector made of high-quality cabling with each set of signal and ground lines the same length gives good results. In some cases, providing a separate buffered clock signal to each DR CLK input may be necessary.



# Section 5

## TL/1 Functions

---

### INTRODUCTION

5.1.

Eight new functions have been added to the 9100A Test Language (TL/1) to support the new Vector Output I/O Module. This section briefly describes each of these new functions. See Appendix A for a complete description of the functions. Figure 5-1 illustrates the relationship between the TL/1 commands and the Vector Output I/O Module operations they affect. (Note that the “storepatt”, “writepatt”, and “clearpatt” functions are not used with the Vector Output I/O Module.)

### CLOCKFREQ

5.2.

The *clockfreq* function controls which internal clock (1, 5, 10, or 20 MHz) is selected. The change is immediate upon command execution and is reflected on the output pin INT CLK. It has no effect on the clock driving the vectors unless the vector driving is synchronized to the “intfreq” mode. Changing the *clockfreq* while vector driving is active and synchronized to the “intfreq” mode is not recommended.

### DRIVEPOLL

5.3.

The *drivepoll* function returns two pieces of information about vector driving status when driving vectors. Bit 0 indicates whether the vector pattern being driven has reached the end of the pattern. If the vector pattern has not reached its end, Bit 1 indicates if vector driving is suspended while waiting for a WAIT handshake.

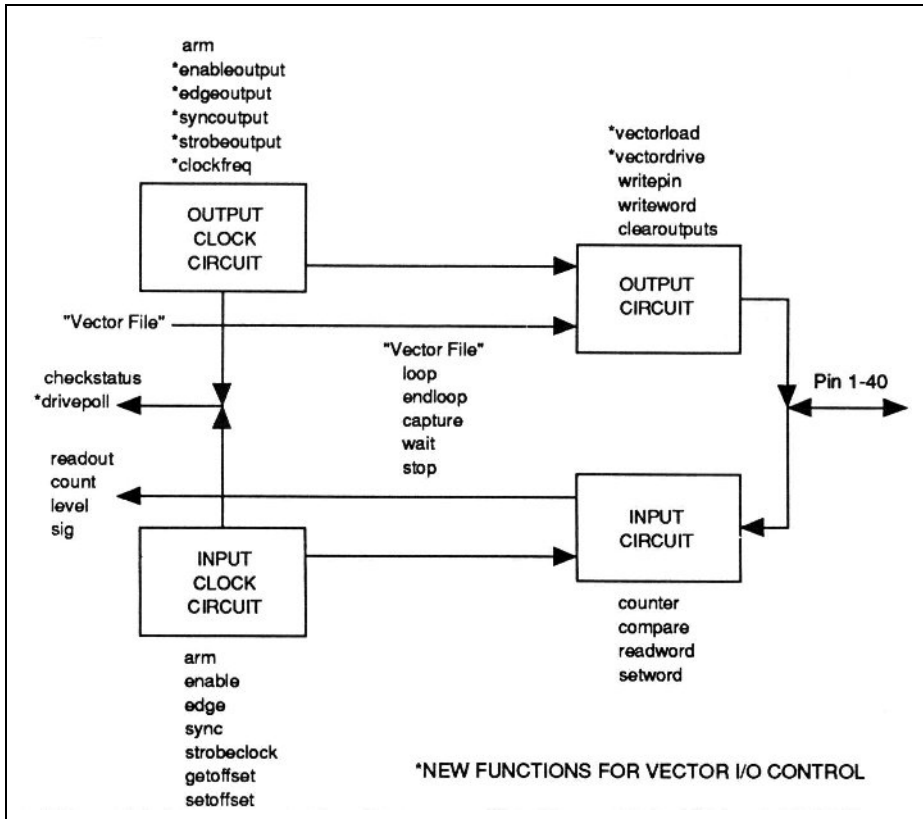


Figure 5-1. Vector I/O Control

*NOTE*

*If an external stop signal has been received and is enabled, the vector driving terminates, but drive-poll does not indicate vector driving as complete.*

**EDGEOUTPUT**

**5.4.**

The *edgeoutput* function sets the signal edges for triggering the START, STOP, and DR CLK lines. Note that the START and STOP lines are in common with the receive side of the module, but can have separate edge control. The START edge can also be set to “at\_vectordrive”, which means that no external start signal is required to begin driving out vectors.

**ENABLEOUTPUT****5.5.**

The *enableoutput* function controls the enabling (or qualifying) of the vector drive clock. If the enable is not received, the vector count remains at the current vector regardless of the clocks received until the clock is enabled. Only *syncoutput* modes “drclk” and “intfreq” are affected by this command.

**STROBEOUTCLOCK****5.6.**

The *strobeoutclock* function provides a software controlled strobe for single-step vector driving when the *syncoutput* mode is set to “int”. This is useful for debugging, troubleshooting, or for applications where other TL/1 functions must be used between vector outputs.

**SYNCOUTPUT****5.7.**

The *syncoutput* function determines the clocking source for driving out the vector patterns. Available clocking sources are “drclk”, “intfreq”, “int”, and “pod”.

**VECTORDRIVE****5.8.**

The *vectordrive* function enables vector driving. The start and enable conditions must be met to drive the vectors out. There are two startmodes associated with vector driving: “now” and “at\_arm”. The “now” argument immediately enables the module for vector pattern driving. The “at\_arm” argument does not enable the vector pattern driving until the TL/1 *arm* command has been executed. The “at\_arm” mode ensures that both the receive and drive sides are started simultaneously. The user can also indicate which vector within the pattern he wishes to begin driving on.

**VECTORLOAD****5.9.**

The *vectorload* function loads the named vector file into the declared module(s). Once a module is loaded with a vector file, the file resides in the module and does not require reloading unless the user has performed a *writeword* or *writepin* command (which overwrites the vector file).



# Section 6

## Using the Vector Editor

---

### INTRODUCTION

6.1.

To be able to drive vectors out the 9100A-017 Vector Output I/O Module, a vector file must be created. This file contains all the information required by the module to output the proper sequence of signals. When used in conjunction with the various signal inputs and outputs of the Vector Output I/O Module, control of the vector patterns and handshaking between the module and the UUT is greatly enhanced.

To use the Vector Editor, you must be familiar with the basic operations of the 9100A Editor. Operation of the 9100A Editor is covered in the *9100 Series Programmer's Manual*.

### CREATING A NEW VECTOR FILE

6.2.

To create a new vector file, press the EDIT key, enter the path name (for example, “/hdr/tmp/vec1”) followed by <CR>, and press the FIELD SELECT key until VECTOR is displayed in the TYPE field. Press <CR>. When the file is created, the following text is displayed:

```
GROUPS [40-1]
DISPLAY BIN

!TIME DATA
  1
```

Whenever a vector file is created, the text at the top of the file is automatically produced. When loading a previously created file, nothing is added to the file.

Once the file is loaded, use the standard editor functions to enter new data. For example, to create a vector file that outputs zeroes to all 40 lines, then outputs ones to all 40 lines, position the cursor on the last character of the last line, and type in “0” 40 times. Go to the second line and type “2” followed by 40 “1”s. Press the F10 softkey (Check) followed by the F2 key (Save). Press the QUIT key. You have just created your first vector file.

## SOFTKEY OPERATIONS

### 6.3.

When you first enter a vector screen, you’ll notice a row of soft keys at the bottom of the screen that provide additional operations during editing. The following list describes the function of each of the softkeys:

- F1 - Goto

Allows you to go to a specific line of the vector file. (This softkey takes you to a specified line number, and not the number of the vector).

- F2 - Save

Checks the vector file to see that no syntax errors are present and then saves the vector file to disk.

- F3 - Renum

Automatically renumbers the sequence of vector numbers. This key can be used during the course of entering a new or editing an existing file.

- F4 - Mark

Marks the beginning of a line or character (i.e., as in Mark/CUT/PASTE). See the *9100 Series Programmer’s Manual* for more information.

- F5 - Cut

Deletes the marked block from the display and moves it into one of the temporary buffers. This softkey does not appear until a Mark has been set. See the *9100 Series Programmer's Manual* for more information.

- F6 - Yank

Copies the marked block into one of the temporary buffers, but does not remove the block from the display. This softkey does not appear until a Mark has been set. See the *9100 Series Programmer's Manual* for more information.

- F7 - Paste

Copies the contents of one of the temporary buffers into the display just before the cursor position. See the *9100 Series Programmer's Manual* for more information.

- F8 - Replace

Moves the cursor to and replaces the next occurrence of a character string you have specified. See the *9100 Series Programmer's Manual* for more information.

- F9 - Search

Moves the cursor to the next occurrence of a character string you have specified. See the *9100 Series Programmer's Manual* for more information.

- F10 - Check

Performs an overall check of the vector file by searching for syntax errors that cannot be detected by the line-checker. See the *9100 Series Programmer's Manual* for more information.

## WORKING WITH THE VECTOR FILE

### 6.4.

When a vector file is first created, four lines of text are automatically created at the beginning of the file. These four lines are:

```
GROUPS [40-1]
DISPLAY BIN

!TIME DATA
1
```

The first line is called the groups line. The second line is called the display line. The third line is a comment line. The fourth line is the beginning of the data fields. The following paragraphs describe these lines in detail.

Blank lines are only allowed after the display line. If, during editing, a blank line is inserted between the groups line and the display line, an error message is displayed.

### The Groups Line

### 6.5.

The first line of the display is the groups line. The groups line determines how the pins of the Vector Output I/O Module are to be grouped together. All 40 pins on the module must be listed in this group. This line may be changed to group together the 40 pins using the following syntax:

```
GROUPS <label>[<group>]...
```

where GROUPS is the name of the line, <label> is an optional label made of any ASCII characters except “[”, and <group> is any number or range of numbers from 1 to 40 (in any order) separated by a comma (,), a space ( ), or a comma and a space (,)(,). If more than 2 numbers are listed in a row, the numbers can be shown using the beginning and ending number separated by a dash (-). For example, pins 1, 2, 3, and 4 on the module can be listed as 1-4.

More than one <label> and <group> pair can be listed on the group line. For example, if you wanted to group together pins 1



through 4 and pin 6, and put the rest of the pins in another group, the groups line would be changed to read:

```
GROUPS [1-4,6] [5,7-40]
```

Note that the groups line in the previous example did not use the optional <label>. If, however, you want to add a label that designates pins 1 through 4 and 6 as control lines, the groups line could be changed to read:

```
GROUPS control [1-4,6] [5,7-40]
```

If the second group consists of address lines, the groups line could be changed to read:

```
GROUPS control [1-4,6] address [5,7-40]
```

Any combination of numbers from 1 to 40 in any order can be listed on the <group>. A <group> consists of a minimum of one number (i.e., [24]) and a maximum of forty numbers (i.e., [40-1]). All forty pins of the Vector Output I/O Module must be listed. For example, if all the <group>s consisted of one number, forty <group>s would be listed on the groups line.

## The Display Line

## 6.6.

The display line specifies whether the data shown in the data fields are displayed in binary or hexadecimal. When a vector file is first created, the display line is shown as:

```
DISPLAY BIN
```

This designates that all forty pins in the data field are displayed in binary. If you want all the pins in the data fields to be displayed in hexadecimal, change the display line to read:

```
DISPLAY HEX
```

If more than one group exists in the groups line, each of the groups can be displayed individually as either binary or hexadecimal. A comma or a comma and a space is used to separate the designators. For instance, if there are two groups

and you want the first group to be displayed in hexadecimal and the second group to be displayed in binary, you would change the display line to read:

```
DISPLAY HEX, BIN
```

If more than one group exists in the groups line and both are to be displayed as hexadecimal, only one designator is necessary. The vector editor automatically assumes that any group not identified by an individual designator is the same as the last identified designator.

If there are more designators on the display line than there are groups in the groups line, any extra designators are ignored.

The designators are case insensitive. The display line could be changed to read:

```
DISPLAY hex, HeX, BIN
```

## **Comment Lines**

**6.7.**

Comment lines can be added to the vector file on any line after the groups and display line. All comment lines must begin with a "!". Comment lines must contain only comments (i.e., no comments can be added to the end of a data field).

## **Data Fields**

**6.8.**

Data fields consist of the data that is to be sent out the Vector Output I/O Module pins. In addition, a set of special phrases are available for looping, capturing input data, pausing until a specific signal is received, and stopping the vector driving.

Data fields are composed of 1s, 0s, and Xs in either binary or hexadecimal notation. "1" denotes an active high signal, "0" denotes an active low signal, and "X" denotes a tri-state signal.

Using the first four lines that appear when the vector file is created, a data field may be entered that looks like:

```
GROUPS [40-1]
DISPLAY BIN

!TIME DATA
1 100000001000100010000000001111111110111
```

The data field in the previous example must be 40 characters long and is entered in binary format. Each binary number corresponds to the data sent out the Vector Output I/O Module. If the binary number you enter is less than 40 characters long, the numbers are right justified and zeroes fill in the missing numbers to the left. If you enter more than 40 numbers, only the right-most 40 numbers you enter are placed on the data field line. All other numbers are discarded. If you try to enter anything but a 0, 1, or X into a binary data field, an error message is displayed.

The first field on the data field is the line number. These numbers are used by the TL/1 *vectordrive* command to determine where in the vector file to begin driving vectors. The line numbers must be in the range of 1 through 8192. If a number outside this range is used, an error message is displayed. The first data field has its number automatically supplied when a vector file is created. All other line numbers must be entered by the user. These numbers do not have to be in any order, and repeated numbers do not cause an error. Once the program is saved, however, all of the lines are automatically renumbered in ascending order. The order in which they are displayed on the screen is not changed. For example, if:

```
2 XX
2 11
1 00
8000 XXXX00001111XXXX00001111XXXX00001111XXXX
```

is saved, the lines are automatically renumbered to:

```
1 XX
2 11
3 00
4 XXXX00001111XXXX00001111XXXX00001111XXXX
```

Press the F3 (RENUM) key to renumber the lines while editing.

To fill a data field line when there is more than one group on the groups line, you must enter the data with a space separating the groups. For example, the following vector file contains two groups that are separated by a space:

```
GROUPS [40-9] [8-1]
DISPLAY BIN

!TIME DATA
  1 01000000000100000000011111111101 1111XXXX
```

Since both of the data fields in the previous example are binary, the display line did not need to be changed. However, you could have changed it to read:

```
DISPLAY BIN, BIN
```

and the data fields would remain the same.

If you did not enter a space on data field line in the previous example, all of the numbers in the second group would have been automatically tri-stated. For example, if you entered:

```
1111
```

the first field would have been right justified and filled in with zeroes and the second field would have been tri-stated:

```
1 0000000000000000000000000000000001111 XXXXXXXX
```

If the display line is changed so that the second data field is a hexadecimal number, the data that you enter on the data field line is also expected to be in hexadecimal notation (0 through F). For example, if the first field in the previous example was changed to hexadecimal, and you enter the following data:

```
401007FD 1111XXXX
```

the editor would display the file as:

```
GROUPS [40-9] [8-1]
DISPLAY HEX, BIN

!TIME DATA
  1 $401007FD                1111XXXX
```

The “\$” sign that was added to the data field line indicates a hexadecimal number. The second field is still in binary notation. The space between the first and second field is the space necessary to expand the hexadecimal field to a binary field.

If the display line designates that a group is displayed in hexadecimal format and you need to tri-state one pin of the Vector Output I/O Module, you must use the “\*” special character to enter the data on the data field line. The “\*” character indicates that the data field is normally hexadecimal, but is being displayed in binary. For example, if you wanted pin 9 to output a tri-state condition in a hexadecimal data field, you would enter the data line as:

```
*010000000001000000001111111110X 1111XXXX
```

The line on this field will continue to be displayed as a binary representation in the hexadecimal field as long as the “\*” is left in place. If the “\*” character is removed, the data field converts to hexadecimal representation, taking only the rightmost numbers that will fit in the field and discarding the rest (this effectively changes the data in the field). If the “\*” character is removed from line 1 in the previous example, the line would change to:

```
1 $11111110X                1111XXXX
```

If the “\$” sign was removed and the “\*” character was placed at the beginning of the field, the line would expand out to:

```
1 *00000000000000000000000011111110X 1111XXXX
```

If the groups line or the display line are modified after data fields have been entered into the file, the data fields change to match the modifications. For example, if the following vector file was loaded:

```
GROUPS [40-9] [8-1]
DISPLAY HEX, BIN

!TIME DATA
  1 $401007FD          1111XXXX
  2 $FFFFFFFF          0000XXXX
  3 $00000000          1100XX11
```

and the groups line was changed to read:

```
GROUPS [40-13] [12-1]
```

the vector file would change to:

```
GROUPS [40-13] [12-1]
DISPLAY HEX, BIN

!TIME DATA
  1 $401007F          11011111XXXX
  2 $FFFFFFF          11110000XXXX
  3 $0000000          00001100XX11
```

If the display line was then changed to read:

```
DISPLAY BIN, HEX
```

the vector file would change to:

```
GROUPS [40-13] [12-1]
DISPLAY BIN, HEX

!TIME DATA
  1 0100000000010000000001111111 $DFX
  2 1111111111111111111111111111 $FOX
  3 0000000000000000000000000000 *00001100XX11
```

On line 3, the binary representation in the second field could not be converted to a hexadecimal representation, so the “\*” character was added to the beginning of the field to retain its binary representation.

## Control Phrases in Data Fields

## 6.9.

A set of special control phrases are available for performing specialized functions within the data fields. The following list describes these control phrases:

Loop <n>  
Endloop

The Loop <n> phrase begins a loop. The loop count <n> may be set to any number between 1 and 65536. The Endloop phrase indicates the end of the loop. Only one loop/endloop pair is allowed for each vector file. There must be at least 2 data fields (i.e., vectors) between a loop/endloop pair. The loop count <n> value determines the number of times the vectors between loop and endloop are driven. When the loop count <n> is set to 1, the effect is the same as if the loop/endloop pair was not used.

Stop

The Stop phrase stops vector driving after the vector on the previous line. If this phrase is positioned just before an endloop, stop acts as if the phrase was entered after the endloop line type. For example:

```
stop
endloop
```

is the same as:

```
endloop
stop
```

Capture  
Capture On  
Capture Off

The Capture phrase causes the input section to be clocked on the opposing edge of the selected output clock (which causes the clocking to occur near the center of the preceding vector). Only one capture should be used for a single data field (i.e., you can place two capture statements in a row, but only one will be performed). The Capture On/Capture Off pair can be placed in a file where clocking the input section for all data fields contained between Capture On and Capture Off is required. If INTFREQ, INT, or POD are the selected clocks, an input capture is guaranteed. If DRCLK is selected, the clock supplied by the user must have an opposing edge to assure that an input capture occurs.

Wait <x>

The Wait phrase pauses vector driving until the appropriate rising or falling edge is detected on the Vector Output I/O Module WAIT line before vector driving continues. <x> may be specified as + for rising edges and - for falling edges. <x> can only be set to one edge type in a vector file. If more than one edge type is specified in the file, an error message is generated.

If the vector file contains a loop/endloop pair and you start driving vectors from a data field within that loop using the TL/1 *vectordrive* command, vectors are driven until the count reaches the endloop statement, and then starts the loop again on the vector just after the loop command. For example, if your vector file looks like:

```
GROUPS [40-1]
DISPLAY BIN

TIME DATA
  1 1111111000111110000000000000111010001110
LOOP 20
  2 11
```



```
3 1111110001101010100001010010101010000111
4 00
5 11
ENDLOOP
```

and the *vectordrive* command specifies that the file begins driving on data field 4, the vectors are driven out in the order - 4, 5, 2, 3, 4, 5, 2, etc.



# Section 7

## Applications

---

### USING THE VECTOR OUTPUT I/O MODULE

### 7.1.

The Vector Output I/O Module can be used in a wide variety of applications, from simple pattern stimulus to performing bus cycles such as read and write cycles. Regardless of the application, five main steps are required to fully use the capabilities of the module. These are:

1. Create the vector files using the editor or another source.
2. From TL/1, use a program to setup the output section.
3. From TL/1, use a program to setup the input section.
4. From TL/1, use a program to drive vectors and measure responses.
5. From TL/1, use a program to process the results.

Section 6 “Using the Vector Editor” illustrated how to make vector files. (To make vector files that perform bus cycles, see “Bus Emulation”, “Testing the IBM CGA Video Card”, and “68000 Bus Video Application” further on in this section.)

The following example shows a typical program with the four steps necessary for setting up the output and input sections, driving vectors and measuring responses, and processing the results.

```

*****
program example

  devname - "/mod1"
  reset device devname

  setword device devname, word 1, as_pins "1 2 3 4 5 6 7 8 9 10 11 12 13 14
  15 16 17 18 19 20 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39
  40"

  low_word = "00"

! *****Setup Output Section*****
writeword device devname, word 1, patt low_word      ! initialize outputs
  ! before loading file
! vec_hilo is 40 pins wide 8192 vectors deep alternating hi and lo
vectorload device devname, file "/hdr/vector/vec_hilo"

clockfreq device devname, freq "1MHz"
syncoutput device devname, mode "intfreq"

edgeoutput device devname, start "at_vectordrive" ! stop don't care
  ! clock is always "+"
  ! for syncoutput=
  ! "intfreq"

enableoutput device devname, mode "always"

! *****Setup Input Section*****
counter device devname, mode "transition"
sync device devname, mode "capture"      ! in "capture" mode start is forced
   ! at arm, clock edge is fixed,
   ! stop don't care

! *****Drive and Measure Responses*****
arm device devname
  vectordrive device devname, startmode "now", vector 1
  loop until (drivepoll device devname) = 3  ! ensure driving complete
  end loop
  readout device devname

! *****Process the Results*****
for pinnum = 1 to 40
  alvl = level device devname, pin pinnum, type "async"
  slvl = level device devname, pin pinnum, type "clocked"
  cnt = count device devname, pin pinnum
  crc = sig device devname, pin pinnum
  ! if not expected results, display them
  if ((alvl <> 5) or <slvl <> 5) or (cnt <> 4096) or (crc <> $BC67)) then
    print "Pin" + str (pinnum,10) + "ALVL = " + str (alvl,10) +
          " SLVL - " + str (slvl,10) + " CNT " + str (cnt,10) +
          " CRC = " + str (crc, 16)
  end if
next

clearoutputs device devname
end program
*****

```

## BUS EMULATION

## 7.2.

The Vector Output I/O Module can provide bus-cycle emulation for many popular buses for testing non-microprocessor based UUTs.

To obtain the simplest level of emulation, the signal timing required to perform a read and a write cycle should first be determined. Once the timing is determined, data and/or address information that is required can be inserted on the bus. Timing diagrams from a microprocessor or UUT controller specification sheet typically provide this information.

A logic analyzer may also be useful for determining cycle information. The logic analyzer can be connected to a known-good system with the UUT plugged into a motherboard so the cycles can be observed. The analyzer can also provide other information (such as initialization sequences, handshaking protocols, etc.) that may be necessary to perform bus communication.

There are two methods by which a read cycle may be implemented:

- For read cycles where individual data reads are not important or not possible (such as block reads), but the block of data must be correct, a cyclic redundancy check (CRC) can be used to verify the data. When CRC checking is used, read cycles occur at full speed with no delays between cycles.
- For read cycles where the individual data must be observed, a TL/1 *readword* function must be performed after each cycle. By using the “capture” sync mode and placing the CAPTURE keyword in the appropriate vector in the vector file, or by using the “ext” sync mode and connecting the external clock line (CLOCK) to an address strobe or data strobe, the *readword* function can read the “stored” synchronous level history. This method allows read cycles at full speed with some delay between read accesses.

Data Compare Equal (DCE) can trap asynchronous events on the bus, or can detect bus errors or interrupts if a line(s) of the Vector Output I/O Module is dedicated for monitoring the event.

For applications where the Vector Output I/O Module provides a clock to the UUT for synchronization and the skew between INT CLK and the data out is too large, the vector data output can be synchronized with the clock by dedicating an output pin of the module to be the UUT clock.

## TESTING THE IBM CGA VIDEO CARD

### 7.3.

The Vector Output I/O Module can test bus-based UUTs that do not have an on-board microprocessor. The key to testing this type of board is to determine the vector patterns necessary to perform the required bus cycles.

In the following example, the Vector Output I/O Module is used to control an IBM PC<sup>\*</sup> CGA Video Card. There are four main areas to be tested on the CGA Video Card: the CRT Controller, the Video RAM, the Video Character ROM, and the Video Output. The basic test strategy for each of the main areas is defined as:

- CRT Controller

The controller to card-edge interface is checked and the controller's ability to control the video in all modes is verified.

- Video RAM

The video RAM is tested by filling it with various patterns and reading back the stored information.

- Video Character ROM

Various character codes are loaded into video RAM and the pattern of the ROM output characters is verified.

- Video Output

The video RGBI output is displayed for user pattern inspection and the 9100A Probe and Clock Module are used to take CRCs of the R, G, B, and I outputs.

---

\* IBM and IBM PC are registered trademarks of International Business Machines Corporation

## Fixturing the CGA Interface

### 7.4.

Table 7-1 maps the pins and signal names of the CGA Card with the pins of the Vector Output I/O Module.

To accomplish the fixture design and vector bus cycle programming, you must use the specifications for the MC6845 CRT Controller and the information on the CGA Card contained in the IBM Technical Reference Manual.

**Table 7-1. CGA to Vector Output I/O Module Mapping**

| Pin | Signal | Module Pin | Pin | Signal                      | Module Pin |
|-----|--------|------------|-----|-----------------------------|------------|
| A1  |        |            | B1  | GND                         |            |
| A2  | D0     | 13         | B2  | RESET                       | 7          |
| A3  | D1     | 14         | B3  | +5V                         |            |
| A4  | D2     | 15         | B4  |                             |            |
| A5  | D3     | 16         | B5  |                             |            |
| A6  | D4     | 17         | B6  |                             |            |
| A7  | D5     | 18         | B7  |                             |            |
| A8  | D6     | 19         | B8  |                             |            |
| A9  | D7     | 20         | B9  | +12V                        |            |
| A10 | IORDY  | 8/WAIT     | B10 | GND                         |            |
| A11 | AEN    | 5          | B11 | <u>MEMW</u>                 | 3          |
| A12 | A19    | 40         | B12 | <u>MEMR</u>                 | 4          |
| A13 | A18    | 39         | B13 | <u>IOW</u>                  | 2          |
| A14 | A17    | 38         | B14 | <u>IOR</u>                  | 1          |
| A15 | A16    | 37         | B15 |                             |            |
| A16 | A15    | 36         | B16 |                             |            |
| A17 | A14    | 35         | B17 |                             |            |
| A18 | A13    | 34         | B18 |                             |            |
| A19 | A12    | 33         | B19 |                             |            |
| A20 | A11    | 32         | B20 | IOCLK<br>(6 MHz)            | 6/DRCLK    |
| A21 | A10    | 31         | B21 |                             |            |
| A22 | A9     | 30         | B22 |                             |            |
| A23 | A8     | 29         | B23 |                             |            |
| A24 | A7     | 28         | B24 |                             |            |
| A25 | A6     | 27         | B25 |                             |            |
| A26 | A5     | 26         | B26 |                             |            |
| A27 | A4     | 25         | B27 |                             |            |
| A28 | A3     | 24         | B28 |                             |            |
| A29 | A2     | 23         | B29 | +5V                         |            |
| A30 | A1     | 22         | B30 | 14 MHz<br>(14.31818<br>MHz) |            |
| A31 | A0     | 21         | B31 | GND                         |            |

## Fixturing Considerations

### 7.5.

Fixturing between the Vector Output I/O Module and the CGA Card is minimal. To keep line lengths as short as possible and to keep interfacing easy, the Y9100A-102 Card Edge Interface Module is recommended. Most lines from the card edge module to the CGA card-edge connector match except for the following:

- IOCLK

To provide the 6 MHz bus timing clock, a 24 Mhz Oscillator is divided by four using a 74LS74 D Flip-Flop (that is placed on the Performance board in the Card Edge Fixture Kit). The clock is connected to pin B20 of the CGA connector, and to pin 6 and DR CLK (to synchronize the vectors with the bus) on the Vector Output I/O Module.

- IORDY

The IORDY line is connected to pin A10 of the CGA connector, and pin 8 and WAIT (to provide Video RAM synchronization) of the Vector Output I/O Module. A pullup resistor is added to this line on the performance board of the Card Edge Fixture Kit.

- 14 MHz

An external signal generator provides the 14.31818 MHz signal required for Video Sync.

- +5V, +12V, GND

An external power supply provides the required voltages to a DPDT switch that controls the power to the CGA Card.

## Vector File Programming

### 7.6.

To control the CGA card, the proper vector patterns to perform read and write bus cycles must be determined and placed in vector files. There are two sets of these cycles on the CGA card: one set to communicate with the MC6845 CRT Controller and another to access the video RAM.



## MC6845 CRT CONTROLLER BUS CYCLES

7.7.

Examine the timing diagram for the MC6845 (shown in Figure 7-1) to determine the vector patterns necessary to achieve the required bus cycles.

To perform a read/write cycle, the following conditions must be met:

- The address (RS) and read/write ( $\overline{R/\overline{W}}$ ) signal must be set a minimum of 80 ns before the rising edge of the E clock and held for 10 ns after the falling edge of the clock.
- WRITE DATA must be set a minimum of 165 ns before the falling edge of the E clock and held for 10 ns after the falling edge of the clock.
- READ DATA must be output a maximum of 290 ns after the rising edge of the E clock and will be held for a maximum of 50 ns after the falling edge of the E clock.

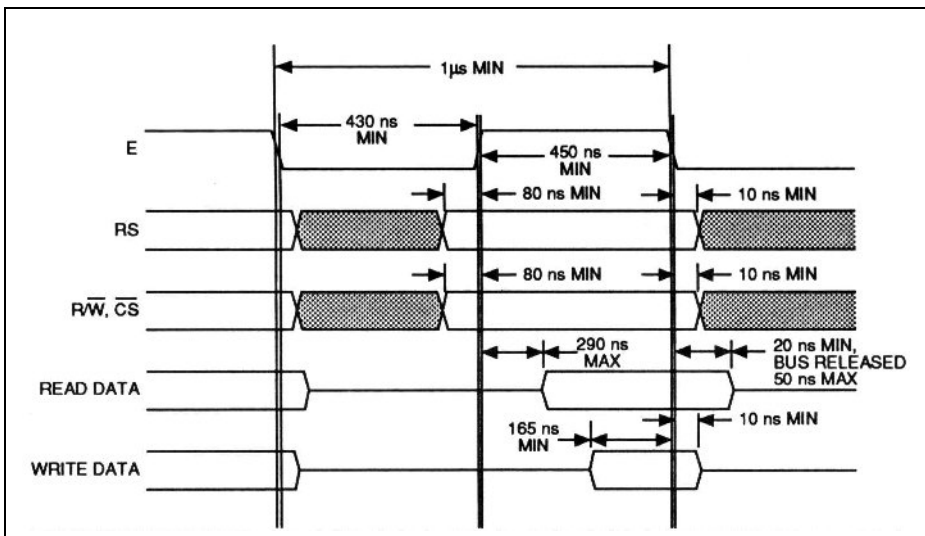


Figure 7-1. MC6845 Cycle Timing Diagram

The following CRT Controller Write Cycle Vector File contains the vectors necessary to perform the write cycle. Since the driven vectors are synchronized with the 6 MHz bus clock, each

vector period is 167 ns. Vector 1 sets the address and control data. Vector 2 is driven out 167 ns later.  $\overline{IOW}$  is set low so that, on the next clock cycle, the E clock will be set high. To ensure that the E clock remains high for the required time, vector 3 and 4 are driven. Vector 5 adds the data to the data bus to meet the 165 ns setup time to the falling edge of the E clock. Finally, vector 6 sets  $\overline{IOW}$  high, resetting  $\overline{E}$  low, thereby latching the data. (In this example, all of the timing requirements have been met for the write cycle except for the 10 ns hold time on  $\overline{R/W}$  and  $\overline{CS}$ ; these cannot be met since the removal of  $\overline{IOW}$  is required to force the E clock to return low. However, the card and bus cycle function properly.)

### CRT Controller Write Cycle Vector File

```
GROUPS [40-21] [20-13] [1] [2] [3] [4] [5] [7] [8] [6] [12-9]
DISPLAY HEX, HEX, BIN

!   A19 -- AO           D7-DO           I I M M A R I I UNUSED
!                               O O E E E E O O
!                               R W M M N S R C
!                               - - W R   E D L
!                               - -   -   T Y K
!
! START OF WRITE CYCLE write $3D5, data $55
1 $003D5                $XX           1 1 1 1 0 0 X X XXXX
2 $003D5                $XX           1 0 1 1 0 0 X X XXXX
3 $003D5                $XX           1 0 1 1 0 0 X X XXXX
4 $003D5                $XX           1 0 1 1 0 0 X X XXXX
5 $003D5                $55           1 0 1 1 0 0 X X XXXX
6 $003D5                $55           1 1 1 1 0 0 X X XXXX
STOP
```

(Note that the grouping of vectors in the file with a few comments can be self-documenting. By using the appropriate mixture of hexadecimal and binary vector entry, a very readable file results. Also, the GROUPS statement in the vector file allows flexibility in mapping the module pins to the fixture while maintaining a readable vector file.)

The following CRT Controller Read Cycle Vector File contains the vectors necessary to perform the read cycle. The read cycle is almost identical to the write cycle except for a few points.  $\overline{IOR}$  is used in place of the  $\overline{IOW}$  line. The data lines are not driven since the CGA Card provides the bus data. After vector 5, the CAPTURE statement is entered to clock the input section. At the time the actual clocking occurs, the E clock will have been

high for two cycles (334 ns) to ensure that the data held on the bus is valid. Using the TL/1 *readword* command in the “stored” mode will return the data read.

### CRT Controller Read Cycle Vector File

```

GROUPS [40-21] [20-13] [1] [2] [3] [4] [5] [7] [8] [6] [12-9]
DISPLAY HEX, HEX, BIN

!   A19 - A0                D7-D0      I  I  M  M  A  R  I  I  UNUSED
!                               O  O  E  E  E  E  O  O
!                               R  W  M  M  N  S  R  C
!                               -  -  W  R      E  D  L
!                               -  -          T  Y  K
!
! START OF READ CYCLE read addr $3D5
1 $003D5          $XX          1  1  1  1  0  0  X  X  XXXX
2 $003D5          $XX          0  1  1  1  0  0  X  X  XXXX
3 $003D5          $XX          0  1  1  1  0  0  X  X  XXXX
4 $003D5          $XX          0  1  1  1  0  0  X  X  XXXX
5 $003D5          $XX          0  1  1  1  0  0  X  X  XXXX
  CAPTURE
6 $003D5          $XX          1  1  1  1  0  0  X  X  XXXX
STOP

```

## VIDEO RAM BUS CYCLES

## 7.8.

Another pair of bus cycles must be determined for accessing the video RAM. The access time of the RAM is much greater than that of the CRT Controller and, because the RAM is shared between the controller and bus accesses, the IORDY line is used with the WAIT line of the module to provide synchronization.

In the Video RAM Write Cycle Vector File that follows, vector 1 sets addresses A19 through A0 and data D7 through D0. Vector 2 holds the address and data while MEMW goes low. Upon receipt of MEMW and the correct address, IORDY goes low. The module then waits until the UUT acknowledges the write by returning the IORDY line high, which satisfies the WAIT condition, and vector driving continues. Vector 3 maintains the address and data for hold time.

### Video RAM Write Cycle Vector File

```

GROUPS [40-21] [20-13] [1] [2] [3] [4] [5J] [7] [8] [6] [12-9]
DISPLAY HEX, HEX, BIN

!   A19 - A0                D7-D0      I  I  M  M  A  R  I  I  UNUSED
!                               O  O  E  E  E  E  O  O
!                               R  W  M  M  N  S  R  C
!                               -  -  W  R      E  D  L
!                               -  -          T  Y  K
!
! WRITE CYCLE address $B8000, data $41
1 $B8000          $41          1  1  1  1  0  0  X  X  XXXX

```

```

2 $B8000          $41      1 1 0 1 0 0 X X XXXX
WAIT +
3 $B8000          $XX     1 1 1 1 0 0 X X XXXX
STOP

```

In the Video RAM Read Cycle Vector File that follows, vector 1 sets addresses A19 through A0 and data D7 through D0 to tri-state. Vector 2 holds the address and data while MEMR goes low. Upon receipt of MEMR and the correct address, IORDY goes low. The module then waits until the UUT acknowledges the read (by returning the IORDY line high), which satisfies the WAIT condition. Vector driving then continues. Vector 3 maintains the address and MEMR while the CAPTURE Clock clocks the data on the bus. Vector 4 returns MEMR high to finish up the cycle.

### Video RAM Read Cycle Vector File

```

GROUPS [40-21] [20-13] [1] [2] [3] [4] [5] [7] [8] [6] [12-9]
DISPLAY HEX,HEX,BIN

!   A19 - A0          D7-D0          I I M M A R I I UNUSED
!                                     O O E E E E O O
!                                     R W M M N S R C
!                                     - - W R   E D L
!                                     - -   -   T Y K
!
! READ CYCLE address $B8000
1 $B8000          $XX     1 1 1 1 0 0 X X XXXX
2 $B8000          $XX     1 1 1 0 0 0 X X XXXX
  WAIT +
3 $B8000          $XX     1 1 1 0 0 0 X X XXXX
  CAPTURE
4 $B8000          $XX     1 1 1 1 0 0 X X XXXX
  STOP

```

## Vector File Generation

## 7.9.

Once the basic read and write cycles have been determined, the vector files to control and test the CGA Card can be built. One vector file can contain the initialization routines for the CGA Card for the 40 Column, 80 Column, and Graphics Modes. This consists of a sequence of writes to the CRT Controller. The data required for each register is described in the 6845 Register Description in the Color/ Graphics Monitor Adapter Section in the IBM Technical Reference Manual.

Other vector files can be built to read and write to various registers for testing the CRT Controller, and to write various patterns into video RAM and to read back the values. In

addition to performing writes and reads to the CGA Card, the 9100 probe and clock module can take signatures of the RGBI outputs to test the outputs.

## CGA Write/Read Program Example

## 7.10.

The following program initializes the CGA Card in the 80 column character mode, fills the screen with ASCII characters, and then reads back two locations.

```

*****
program cga_prog

declare
  global string devname
end declare

! *****
! READ_LOC (ADDR) - This function calls the read_cycle function with the
! vector start number (which corresponds to the address to be read), and
! processes the returned byte read.
! *****
function read_loc (addr)

  declare
    numeric addr
  end declare

  ! *****get the vector start address*****
  ! $B8000 = 1, $B8001 = 4, $B8002 = 7, . . .
  vec_num = ((addr - $B8000) * 3) + 1

  ! *****go do the read*****
  byte_str = read_cycle vec_start vec_num

  ! *****check for stable levels on the read (no ""'s)*****
  if (instr (byte_str, "") = 0) then
    ! *****stable read*****
    byte_nun = val (byte_str,2)
    print "Read @ " + str (addr,16) + " = " + str (byte_nun,16)
  else
    ! *****unstable read*****
    print "Read @ " + str (addr,16) + " Is Not Stable"
  end if

  return errflg

end function

! *****
! READ_CYCLE (VEC_START) - This function drives the vector at vec_start
! returns the stored readword byte.
! *****
function read_cycle (vec_start)

  declare
    numeric vec_start
    global string devname
  end declare

  ! *****set up input section for read*****
  sync device devname, mode "capture"

  arm device devname
  vectordrive device devname, startmode "now"; vector vec_start
  loop until (drivepoll device devname) = 3
  end loop
  readout device devname

  byte_str = readword device devname, word 1, mode "stored"

  return byte_str

```

```

end function
! *****
! This is the main program body.
! *****
devname - "/mod1"

! *****Define Word To Match D7 - D0*****
setword device devname, word 1, as_pins "20 19 18 17 16 15 14 13"

! *****Reset PC CGA Card*****
writepin device devname, pin 7, level "1", mode "latch" ! Reset
writepin device devname, pin 7, level "0", mode "latch"

! *****Sync Vector Output With 6 MHz Bus Clock*****
syncoutput device devname, mode "drclk"

! *****Drive upon "vectordrive" command, use rising edge*****
edgeoutput device devname, start "at_vectordrive", stop "+", clock "+"

! *****initialize CGA card*****
vectorload device devname, file "init_color"
vectordrive device devname, startmode "now", vector 1
  loop until (drivepoll device devname) = 3 ! Wait for drive complete
end loop

! *****Put up ascii screen (part 1)*****
vectorload device devname, file "alpha_80_1"
vectordrive device devname, startmode "now", vector 1
  loop until (drivepoll device devname) = 3
end loop
! *****Put up ascii screen <part 2>*****
vectorload device devname, file "alpha_80_2"
vectordrive device devname, startmode "now", vector 1
  loop until (drivepoll device devname) = 3
end loop

vectorload device devname, file "read_vram"
read_loc addr $B8000 ! read character 1
read_loc addr $B8001 ! read attributes
read_loc addr $B8002 ! read character 2
read_loc addr $B8003 ! read attributes

end program
*****

```

## 68000 BUS VIDEO APPLICATION

7.11.

The video board of the 9100A is a 68000 bus-controlled UUT. It uses the Signetics SCN2674 Advanced Video Display Controller as the CRT Controller. The board has a write-only, character mode, memory-mapped video RAM to which the character code and attribute information are written.

## Testing The Video Board

7.12.

The test strategy for the video board is similar to that of the CGA Card. One significant difference is that the video RAM is write-only, so its performance is tested by using CRCs of the video output. Because the CRT controller is different than that of the CGA Card, a different fixture interface and bus cycles are required.

## Fixturing the Video Board Interface

## 7.13.

Table 7-2 maps the pin and signal names of the Video Board with the pins of the Vector Output I/O Module.

**Table 7-2. Video Board to Vector Output I/O Module Mapping**

| Pin | Signal | Module Pin | Pin | Signal | Module Pin |
|-----|--------|------------|-----|--------|------------|
| 27C | A11    | 40         | 7A  | D6     | 20         |
| 28C | A10    | 39         | 6A  | D5     | 19         |
| 29C | A9     | 38         | 5A  | D4     | 18         |
| 30C | A8     | 37         | 4A  | D3     | 17         |
| 24A | A7     | 36         | 3A  | D2     | 16         |
| 25A | A6     | 35         | 2A  | D1     | 15         |
| 26A | A5     | 34         | 1A  | D0     | 14         |
| 27A | A4     | 33         | 14A | R/W    | 13         |
| 28A | A3     | 32         | 12A | UDS    | 12         |
| 29A | A2     | 31         | 13A | LDS    | 11         |
| 30A | A1     | 30         | 10C | PS7    | 10         |
| 8C  | D15    | 29         | 13C | PS6    | 9          |
| 7C  | D14    | 28         | 12C | RESET  | 8          |
| 6C  | D13    | 27         | 22A | VWAIT  | 7          |
| 5C  | D12    | 26         | 16A | VDTACK | 6          |
| 4C  | D11    | 25         | 32A | +5V    |            |
| 3C  | D10    | 24         | 32C | +5V    |            |
| 2C  | D9     | 23         | 11C | +5V    |            |
| 1C  | D8     | 22         | 9A  | GND    |            |
| 8A  | D7     | 21         | 9C  | GND    |            |
|     |        |            | 31A | GND    |            |

The fixturing between the Vector Output I/O Module and the Video Board is minimal. To keep line lengths as short as possible and to keep interfacing easy, the Y9100A-102 Card Edge Interface Module is recommended. Most lines from the card edge interface module to the video 64-pin DIN connector go directly from the module to the connector except for the following:

- $\overline{\text{VDTACK}}$ ,  $\overline{\text{VWAIT}}$

To synchronize the bus cycles with the UUT,  $\overline{\text{DTACK}}$  must be applied to the WAIT input on the module. There are two separate signals from the video board;  $\overline{\text{VDTACK}}$  (a one wait-state  $\overline{\text{DTACK}}$  from the controller) and  $\overline{\text{VWAIT}}$  (a three wait-state  $\overline{\text{DTACK}}$  from the video RAM). Both of these lines are connected to the inputs of a 74LS08 AND

gate whose output is connected to WAIT. If either signal goes low, the WAIT input will go low as well. For troubleshooting in the case of no DTACK, both the VDTACK and VWAIT lines should also be connected to Vector Output I/O Module pins where their activity can be monitored.

- +5V, GND

An external power supply provides the required voltages to a DPDT switch that controls power to the video board.

## Determining Bus Cycles

### 7.14.

The SCN2674 Advanced Video Display Controller timing diagrams and the video board schematic (found in the 9100 Series Service Manual) can be used to determine the vector patterns necessary to perform bus cycles (see Figure 7-2). Signals PS7 and PS6 are formed by address strobe AS and memory decoding (PS7 = \$FXXXX, PS6 = \$EXXXX) on the 9100A main board. To access either the controller or video RAM requires the same basic cycle, the only difference being the controller data is D7 through D0 and is selected by PS7, while the video RAM data is D15 through D0 and is selected by PS6.

Because of the similarity of the cycles to access the controller and the video RAM, only the bus cycles to the controller are illustrated in this section.

The video input PS7 provides CE and also exits the board as VDTACK. The R/W signal is “ANDed” with LDS to get the RD and WR signals.

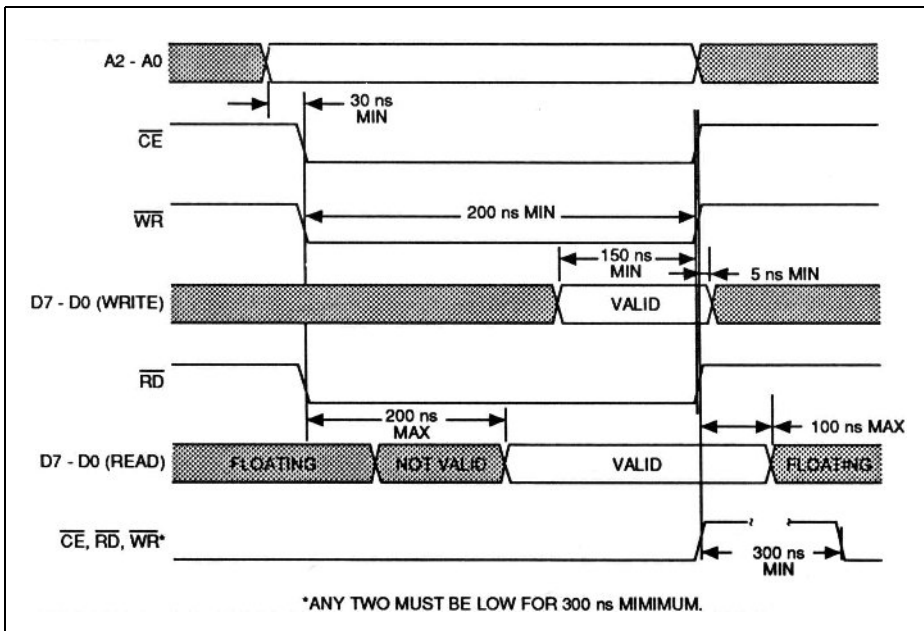
## Determining the Write Cycle

### 7.15.

Examine the SCN2674 4 MHz write cycle timing diagram to learn how its signals interface with the video board. To perform a write cycle, the following events must occur:

- Address bus A11 through A1 must be set a minimum of 30 ns prior to WR going low (R/W low and LDS low).





**Figure 7-2. SCN2674 Controller Timing Diagram (4 MHz Version)**

- $\overline{CE}$  (which is controlled by  $\overline{PS7}$ ) has a setup time of 0 ns prior to  $R/\overline{W}$  going low.
- $\overline{WR}$  has a minimum pulse width of 200 ns.
- Data D7 through D0 must be setup 150 ns minimum before  $\overline{WR}$  goes high.
- Data D7 through D0 must be held a minimum of 5 ns after  $\overline{WR}$  goes high.
- Address bus A11 through A1 and  $\overline{CE}$  have a 0 ns hold time after  $\overline{WR}$  goes high.
- $\overline{CE}$  must be high for 300 ns between chip accesses.

Using an internal clock frequency of 5 MHz (which gives 200 ns per vector resolution), the following 2674 Video Display Controller Write Cycle vector file can be made.

### 2674 Video Display Controller Write Cycle

```

GROUPS [40-30,1] [29-22] [21-14] [13] [12] [11] [10] [9] [8] [7] [6] [5-2]
DISPLAY HEX, HEX, HEX, BIN

!   ADDR          DATA          CTRL          UNUSED
! All - A1      D15-D8      D7-D0      R U L P P R V V
! (A0)          / D D S S E W D
!              W S S 7 6 S A T
!              - - - - - E I A
!              T T C
!              - - K
!              -
! WRITE CYCLE addr $f0013, data $XX00 (addr $fxxxx = PS7-, data $XX00 =
LDS-)
  1 $013          $XX          $XX          1 1 1 1 1 1 X X XXXX
  2 $013          $XX          $00          0 1 0 0 1 1 X X XXXX
    WAIT -
  3 $013          $XX          $00          1 1 1 1 1 1 X X XXXX
    STOP

```

Eleven address lines go to the video board for addressing the controller and video RAM. In a 68000-based system, A0 determines if the access is  $\overline{\text{LDS}}$  or  $\overline{\text{UDS}}$ . To provide a more readable vector file, the unused pin 1 in the previous example is assigned to be A0. (The  $\overline{\text{LDS}}/\overline{\text{UDS}}$  information is still contained under the control section of the vector file.)

The first vector in this file sets the address. Vector 2, which is driven 200 ns later, holds the address, sets the data, sets  $\overline{\text{R/W}}$  and  $\overline{\text{LDS}}$  low (to get  $\overline{\text{WR}}$  low), and sets  $\overline{\text{PS7}}$  low (to get  $\overline{\text{CE}}$  low). The WAIT statement ensures that  $\overline{\text{VDTACK}}$  is returned before continuing. The final vector, which is driven 200 ns later, holds the address and data while the CTRL signals are changed to return  $\overline{\text{WR}}$  and  $\overline{\text{CE}}$  high, thereby latching the data. Compare the write cycle described above with Figure 7-2 to confirm that the requirements for writing to the SCN 2674 are met.

## Determining the Read Cycle

### 7.16.

Examine the SCN2674 4 MHz read cycle timing diagram (Figure 7-2) to learn how its signals interface with the video board. To perform a read cycle, the following events must occur:

- Address bus  $\overline{\text{All}}$  through A1 must be set a minimum of 30 ns prior to  $\overline{\text{RD}}$  going low ( $\overline{\text{R/W}}$  high and  $\overline{\text{LDS}}$  low).

- $\overline{CE}$  (which is controlled by  $\overline{PS7}$ ) has a setup time of 0 ns prior to  $\overline{RD}$  going low.
- $\overline{RD}$  has a minimum pulse width of 200 ns.
- Data  $\overline{D7}$  through  $\overline{D0}$  will be valid a maximum of 100 ns after  $\overline{RD}$  goes low.
- Address bus  $\overline{A11}$  through  $\overline{A1}$  and  $\overline{CE}$  have a 0 ns hold time after  $\overline{RD}$  goes high.
- $\overline{CE}$  must be high for 300 ns between chip accesses.

The following 2674 Video Display Controller Read Cycle vector file performs a read cycle.

### 2674 Video Display Controller Read Cycle

```

GROUPS [40-30,1] [29-22] [21-14] [13] [12] [11] [10] [9] [8] [7] [6] [5-2]
DISPLAY HEX, HEX, HEX, BIN

!   ADDR                DATA                CTRL                UNUSED
!   A11 - A1            D15-D8      D7-D0                R  U  L  P  P  R  V  V
!                                     /  D  D  S  S  E  W  D
!                                     W  S  S  7  6  S  A  T
!                                     -  -  -  -  -  E  I  A
!                                     -  -  -  -  -  T  T  C
!                                     -  -  -  -  -  -  -  K
!                                     -  -  -  -  -  -  -  -
!
! READ CYCLE addr $f0011 (addr $fxxxx =PS7-, data = LDS-)
1 $011          $XX      $XX      1  1  1  1  1  1  X  X  XXXX
2 $011          $XX      $XX      1  1  0  0  1  1  X  X  XXXX
   WAIT -
3 $003          $XX      $XX      1  1  1  1  1  1  X  X  XXXX
   CAPTURE
   STOP

```

The first vector in this file sets the address. Vector 2, which is driven 200 ns later holds the address holds  $\overline{R/W}$  high and sets  $\overline{LDS}$  low (to get  $\overline{RD}$  low), and sets  $\overline{PS7}$  low (to get  $\overline{CE}$  low). The WAIT statement ensures that  $\overline{VDTACK}$  is returned before continuing. The final vector, which is driven 200 ns later, holds the address while the CTRL signals are changed to return  $\overline{RD}$  and  $\overline{CE}$  high. Approximately 100 ns after  $\overline{RD}$  and  $\overline{CE}$  have returned high (in the center of the vector period), the CAPTURE clock clocks the data in on the data bus. A *readword* command using the “stored” mode can then be performed to retrieve the data. Note that the read data is only valid for 100 ns maximum after the return of  $\overline{RD}$  and  $\overline{CE}$  to the high state. If the read

results are not stable, the *setoffset* command should be used to delay the data read with respect to the clock. An alternative method would be to add one more vector to hold  $\overline{RD}$  and  $\overline{CE}$  low while the capture is performed before returning  $\overline{RD}$  and  $\overline{CE}$  high.

Once the controller read and write cycles have been determined, a vector file to control and test the controller can be written by changing the address and data fields as appropriate. By changing  $\overline{PS7}$  and  $\overline{PS6}$  and adding  $\overline{UDS}$  to follow the  $\overline{LDS}$  pattern, the write cycle to access the video RAM can be made. Vector files can then be built that display various patterns that can be used to test the video board

## WAIT APPLICATION

## 7.17.

The edge specified for the WAIT command in a vector file cannot be changed within the vector file. For some handshaking protocol, it may be necessary to wait for an edge of one polarity, drive some vectors, and wait for the edge of the other polarity. This can be performed externally to the module by dedicating one of the output lines of the module to control an exclusive OR gate to be used as a programmable inverter. The vector file entries for the dedicated line determine if the event connected to the other input of the gate should be inverted (or not inverted) before going to the WAIT input of the module.

In some applications, WAIT may need to be checked for a level rather than an edge (for example, checking for clear-to-send (CTS) or BUSY signals). The external ENABLE line could be used to perform this function only if the application does not require programmable checks and does not have problems if vector driving is stopped during a particular series of vectors. If neither of these conditions can be met, there are several different methods that could be used to check for a level.

One method would dedicate a line from the Vector Output I/O Module for a gate on a test fixture that enables the controlling signal (CTS, etc.)(see Figure 7-3). The vector immediately preceding the WAIT statement in the vector file would contain a level that enables the controlling signal. When the vector was driven and the controlling signal was at the desired level, setting

the line on the vector would provide the edge necessary to satisfy the WAIT. If the controlling signal was not at the desired level, vector driving would halt until the signal was set to the correct level.

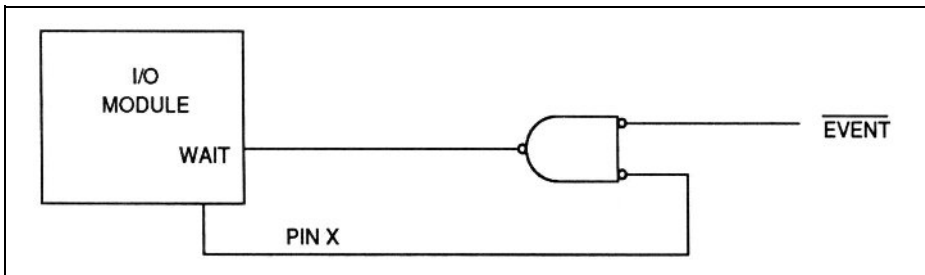


Figure 7-3. Using WAIT to Check for a Level (Method 1)

Another method would “AND” the controlling signal with the Vector Output I/O Module INT CLK line (See Figure 7-4). This method would provide an edge when the controlling signal was satisfied.

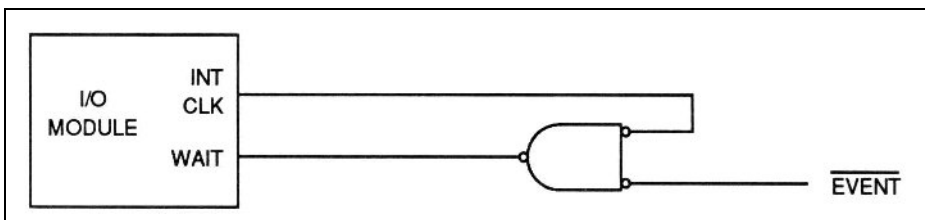


Figure 7-4. Using WAIT to Check for a Level (Method 2)

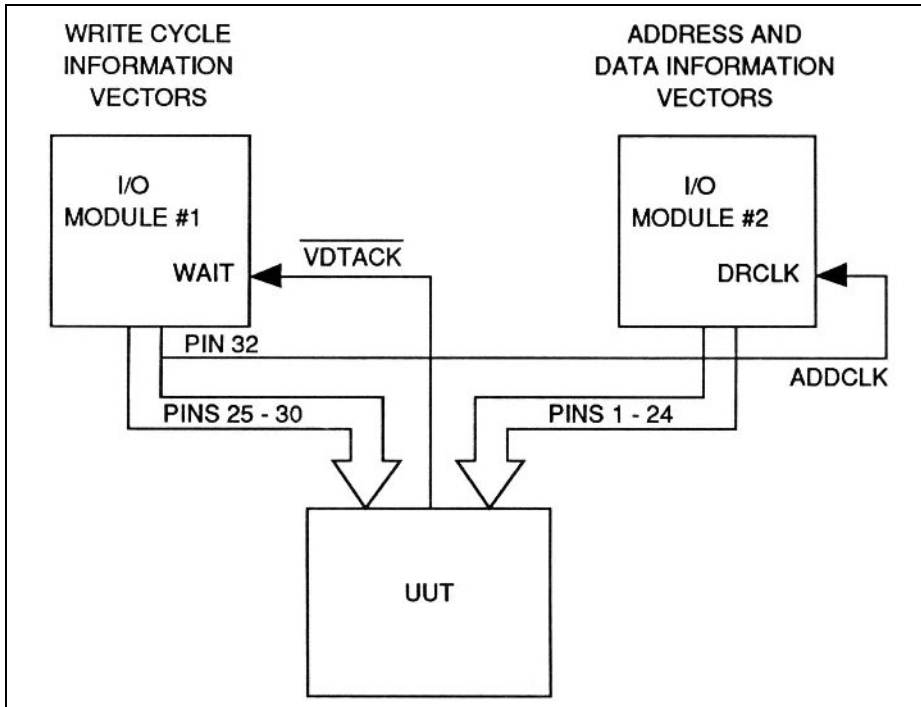
## TIMING SETS

## 7.18.

The Vector Output I/O Module does not require the use of timing sets for most applications because of its pattern depth of 8192 vectors and loading time of 3 seconds for an 8K vector file. However, to reduce programming time or the number of vector files necessary, two Vector Output I/O Modules can be used to provide timing sets.

One module would contain a vector file of the basic timing sets; that is, the control signal vectors required to perform the read-cycle, writecycle, or other cycles. The vector file would contain a LOOP statement enclosing the cycle information set to the

number of times the cycle is to be repeated. One output line of the first module would be connected to the second module's DR CLK line to clock out the address and data vectors (as shown in Figure 7-5).



**Figure 7-5. Connecting Two Modules for Timing Sets**

The second module would contain the vector patterns to be output on the address and data bus lines. The number of entries in this vector file should be the same as the LOOP value set in the first module's vector file.

The following example demonstrates this method:

### Module 1 Vector File (Timing Sets)

```

GROUPS [25] [26] [27] [28] [29] [30] [31] [32] [16-9] [8-1] [24-17] [33-40]
DISPLAY BIN, BIN, BIN, BIN, BIN, BIN, BIN, BIN, BIN, HEX, HEX, HEX, HEX

!      R U L P P R V A A15      A7      D7      UNUSED
!      / D D S S E D D -A8      -A0      -D0
!      W S S 7 6 S T D
!      - - - - - E A C
!      - - - - - T C L
!      - - - - - K K
!      -
!      1 1 1 1 1 1 1 X 0 $XX      $XX      $XX      $XX

      LOOP 5
! START OF WRITE CYCLE
2 1 1 1 1 1 1 X 1 $XX      $XX      $XX      $XX
3 0 0 0 1 0 1 X 0 $XX      $XX      $XX      $XX
      WAIT -
4 0 1 1 1 1 1 X 0 $XX      $XX      $XX      $XX
5 1 1 1 1 1 1 X 0 $XX      $XX      $XX      $XX
      ENDLLOOP
      STOP

```

### Module 2 Vector File (Address/Data Information)

```

GROUPS [16-9] [8-1] [24-17] [25-40]
DISPLAY HEX

!TIME DATA

!      A15-A8      A7-A0      D7-D0      UNUSED
1 $01      $00      $55      $XXXX
2 $01      $01      $67      $XXXX
3 $01      $02      $46      $XXXX
4 $01      $03      $00      $XXXX
5 $01      $04      $13      $XXXX
      STOP

```

In this example, module 1 is set to the appropriate *syncoutput* mode, module 2 is set to *syncoutput* mode “drclk”, and the *edgeoutput* clock argument is set to “+”. When module 1 begins driving out vectors, its pin 32 ADDCLK line clocks the second module to get the correct address and data. Each time the loop is driven, module 2 gets clocked to update its address and data.

Note that module 1 waits until the UUT returns  $\overline{\text{VDTACK}}$  before continuing.

## TL/1 VECTOR FILE GENERATION

## 7.19.

Generation of large vector files that use bus cycles can be easily automated using TL/1 programming. Once a bus cycle has been determined, it can be used as a pattern for large vector files that repeat the cycle with new address and data information. The bus cycle vector file can be copied into a text file to be used as a

template (such as the following Write Cycle Text File). A second text file can be made that contains the address and data to be inserted into the bus cycle (such as the following Address Data Text File). A program (such as the example “build” program), can be used to build a text file that is filled with the bus cycle information that contains the required address and data information. Once the file is complete, the text file may be copied to a vector file.

### Write Cycle Text File

```
GROUPS [40-21] [20-13] [1] [2] [3] [4] [5] [7] [8] [6] [12-9]
DISPLAY HEX,HEX,BIN

!   A19 - A0           D7-D0           I I M M A R I I  UNUSED
!                               O O E E E E O O
!                               R W M M N S R C
!                               - - W R E D L
!                               - - T Y K
! START OF WRITE CYCLE write $B8000, data $4107
  1 $B8000             $41             1 1 1 1 0 0 X X  XXXX
  2 $B8000             $41             1 1 0 1 0 0 X X  XXXX
    WAIT +
  3 $B8000             $XX             1 1 1 1 0 0 X X  XXXX
  4 $B8001             $07             1 1 1 1 0 0 X X  XXXX
  5 $B8001             $07             1 1 0 1 0 0 X X  XXXX
    WAIT +
  6 $B8001             $XX             1 1 1 1 0 0 X X  XXXX
```

Note that the previous vector file contains writes at two different addresses. The build program automatically handles the address incrementing.

The following text file contains the address followed by the attribute byte and character.

### Address Data Text File

```
B8000 07 41
B8002 07 42
B8004 07 43
B8006 07 44
B8008 07 45
BS00A 07 46
BS00C 07 47
B800E 07 48
B8010 07 49
B8012 07 4A
```

The following program was written specifically to process the write cycle given in this example. This program can be easily changed to handle different bus cycles.



```

program build
! *****
! The build program is used to generate textfiles that may be coerced into
! vector files. It is necessary to have a textfile "WRITECYCLE" which
! contains a description of a single write cycle. The user will be prompted
! to enter the text file name which contains the address and data information
! and the text file name to which the file is to be written to.
!
! Last updated: 4/21/89
! *****

declare
  global string array [1:17] writecyc
  global numeric in_text
  global numeric out_text
  global numeric t2o
  string old_file
  string new_file
end declare

! *****
! The build vector routine.
! *****
function build_vector

  declare
    string array [1:9] write_it
    global string array [1:17] writecyc
    global numeric in_text
    global numeric out_text
    string addr_val_str
    string patt_str
    string attr_str
    global numeric t2o
  end declare

  loop while (poll channel in_text, event "input") ! go until end of file

    ! *****get address, attribute, character*****
    input on in_text, using "#### ## #\n", addr_val_str, attr_str, patt_str
    print on t2o, "\1B[3;1f\1B[KWrite At $" + addr_val_str + " - Data $" +
      patt_str
    ! *****heading comment for each cycle*****
    write_it [1] = mid (writecyc [9], 1, 29) + addr_val_str + ", data $" +
      patt_str + attr_str

    ! *****pick apart old cycle, slip in new stuff*****
    preamble = mid (writecyc [10], 1, 7) + addr_val_str + "          $" +
      patt_str

    ! *****write cycle to character location*****
    write_it [2] = preamble + mid (writecyc [10], 32, 36)
    write_it [3] = preamble + mid (writecyc [11], 32, 36)
    write_it [4] = writecyc [12]
    preamble = mid (writecyc [13], 1, 7) + addr_val_str + "          $"
    write_it [5] = preamble + mid (writecyc [13], 32, 36)

    ! *****increment the address by one*****
    addr_val_str = str (((val (addr_val_str,16)) + 1), 16)
    print on t2o, "\1B[4;1f\1B[KWrite At $" + addr_val_str +
      " - Attribute $" + attr_str
    preamble = mid (writecyc [14], 1, 7) + addr_val_str +
      "          $" + attr_str

    ! *****write cycle to the attribute location*****
    write_it [6] = preamble + mid (writecyc [14], 32, 36)
    write_it [7] = preamble + mid (writecyc [15], 32, 36)
    write_it [8] = writecyc [16]
    preamble = mid (writecyc [17], 1, 7) + addr_val_str + "          $"
    write_it [9] = preamble + mid (writecyc [17], 32, 36)

    ! *****write whole cycle*****
    for vec_out = 1 to 9
      print on out_text, write_it [vec_out]
    next

  end loop

end build_vector

```

```

! *****
! This is the main program body.
! *****
! *****open devices, print screen****
t2u = open device "/term2", as "update"
t2o = open device "/term2",as "output"

print on t2o, "\1B[H\1B[J"

! *****get existing cycle information****
! *****this example has 17 lines in the writecycle file****
! *****which are stored in writecyc string array****
if (filestat file "writecycle") <> "" then

    cycle = open device "writecycle", as "input"

    print on t2o, "\1B[1;1fRetrieving Write cycle Information ..."

    for n = 1 to 17
        input on cycle, writecyc [n]
    next

    close (cycle)

! *****get name of file with address and data information****
print on t2o, "\1B[1;1f\1B[JInput Name Of Address/Data File"
input on t2u, old_file

if (filestat file old_file) <> "" then
    in_text = open device old_file, as "input"

! *****get file name of the file to store text file to****
print on t2o, "\1B[1;1f\1B[JInput Name Of Output File"
input on t2u, new_file out_text = "open device new_file, as "output"

! *****Print out header information. This includes the GROUPS line,
! *****DISPLAY line, and comments describing signal names of the groups.
for n = 1 to 8
    print on out_text, writecyc [n]
next

print on t2o, "\1B[1;1f\1B[J\1B[?251Building Vector Text File . . ."
build_vector()

close (in_text)
close (out_text)

print on t2o, "\1B[1;1f\1B[JVector Textfile Complete."
print on t2o, "\1B[2;1fCopy to Vector File."
else
    print on t2o, "\1B[1;1f\1B[J" + old_file + " Text File Does Not Exist."
end if

else
    print on t2o, "\1B[1;1f\1B[JWRITECYCLE Text File Does Not Exist."
end if

close (t2o)
close (t2u)

end program

```

# Appendix A

# New TL/1 Commands

---



# clockfreq

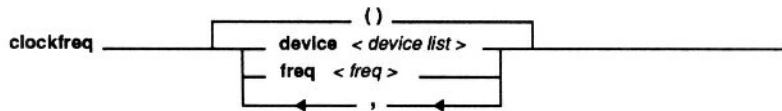
## Syntax:

```
clockfreq [device <device list>] [, freq  
          <freq>]
```

```
clockfreq (<device list>,<freq>)
```

```
clockfreq()
```

## Syntax Diagram:



## Description:

Sets the internal vector output drive clock frequency to 1 MHz, 5 MHz, 10 MHz, or 20 MHz. This command is used with the Vector Output I/O Module only.

## Arguments:

|             |                                                                             |
|-------------|-----------------------------------------------------------------------------|
| device list | I/O module name, clip name, or combination of these.<br>(Default = “/mod1”) |
| freq        | freq of “1MHz”, “5MHz”, “10MHz”, or “20MHz”.<br>(Default = “1MHz”)          |

### Example 1:

```
! Set clock frequency of /mod1 to 1MHz
! Set clock frequency of /mod2 to 5MHz
! Set clock frequency of /mod3 to 10MHz
! Set clock frequency of /mod4 to 20MHz
clockfreq device "/mod1", freq "1MHz"
clockfreq device "/mod2", freq "5MHz"
clockfreq device "/mod3", freq "10MHz"
clockfreq device "/mod4", freq "20MHz"
```

### Example 2:

```
! Use default values to set clock freq. of
! /mod1 to 1MHz
clockfreq()
```

### Example 3:

```
! Set clock frequency of "/mod1" to "10MHz"
clockfreq ("/mod1", "10MHz")
```

### Remarks:

The *clockfreq* command sets the internal clock frequency that is used to drive out vectors. The frequency changes immediately upon command execution. Note that there is no synchronization between the internal clocks of each I/O module. Changing the internal clock frequency during vector driving is not recommended. This frequency is also available on the INT CLK output pin of the module as a free-running signal and, unlike the vector output clock, is not affected by external START, STOP, or ENABLE conditions.

### Related Commands:

*syncoutput, vectordrive*

### **For More Information:**

- The “Overview Of TL/1” section of the *Programmer’s Manual*.





# drivepoll

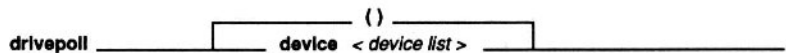
## Syntax:

```
drivepoll device <device list>
```

```
drivepoll (<device list>)
```

```
drivepoll()
```

## Syntax Diagram:



## Description:

Returns the current drive and handshake status of the requested I/O module(s). This command is used with the Vector Output I/O Module only.

## Arguments:

|             |                                                                                 |
|-------------|---------------------------------------------------------------------------------|
| device list | I/O module name, clip module name, or combination of these. (Default = “/mod1”) |
|-------------|---------------------------------------------------------------------------------|

## Returns:

Vector drive status is indicated by bit 0. Bit 0 is 0 if vector driving is not complete and is 1 if vector driving is complete.

Handshake status is indicated by bit 1. Bit 1 is 0 if vector driving is suspended and the selected edge of the signal on the WAIT input has not yet been received. Bit 1 is 1 if the module is not awaiting a WAIT input.

### Example 1:

```
!Wait for drive and handshake to complete on
!/mod1
vectordrive device "/mod1"
loop while ((drivepoll device "/mod1") <> 3)
end loop
print "vector drive completed"
```

### Example 2:

```
! Use "/mod2" to supply the rising edge to
! "/mod1" each time a handshake is needed.
writepin device "/mod2", pin 1, level "0"
vectorload device "/mod1", file "vec1"
edgeoutput device "/mod1", start
  "at_vectordrive"
vectordrive device "/mod1"
n = 0
loop while (n <> 3)
  n = drivepoll device "/mod1"
  if n = 0 then
    ! give edge to mod1
    writepin device "/mod2", pin 1,
      level "1"
    writepin device "/mod2", pin 1,
      level "0"
  end if
end loop
```

### Remarks:

If the *sysncoutput* mode is set to “drclk” or “intfreq”, and a signal on the external STOP line is received with the proper edge, vectordriving will be terminated, but *drivepoll* will not show this termination since all vectors loaded have not been driven.

### Related Commands:

*vectordrive*

**For More Information:**

- The “Overview Of TL/1” section of the *Programmer’s Manual*.



# edgeoutput

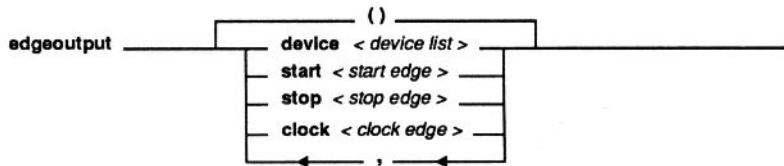
## Syntax:

```
edgeoutput [device <device list>
           [,start<start edge>] [, stop <stop
           edge>] [, clock <clock edge>]

edgeoutput (<device list>,<start edge>, <stop
           edge>, <clock edge>)

edgeoutput ()
```

## Syntax Diagram:



## Description:

Specifies the active edge for the external START, STOP, and DR CLK inputs used to drive out vectors. Does not affect the input section hardware. This command is used with the Vector Output I/O Module only.

## Arguments:

|             |                                                                             |
|-------------|-----------------------------------------------------------------------------|
| device list | I/O module name, clip name, or combination of these.<br>(Default = “/mod1”) |
| start edge  | External start edge “+”, “-”, or “at_vectordrive”.<br>(Default = “+”)       |
| stop edge   | External stop edge “+” or “-”.<br>(Default = “+”)                           |

clock edge

External clock edge “+” or “-”.  
(Default = “+”)

### Example 1:

```
mod = "/mod1"  
edgeoutput device mod, start "+", stop "-"  
,  
clock "+"
```

### Example 2:

```
modlist = "/mod1,/mod2"  
edgeoutput device modlist, start "-", stop  
"-", clock "+"
```

### Remarks:

The start edge argument “at\_vectordrive” allows vector driving to begin upon the receipt of a qualified clock. This occurs regardless of the start mode argument set in the *vectordrive* command. The start edge and the stop edge arguments have no effect on *syncoutput* modes “pod” and “int”. The clock edge argument has no effect on *syncoutput* modes “pod”, “int”, and “intfreq”.

### Related Commands:

*syncoutput*, *vectordrive*

### For More Information:

- The “Overview Of TL/1” section of the *Programmer’s Manual*.

# enableoutput

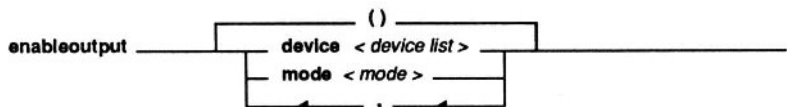
## Syntax:

```
enableoutput [device <device list>] [, mode  
             <mode>]
```

```
enableoutput (<device list>, <mode>)
```

```
enableoutput ()
```

## Syntax Diagram:



## Description:

Specifies the level for the external ENABLE line used to qualify the clocks for vector driving. Does not affect the input section hardware. This command is used with the Vector Output I/O Module only.

## Arguments:

|             |                                                                                    |
|-------------|------------------------------------------------------------------------------------|
| device list | I/O module name, clip module name, or combination of these.<br>(Default = "/mod1") |
| mode        | Enable modes: "always", "high", or "low".<br>(Default = "always")                  |

## Example 1:

```
mod = "/mod1"  
enableoutput device mod, mode "high"
```

## Example 2:

```
enableoutput device "/mod4B", mode "always"
```

## Remarks:

The *enableoutput* mode argument determines the clock qualifier used for vector driving. If the “always” mode is selected, all clocks are qualified to drive out vectors. If the “high” or “low” mode is selected, the appropriate level must be present on the external ENABLE line to qualify the clock. This command only qualifies the clock when the *syncoutput* mode is “drclk” or “intfreq”. Both *syncoutput* modes “int” and “pod” clocks are always qualified.

## Related Commands:

*edgeoutput, syncoutput, vectordrive*

## For More Information:

- The “Overview Of TL/1” section of the *Programmer’s Manual*.

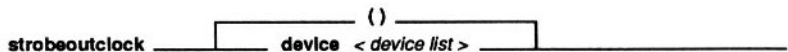


# strobeoutclock

## Syntax:

```
strobeoutclock device <device list>
strobeoutclock (<device list>)
strobeoutclock()
```

## Syntax Diagram:



## Description:

The *strobeoutclock* command performs an internal clocking of a vector when the module syncoutput mode is set to “int”. Subsequent executions of the *strobeoutclock* command continues to drive vectors until the end of the vector file is reached. This command is used with the Vector Output I/O Module only.

## Arguments:

|             |                                                                                        |
|-------------|----------------------------------------------------------------------------------------|
| device list | I/O module name, clip module name, or any combination of these.<br>(Default = “/mod1”) |
|-------------|----------------------------------------------------------------------------------------|

## Example 1:

```
mod = "/mod1"
syncoutput device mod, mode "int"
    ! syncoutput must be set to "int"
    ! or error occurs on strobeoutclock
edgeoutput device mod, start "at_vectordrive"
vectorload device mod, file "vecl"
vectordrive device mod
arm device mod
    strobeoutclock device mod
```

```

        strobeoutclock device mod
readout device mod
crc = sig device mod, pin 1
lvl = level device mod, pin 1, type "clocked"

```

## Example 2:

```

! use "/mod1" to set up peripheral port at
! $90000
! use pod to read results
mod = "/mod1"
edgeoutput device mod, start "at_vectordrive"
syncoutput device mod, mode "int"
    ! syncoutput must be set to "int"
    ! or error occurs on strobeoutclock
vectorload device mod, file "vec1"
! Clock out vector 1 = $55
vectordrive device mod
strobeoutclock device mod
if ((read addr $90000) <> $55) then print
    "Port 2 Failed $55"
! Clock out vector 2 = $AA
strobeoutclock device mod
if ((read addr $90000) <> $AA) then print
    "Port 2 Failed $AA"

```

## Remarks:

The *syncoutput* mode must be “int” or an error is issued upon *strobeoutclock* execution. The *strobeoutclock* command is useful for single-step debugging. It is also useful for controlling vector driving when other TL/1 commands must be used between vectors (such as pod actions) to perform initialization or control routines.

## Related Commands:

*syncoutput*, *vectordrive*

## For More Information:

- The “Overview Of TL/1” section of the *Programmer’s Manual*.

# syncoutput

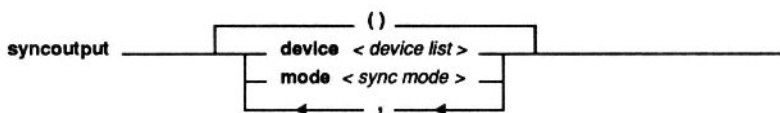
## Syntax:

```
syncoutput [device <device list>] [, mode  
          <sync mode>]
```

```
syncoutput (<device list>,<sync mode>)
```

```
syncoutput ()
```

## Syntax Diagram:



## Description:

Selects the vector output clock source. This command does not affect the input section hardware. The actual clock selection does not occur until the *vectordrive* command has been executed. This command is used with the Vector Output I/O Module only.

## Arguments:

|             |                                                                                        |
|-------------|----------------------------------------------------------------------------------------|
| device list | I/O module name, clip module name, or any combination of these.<br>(Default = "/mod1") |
| sync mode   | Sync Modes: "pod", "drclk", "int" or "intfreq".<br>(Default = "intfreq")               |

## Example 1:

```
syncoutput device "/mod1", mode "intfreq"  
clockfreq device "/mod1", freq "10MHz"  
vectordrive device "/mod1"
```

## Example 2:

```
syncoutput device "/mod2", mode "pod"  
syncoutput device "/mod1", mode "drclk"  
vectordrive device "/mod1, /mod2"
```

## Remarks:

When the *syncoutput* mode is set to “pod”, use the *sync* command to select either pod address, pod data, or another pod sync mode as the clock source.

When the *syncoutput* mode is set to “intfreq”, use the *clockfreq* command to control the frequency of the clock.

When the *syncoutput* mode is set to “int”, use the *strobeoutclock* command to clock out the vectors.

## Related Commands:

*clockfreq*, *edgeoutput*, *enableoutput*, *strobeoutclock*,  
*sync*, *vectordrive*

## For More Information:

- The “Overview Of TL/1” section of the *Programmer’s Manual*.

# vectordrive

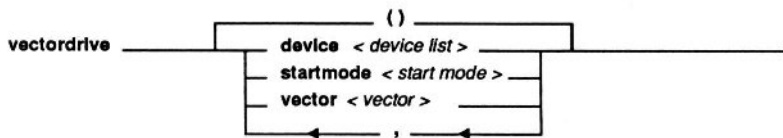
## Syntax:

```
vectordrive [device <device list> [,  
            startmode <startmode>] [, vector  
            <vector>]
```

```
vectordrive (<device  
            list>, <startmode>, <vector>)
```

```
vectordrive()
```

## Syntax Diagram:



## Description:

The execution of this command enables the loaded vector file to be driven, starting at the indicated vector. This command is analogous to an “arm” command for the output. This command is used with the Vector Output I/O Module only.

## Arguments:

|             |                                                                                |
|-------------|--------------------------------------------------------------------------------|
| device list | I/O module name or any combination of I/O module names.<br>(Default = “/mod1”) |
| startmode   | Start Mode: “at_arm” or “now”<br>(Default = “now”)                             |

vector

Vector number from which to begin driving (from 1 to the number of vectors loaded). The vector number directly corresponds to the vector number in the vector file. (Default = 1)

### Example 1:

```
vectorload device "/mod1", file "vec1"  
vectordrive device "/mod1", startmode "now",  
vector 1
```

### Example 2:

```
vectorload device "/mod1", file  
"/hdr/demo/vec1"  
vectordrive device "/mod1", startmode  
"at_arm", vector 3
```

### Remarks:

If *startmode* is “now”:

and *syncoutput* mode is “pod”, vector driving occurs on the receipt of Pod Sync clocks.

and *syncoutput* mode is “int”, vector driving occurs on the execution of the *strobeoutput* command.

and *syncoutput* mode is either “drclk” or “intfreq”:

and *edgeoutput* start is “at\_vectordrive”, vector driving occurs upon the receipt of a qualified clock.

and *edgeoutput* start is “+” or “-”, vector driving occurs upon on first qualified clock following the receipt of the external START.

If *startmode* is “at\_arm”:

the same conditions apply as above, except that the TL/1 *arm* command must be executed first (to permit simultaneous arming of the vector driving and the response gathering hardware).

Once loaded, a vector file can be driven repeatedly, starting at any specified vector, until overwritten by a *writopin* or *writeword* command.

### **Related Commands:**

*edgeoutput*, *enableoutput*, *syncoutput*, *strobeoutclock*,  
*vectorload*

### **For More Information:**

- The “Overview Of TL/1” section of the *Programmer’s Manual*.





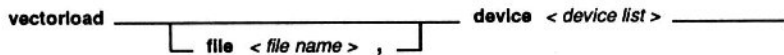
# vectorload

## Syntax:

```
vectorload [device <device list> ,] file  
          <file name>
```

```
vectorload(<device list>, <file name>)
```

## Syntax Diagram:



## Description:

Causes the vector file “filename” to be loaded into all modules designated in <device list>. A single vector file cannot cover more than one I/O Module. However, *vectorload* may load the same file into several modules at the same time. This command is used with the Vector Output I/O Module only.

## Arguments:

|             |                                                                                |
|-------------|--------------------------------------------------------------------------------|
| device list | I/O module name or any combination of I/O module names.<br>(Default = “/mod1”) |
| file name   | Absolute or relative file name of the vector file to be loaded.                |

## Example 1:

```
vectorload device "/mod1", file  
          "/hdr/tmp/vec1"  
vectordrive device "/mod1"
```

## Example 2:

```
vectorload device "/mod1", file "vec1"  
vectordrive device "/mod1"
```

## Remarks:

Whenever a *vectorload* command is executed, all vectors currently in the designated I/O Module are removed and the new vector file is loaded. The file remains in the designated I/O Module until overwritten by execution of the *writeword* or *writepin* commands, or until another file is loaded.

*Vectorload* should not be executed during vector driving or inconsistent results may occur. Loading a vector file has no effect on the current latched output. Because of this, vector files can be loaded between an “arm” and a “readout” with no adverse effects as long as no qualified clocks are received. This method works well when the input section hardware’s sync mode is “pod”, “int”, or “capture”. More care must be taken if the sync mode is “ext”.

When executed, the *vectorload* command uses available memory to transfer the vector file from disk to the I/O Module. If the entire file fits into memory, only one disk access is required and only about 3 seconds are required to load an 8K vector file from hard disk, or 15 seconds from floppy disk, to an I/O Module. If not enough RAM is available, several passes may be required to load the entire file, requiring more time. To reduce this time, the user may consider purchasing additional RAM to increase the available memory.

## Related Commands:

*vectordrive*

**For More Information:**

- The “Overview Of TL/1” section of the *Programmer’s Manual*.



# Appendix B

## I/O Module

### Clip/Pin Mapping

---

Clip size = 14, module installed on "A" side

| Clip | I/O | Mod | Pin | Clip | I/O | Mod | Pin |
|------|-----|-----|-----|------|-----|-----|-----|
| 1    | =   |     | 1   | 14   | =   |     | 20  |
| 2    | =   |     | 2   | 13   | =   |     | 19  |
| 3    | =   |     | 3   | 12   | =   |     | 18  |
| 4    | =   |     | 4   | 11   | =   |     | 17  |
| 5    | =   |     | 5   | 10   | =   |     | 16  |
| 6    | =   |     | 6   | 9    | =   |     | 15  |
| 7    | =   |     | 7   | 8    | =   |     | 14  |

Clip size = 14, module installed on "B" side

| Clip | I/O | Mod | Pin | Clip | I/O | Mod | Pin |
|------|-----|-----|-----|------|-----|-----|-----|
| 1    | =   |     | 21  | 14   | =   |     | 40  |
| 2    | =   |     | 22  | 13   | =   |     | 39  |
| 3    | =   |     | 23  | 12   | =   |     | 38  |
| 4    | =   |     | 24  | 11   | =   |     | 37  |
| 5    | =   |     | 25  | 10   | =   |     | 36  |
| 6    | =   |     | 26  | 9    | =   |     | 35  |
| 7    | =   |     | 27  | 8    | =   |     | 34  |

Clip size = 16, module installed on "A" side

| Clip | I/O | Mod | Pin | Clip | I/O | Mod | Pin |
|------|-----|-----|-----|------|-----|-----|-----|
| 1    | =   |     | 1   | 16   | =   |     | 20  |
| 2    | =   |     | 2   | 15   | =   |     | 19  |
| 3    | =   |     | 3   | 14   | =   |     | 18  |
| 4    | =   |     | 4   | 13   | =   |     | 17  |
| 5    | =   |     | 5   | 12   | =   |     | 16  |
| 6    | =   |     | 6   | 11   | =   |     | 15  |
| 7    | =   |     | 7   | 10   | =   |     | 14  |
| 8    | =   |     | 8   | 9    | =   |     | 13  |

Clip size = 16, module installed on "B" side

| Clip | I/O | Mod | Pin | Clip | I/O | Mod | Pin |
|------|-----|-----|-----|------|-----|-----|-----|
| 1    | =   |     | 21  | 16   | =   |     | 40  |
| 2    | =   |     | 22  | 15   | =   |     | 39  |
| 3    | =   |     | 23  | 14   | =   |     | 38  |
| 4    | =   |     | 24  | 13   | =   |     | 37  |
| 5    | =   |     | 25  | 12   | =   |     | 36  |
| 6    | =   |     | 26  | 11   | =   |     | 35  |
| 7    | =   |     | 27  | 10   | =   |     | 34  |
| 8    | =   |     | 28  | 9    | =   |     | 33  |

Clip size = 18, module installed on "A" side

| Clip | I/O | Mod | Pin | Clip | I/O | Mod | Pin |
|------|-----|-----|-----|------|-----|-----|-----|
| 1    | =   |     | 1   | 18   | =   |     | 20  |
| 2    | =   |     | 2   | 17   | =   |     | 19  |
| 3    | =   |     | 3   | 16   | =   |     | 18  |
| 4    | =   |     | 4   | 15   | =   |     | 17  |
| 5    | =   |     | 5   | 14   | =   |     | 16  |
| 6    | =   |     | 6   | 13   | =   |     | 15  |
| 7    | =   |     | 7   | 12   | =   |     | 14  |
| 8    | =   |     | 8   | 11   | =   |     | 13  |
| 9    | =   |     | 9   | 10   | =   |     | 12  |

Clip size = 18, module installed on "B" side

| Clip | I/O | Mod | Pin | Clip | I/O | Mod | Pin |
|------|-----|-----|-----|------|-----|-----|-----|
| 1    | =   |     | 21  | 18   | =   |     | 40  |
| 2    | =   |     | 22  | 17   | =   |     | 39  |
| 3    | =   |     | 23  | 16   | =   |     | 38  |
| 4    | =   |     | 24  | 15   | =   |     | 37  |
| 5    | =   |     | 25  | 14   | =   |     | 36  |
| 6    | =   |     | 26  | 13   | =   |     | 35  |
| 7    | =   |     | 27  | 12   | =   |     | 34  |
| 8    | =   |     | 28  | 11   | =   |     | 33  |
| 9    | =   |     | 29  | 10   | =   |     | 32  |

Clip size = 20, module installed on "A" side

| Clip | I/O | Mod | Pin | Clip | I/O | Mod | Pin |
|------|-----|-----|-----|------|-----|-----|-----|
| 1    | =   |     | 1   | 20   | =   |     | 20  |
| 2    | =   |     | 2   | 19   | =   |     | 19  |
| 3    | =   |     | 3   | 18   | =   |     | 18  |
| 4    | =   |     | 4   | 17   | =   |     | 17  |
| 5    | =   |     | 5   | 16   | =   |     | 16  |
| 6    | =   |     | 6   | 15   | =   |     | 15  |
| 7    | =   |     | 7   | 14   | =   |     | 14  |
| 8    | =   |     | 8   | 13   | =   |     | 13  |
| 9    | =   |     | 9   | 12   | =   |     | 12  |
| 10   | =   |     | 10  | 11   | =   |     | 11  |

Clip size = 20, module installed on "B" side

| Clip | I/O | Mod | Pin | Clip | I/O | Mod | Pin |
|------|-----|-----|-----|------|-----|-----|-----|
| 1    | =   |     | 21  | 20   | =   |     | 40  |
| 2    | =   |     | 22  | 19   | =   |     | 39  |
| 3    | =   |     | 23  | 18   | =   |     | 38  |
| 4    | =   |     | 24  | 17   | =   |     | 37  |
| 5    | =   |     | 25  | 16   | =   |     | 36  |
| 6    | =   |     | 26  | 15   | =   |     | 35  |
| 7    | =   |     | 27  | 14   | =   |     | 34  |
| 8    | =   |     | 28  | 13   | =   |     | 33  |
| 9    | =   |     | 29  | 12   | =   |     | 32  |
| 10   | =   |     | 30  | 11   | =   |     | 31  |



Clip size = 24, module installed on "A" side

| Clip | I/O | Mod | Pin | Clip | I/O | Mod | Pin |
|------|-----|-----|-----|------|-----|-----|-----|
| 1    | =   |     | 1   | 24   | =   |     | 20  |
| 2    | =   |     | 2   | 23   | =   |     | 19  |
| 3    | =   |     | 3   | 22   | =   |     | 18  |
| 4    | =   |     | 4   | 21   | =   |     | 17  |
| 5    | =   |     | 5   | 20   | =   |     | 16  |
| 6    | =   |     | 6   | 19   | =   |     | 15  |
| 7    | =   |     | 7   | 18   | =   |     | 14  |
| 8    | =   |     | 8   | 17   | =   |     | 13  |
| 9    | =   |     | 9   | 16   | =   |     | 12  |
| 10   | =   |     | 10  | 15   | =   |     | 11  |
| 11   | =   |     | 29  | 14   | =   |     | 32  |
| 12   | =   |     | 30  | 13   | =   |     | 31  |

Clip size = 24, module installed on "B" side

| Clip | I/O | Mod | Pin | Clip | I/O | Mod | Pin |
|------|-----|-----|-----|------|-----|-----|-----|
| 1    | =   |     | 21  | 24   | =   |     | 40  |
| 2    | =   |     | 22  | 23   | =   |     | 39  |
| 3    | =   |     | 23  | 22   | =   |     | 38  |
| 4    | =   |     | 24  | 21   | =   |     | 37  |
| 5    | =   |     | 25  | 20   | =   |     | 36  |
| 6    | =   |     | 26  | 19   | =   |     | 35  |
| 7    | =   |     | 27  | 18   | =   |     | 34  |
| 8    | =   |     | 28  | 17   | =   |     | 33  |
| 9    | =   |     | 29  | 16   | =   |     | 32  |
| 10   | =   |     | 30  | 15   | =   |     | 31  |
| 11   | =   |     | 9   | 14   | =   |     | 12  |
| 12   | =   |     | 10  | 13   | =   |     | 11  |

Clip size = 28

| Clip | I/O | Mod | Pin | Clip | I/O | Mod | Pin |
|------|-----|-----|-----|------|-----|-----|-----|
| 1    | =   |     | 1   | 28   | =   |     | 40  |
| 2    | =   |     | 2   | 27   | =   |     | 39  |
| 3    | =   |     | 3   | 26   | =   |     | 38  |
| 4    | =   |     | 4   | 25   | =   |     | 37  |
| 5    | =   |     | 5   | 24   | =   |     | 36  |
| 6    | =   |     | 6   | 23   | =   |     | 35  |
| 7    | =   |     | 7   | 22   | =   |     | 34  |
| 8    | =   |     | 8   | 21   | =   |     | 33  |
| 9    | =   |     | 9   | 20   | =   |     | 32  |
| 10   | =   |     | 10  | 19   | =   |     | 31  |
| 11   | =   |     | 11  | 18   | =   |     | 30  |
| 12   | =   |     | 12  | 17   | =   |     | 29  |
| 13   | =   |     | 13  | 16   | =   |     | 28  |
| 14   | =   |     | 14  | 15   | =   |     | 27  |

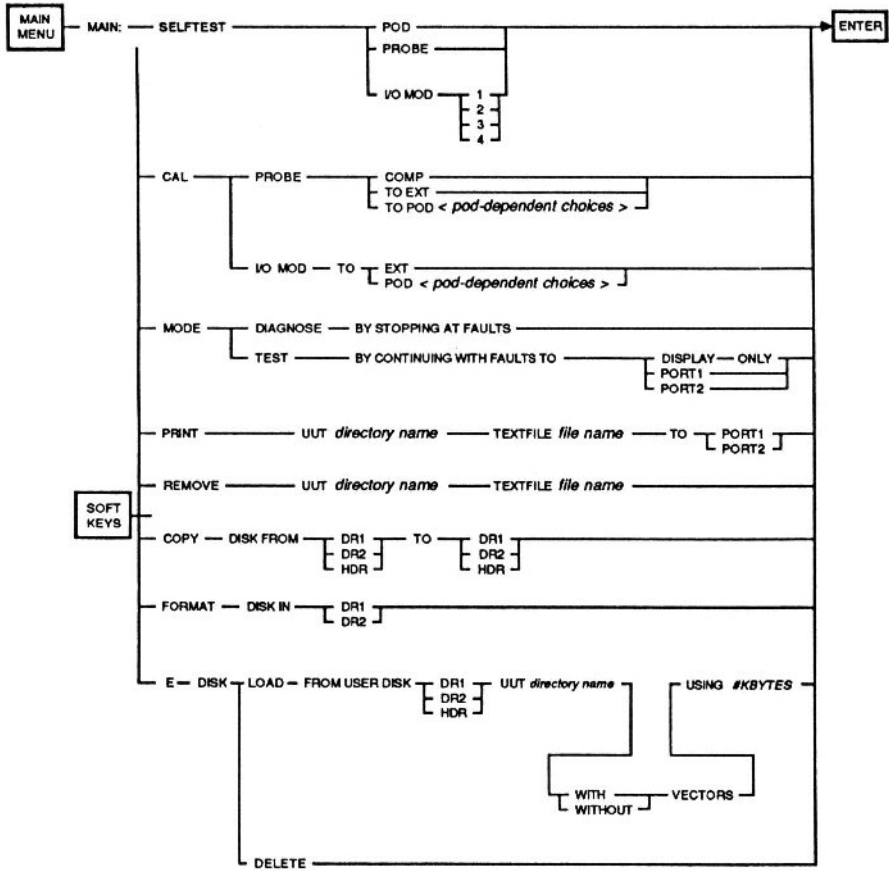
Clip size = 40

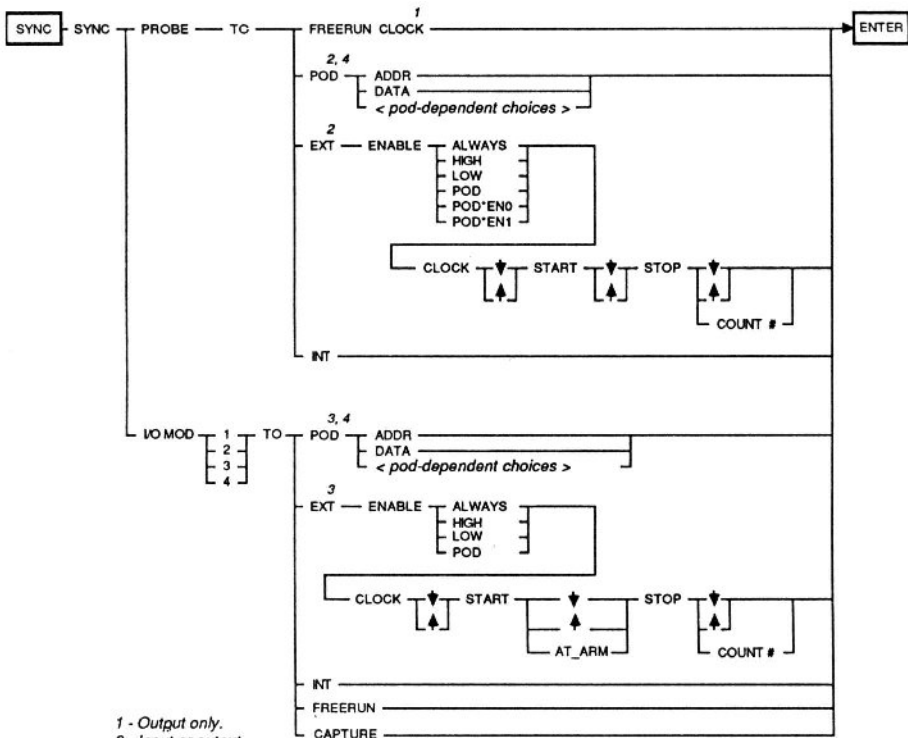
| Clip | I/O | Mod | Pin | Clip | I/O | Mod | Pin |
|------|-----|-----|-----|------|-----|-----|-----|
| 1    | =   |     | 1   | 40   | =   |     | 40  |
| 2    | =   |     | 2   | 39   | =   |     | 39  |
| 3    | =   |     | 3   | 38   | =   |     | 38  |
| 4    | =   |     | 4   | 37   | =   |     | 37  |
| 5    | =   |     | 5   | 36   | =   |     | 36  |
| 6    | =   |     | 6   | 35   | =   |     | 35  |
| 7    | =   |     | 7   | 34   | =   |     | 34  |
| 8    | =   |     | 8   | 33   | =   |     | 33  |
| 9    | =   |     | 9   | 32   | =   |     | 32  |
| 10   | =   |     | 10  | 31   | =   |     | 31  |
| 11   | =   |     | 11  | 30   | =   |     | 30  |
| 12   | =   |     | 12  | 29   | =   |     | 29  |
| 13   | =   |     | 13  | 28   | =   |     | 28  |
| 14   | =   |     | 14  | 27   | =   |     | 27  |
| 15   | =   |     | 15  | 26   | =   |     | 26  |
| 16   | =   |     | 16  | 25   | =   |     | 25  |
| 17   | =   |     | 17  | 24   | =   |     | 24  |
| 18   | =   |     | 18  | 23   | =   |     | 23  |
| 19   | =   |     | 19  | 22   | =   |     | 22  |
| 20   | =   |     | 20  | 21   | =   |     | 21  |

# Appendix C Keypad Reference Changes

---

This appendix lists the changes made to the keypad reference syntax to accommodate the 9100A-017 Vector Output I/O Module. Only the MAIN MENU key and the SYNC key are changed.





- 1 - Output only.
- 2 - Input or output.
- 3 - Input only.
- 4 - Setting the probe synchronization sets the corresponding I/O module synchronization, and vice versa.



# Index

---

68000 video applications, 7-12

arm, 4-14, 5-3

Asynchronous level history register, 4-14

Block diagram overview, 4-3

Block reads, 7-3

Bus emulation, 7-2

Cable length, 4-9

Calibrating the module, 4-6

Calibration module, 4-8

Capture, 4-7, 6-12

Capture mode, 4-15

Card edge fixture kit, 4-10

Card edge interface module, 4-10

CGA Mapping, 7-5

CGA Program Example, 7-11

CGA Video Card, 7-4

Changing the WAIT edge, 7-18

Clip modules, 4-1, 4-9

CLOCK, 4-1

Clock selection, 3-1

Clocked level history register, 4-12

clockfreq, 3-1, 5-1, Appendix A

Comment lines, 6-6

COMMON, 4-1, 4-3

compare, 4-3

Completion status, 3-5

Connecting the module to the mainframe, 4-5

Connecting the module to the UUT, 4-9

Control phrases, 6-11

CRC signature register, 4-12

Creating a new vector file, 6-1

CTS, 7-18  
Cyclic redundancy check, 7-3

Data Compare Equal, 7-3  
Data fields, 6-6  
DCE, 4-3  
Determining bus cycles, 7-14  
Determining the read cycle, 7-16  
Determining the write cycle, 7-14  
Display line, 6-5  
DR CLK, 4-2, 4-16  
drivepoll, 5-1, Appendix A

edgeoutput, 5-2, Appendix A  
ENABLE, 4-1  
enableoutput, 5-3, Appendix A  
Endloop, 6-11  
External, I/O Module to, 4-7  
External lines, 4-10  
External sync mode, 4-11

Fixture, 4-10, 7-5, 7-13  
Flying lead set, 4-10  
Freerun sync mode, 4-14  
Fuse, 4-3

Groups line, 6-4

Handshake input, 4-2  
Handshake status, Appendix A  
Handshaking, 3-2

I/O Module to external, 4-7  
I/O Module clip/pin mapping, Appendix B  
I/O Module to pod, 4-8  
INT CLK, 4-3, 4-16, 7-3  
Internal clock delay, 4-7  
Internal sync mode, 4-14

Keypad reference changes, Appendix C

Loop, 6-11, 7-19  
Loop control, 3-2

MC6845 bus cycles, 7-7



Operating temperature, 2-4  
Output section, 4-3  
Overdrive, 1-3

Physical specifications, 2-4  
Pod, I/O Module to, 4-8  
Pod sync mode, 4-13  
Program example, 7-2

Read cycle, 7-3  
readout, 4-14  
Ready button, 4-1  
Receive section, 4-3

Self test, 4-6  
Softkey operations, 6-2  
Special characters, 6-9  
Specifications, 2-1  
START, 4-1, 5-2  
STOP, 3-3, 4-1, 5-2, 6-11  
Stopping vector driving, 3-3  
Storage/shipping temperature, 2-4  
strobeoutclock, 5-3, Appendix A  
sync, 4-13, 4-14  
Synchronized data gathering, 3-2  
Synchronizing multiple module vectors, 4-16  
Synchronizing the module to the UUT, 4-11  
Synchronizing the UUT, 3-1  
Synchronizing the Vector Output I/O Module, 3-1  
syncoutput, 3-1, 5-3, 7-21, Appendix A

Testing the video board, 7-12  
TL/1 commands, Appendix A  
TL/1 vector file generation, 7-21  
Timing sets, 7-19  
Transition counter, 4-13  
TRISTATE, 4-2, 4-10  
Tri-state control, 3-4

VECTOR, 6-1  
Vector drive status, 5-1, Appendix A  
Vector driving, 3-2  
Vector file generation, 7-10, 7-21  
Vector file programming, 7-6

Vector loading, 3-2  
Vector output clock source, Appendix A  
Vector output operation, 3-1  
vectordrive, 5-3, 6-7, 6-12, Appendix A  
vectorload, 5-3, Appendix A  
Video RAM bus cycles, 7-9  
  
WAIT, 3-2, 4-2, 5-1, 6-12  
WAIT application, 7-18  
Working with the vector file, 6-4  
writepin, 3-2, 5-3  
writeword, 3-2, 5-3