



**CY512**

**INTELLIGENT POSITIONING  
STEPPER MOTOR CONTROLLER**

Cybernetic Micro Systems software products are copyrighted by and shall remain the property of Cybernetic Micro Systems, Inc. Duplication is subject to a license from Cybernetics. Cybernetic Micro Systems, Inc. reserves the right to make changes in its products without notice in order to improve design or performance characteristics. Cybernetic Micro Systems, Inc. assumes no responsibility for the use of any circuitry other than circuitry embodied in Cybernetic products. No other circuit patent licenses are implied.

Information furnished by Cybernetic Micro Systems, Inc. is believed to be accurate and reliable. However, no responsibility is assumed by Cybernetic Micro Systems, Inc. for its use, nor for any infringements of patents or other rights of third parties which may result from its use. No license is granted by implication or otherwise under any patent or patent rights of Cybernetic Micro Systems, Inc. Further, Cybernetic Micro Systems, Inc. reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation to notify any person or organization of such revision or changes; and Cybernetics assumes no responsibility for any errors which may appear in this document and makes no commitment to update the information contained herein.

The following are trademarks of Cybernetic Micro Systems, Inc:

Bin-ASCII  
CYMPL  
Analog-ASCII  
ASCII-Analog

Copyright 1983 by CYBERNETIC MICRO SYSTEMS, INC.  
All rights reserved; no part of this publication may  
be reproduced without the prior written permission of

**Cybernetic Micro Systems, Inc.**

Box 3000 • San Gregorio CA 94074 USA  
Tel: 650-726-3000 • Fax: 650-726-3003  
[www.ControlChips.com](http://www.ControlChips.com)  
[info@ControlChips.com](mailto:info@ControlChips.com)



# CY512

WITH ASCII/BINARY  
POSITION READOUT & AUTOMATIC  
DIRECTION FINDING

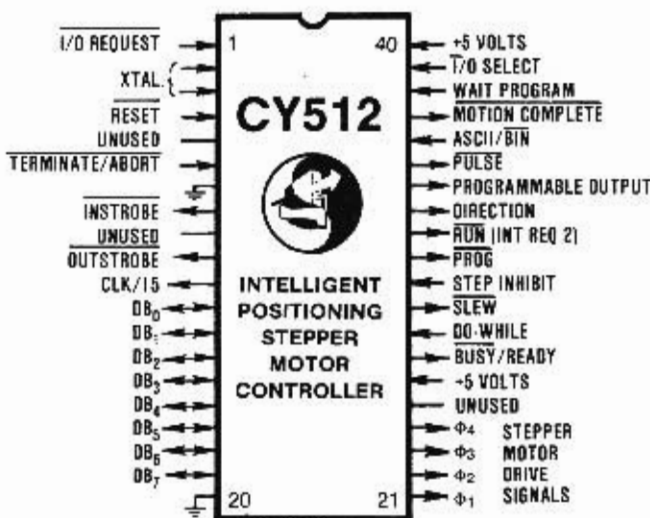
## INTELLIGENT POSITIONING STEPPER MOTOR CONTROLLER

The CY512 intelligent positioning stepper motor controller is a standard 5 volt, 40 pin LSI device configured to control any 4-phase stepper motor. The CY512 will interface to any computer using parallel TTL input and provides numerous TTL inputs and outputs for auxiliary control and interfacing. The CY512 allows sequences of hi-level type commands to be stored internally in a program buffer and be executed upon command. The TTL outputs sequence the stepper drive circuits that consist of standard power transistors or transistor arrays. When absolute position commands are executed, the CY512 automatically determines whether it is necessary to move CW or CCW to reach the specified target position.

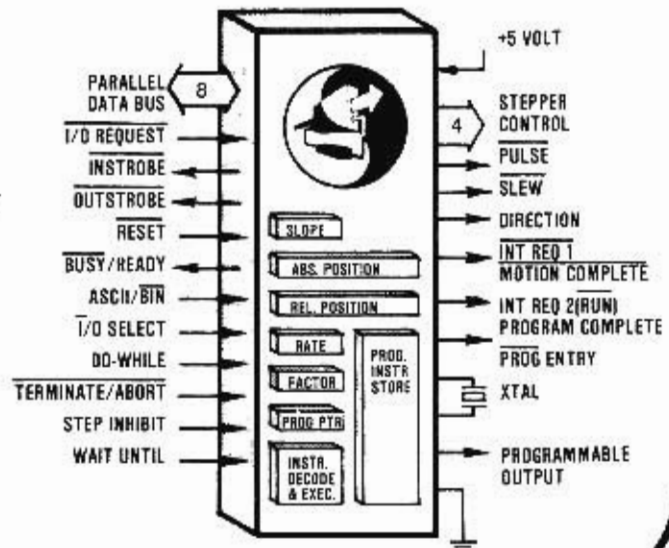
### STANDARD FEATURES

- ASCII-DECIMAL OR BINARY COMMUNICATION
- SINGLE 5 VOLT POWER SUPPLY
- 25 HI-LEVEL LANGUAGE COMMANDS
- STORED PROGRAM CAPABILITY
- HALF-STEP/FULL-STEP CAPABILITY
- ABSOLUTE/RELATIVE POSITION MODES
- PROGRAMMABLE VIA ASCII KEYBOARD
- 8000± STEPS PER SECOND (11 MHz XTAL)
- PROGRAMMABLE OUTPUT LINE
- TWO INTERRUPT REQUEST OUTPUTS
- MORE LINEAR RAMP THAN CY500
- HIGHER RATE RESOLUTION THAN CY500
- PROGRAMMABLE DELAY
- SOFTWARE DIRECTION CONTROL
- HARDWARE/SOFTWARE START/STOP
- 'ABORT' CAPABILITY
- AUTOMATIC DIRECTION DETERMINATION
- RAMP-UP/SLEW/RAMP-DOWN
- VERIFY REGISTER/BUFFER CONTENTS
- STEP INHIBIT OPERATION
- 'DO-WHILE' AND 'WAIT-UNTIL' COMMANDS
- 'JUMP TO' COMMAND
- SEVERAL SYNC INPUTS AND OUTPUTS
- 'SLEWING' INDICATION OUTPUT
- 'TERMINATE' STEP LINE FOR MAX ACCELERATION
- LOOP COMMAND WITH REPETITION COUNT

### PIN CONFIGURATION



### LOGIC DIAGRAM



Cybernetic Micro Systems

# TABLE OF CONTENTS

## SECTION 1

### INTRODUCTION TO THE CY512

CY512 Intelligent Positioning Stepper Motor Controller.....	6
Stored Program Peripheral Controller.....	8
Architecture of the CY512 Stepper Controller.....	9
Absolute vs. Relative Position.....	10

## SECTION 2

### OVERVIEW OF PIN FUNCTIONS

Communication with CY512.....	11
Keyboard Programmable Device.....	11
Synchronization Mechanisms.....	12
CY512 Pinout Diagram.....	14
CY512 Pin Description.....	15

## SECTION 3

### OVERVIEW OF CY512 COMMAND LANGUAGE

"Bin-ASCII" <sup>TM</sup> Feature.....	17
High-Level Language Design Facilitates Programming.....	18
CY512 Command Summary.....	19
Description of Commands.....	20

## SECTION 4

### DETAILED EXAMPLES OF COMMANDS

Reset Command (Initialize).....	26
Program Execution Mode: "Run" mode operation.....	26
Home Position.....	26
Program Looping, Iteration.....	26
Welding Machine Example.....	30
Operational Mode Summary.....	31

## SECTION 5

### BINARY DATA MODE OF CY512 OPERATION

Binary Data Mode.....	32
Internal Program Storage.....	34
Interface Example.....	36

## SECTION 6

### READ-OUT OPERATIONS OF THE CY512

Verify Mode Operation.....	37
----------------------------	----

**SECTION 7**

**CY512 TIMING AND CONTROL INFORMATION**

CY512 Handshake Timing Information.....40  
Doitnow "Run" Timing.....41  
Step Inhibit Pin.....42  
Direction Control.....42  
Step Timing Signals.....43  
Stop Operation.....45  
Software Abort.....46  
Half Step or Full Step.....47

**SECTION 8**

**CY512 STEP RATE INFORMATION**

Rate Control.....49  
Ramping Mode of Operation.....52  
Display of Ramped Operation.....54  
Slew Mode Operation.....58  
Closed Loop Control.....59

**SECTION 9**

**CY512 ELECTRICAL SPECIFICATIONS**

Operating Characteristics .....61  
Electrical Conventions.....61  
Reset Circuitry.....62  
Clock Circuits.....62

**SECTION 10**

**MISCELLANEOUS CIRCUITS AND EXAMPLES**

CY512 / Kit .....63  
Test Demonstration Circuit.....64  
Driver Circuit Considerations.....65  
Handshake Protocol.....68  
Operation of Several CY512s Using a Common Data Bus.....69  
Synchronization of Two CY512s.....70  
Coordination of Several CY512s.....71  
Example Programs and Waveforms.....72  
RS-232-C Receive Only Interface Design.....75  
Prom Stand-alone Interface Design.....77

**SECTION 11**

**COMPUTER CONTROL OF CY512**

Computer Control of CY512.....79  
Enter/Quit Programming Mode.....79  
CY512 Stand-alone Applications.....80  
Programming Examples.....81  
Equate Table for Programming Examples.....82  
Binary Data Programming Example.....83  
Handshake Subroutine.....84  
ASCII Data Programming Example.....85  
Oscilloscope Display Example.....86

**SECTION 12**

**IEEE-488 INTERFACE TO CY512**

IEEE-488 Interface.....87  
GPIB Handshake Signals.....87  
GPIB Interface Management Signals.....89  
GPIB Schematic Example.....91

**SECTION 13**

**GETTING YOUR CY512 RUNNING**

Start-up Procedure.....95

INDEX.....97

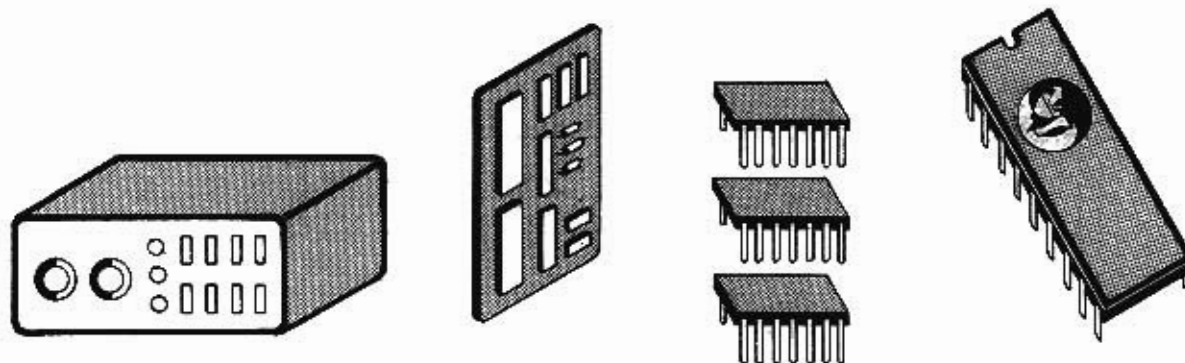


Figure 1.1 Controller evolution, boxes to boards to chips.

The next design phase reduces the random logic of the translator modules into a small number of integrated circuits. About the same time these chip sets became available, the low cost computer came into fashion. It became a natural source of input signals to run the stepper motor controller chip set, providing a pulse train to the translator module. If the loading of the motor was such that acceleration and deceleration was required, then the computer provided the timing of the pulses to affect the acceleration. If, further, the position control required complex motions that were relative to either the current location or to some absolute coordinate system, then the computer also provided these calculations. If the sequence of motions was to be synchronized with other external events, the computer provided such synchronization. At this point, the control of motion became a non-trivial problem, and the programming of a computer to provide this control represented a major design effort. If not one, but many, motors were to be controlled, the problem became even worse, and quickly exceeded the capabilities of the low cost computer.

At this point the single-chip stepper motor controller was introduced, culminating the controller evolution by placing all the control logic into a single part. The advantages of a single controller I.C. are increased system flexibility and reliability, and decreased overall systems cost.

The CY512 Intelligent Positioning Stepper Motor Controller not only contains the timing logic of the earlier bulky designs, it also offers parametric control over step rate, acceleratin rate, number of steps, and direction of stepping. In addition, an internal program buffer can be used to store commands and repeat complex motions, and secondary control lines allow these motions to be synchronized with events external to the motor itself. The central computer can now easily control several motors and take care of system level tasks, while a smart controller handles the details of running each motor.

# CY512

## INTELLIGENT POSITIONING STEPPER MOTOR CONTROLLER

The CY512 is an ASCII-programmable, peripheral controller chip designed to control stepper motors using an instruction sequence that may be stored internally in a program buffer. This feature allows the user to program the device with an ASCII keyboard and vastly simplifies prototype development and experimentation. When the user decides that the control sequence is correct, the ASCII keyboard is replaced by a computer output port, and the motor can be brought on-line. Of course, the computer can be used initially in those systems in which keyboard programming is impractical, but most applications can usually benefit from the immediacy of the keyboard during the development phase. In this mode the user simply types a command on the keyboard and the controller takes the appropriate action. In the Command Mode, the controller simply executes the command. In the Programming Mode, the command is stored in sequence in the on-chip program buffer for later execution.

Early stepper motor controllers consisted of bulky boxes controlled by switches and buttons. Step rates were set in hardware, as were the acceleration and deceleration characteristics. Switches were used to set the number of steps and direction of stepping. Buttons were used to actually start the motion. These controllers were obviously meant for manual operation. They were very expensive, very heavy, and very large when compared to the motors to be controlled.

In the next stage of controller design, the functions of the controller boxes were designed onto single PC boards. These significantly reduced the cost and packaging requirements, but did not increase the capability of the controller. One important benefit of this design was the ability to simulate switch inputs electronically, allowing another machine to command the controller. Pulse-to-step translator modules, still popular today, are also designed in this format. They require pulse and direction inputs, and translate these signals into the driving waveforms for the motors. Some translators also incorporate acceleration and deceleration capabilities.



## STORED PROGRAM PERIPHERAL CONTROLLER

The Cybernetic Micro Systems CY512 Intelligent Positioning Stepper Motor Controller is the second peripheral controller device to offer the user stored program capability, following the example set by the CY500 Stored Program Stepper Motor Controller. This feature significantly increases the power of the device and, as a consequence, decreases the amount of host time and software required to perform a given task. Stored program devices operate in three basic modes:

### Command execution mode

The CY512 executes commands as they are received.

### Program entry mode

The CY512 stores commands in an internal program buffer for later execution.

### Program execution mode

The CY512 executes the commands that were previously stored in the program buffer.

In addition to the Command Execution mode common to all peripheral controllers, the stored program controller can be placed in a Program Entry mode in which the sequence of commands is entered and stored in the program buffer, and then the device can be placed in the Program Execution mode in which stored sequences of commands are executed. The ASCII command language allows the user to program the device with an ASCII keyboard and vastly simplifies prototype development and experimentation. The keyboard can of course be replaced by a computer output port, and the functions generated on-line. However most applications can benefit from the immediacy of the keyboard during development.

In many applications the user will find that the CY512 can function as a stand-alone device, completely independent of the host processor, except for program loading. In most of these applications, it may be possible to generate custom devices that load the desired program upon power-up and are triggered by external hardware. The user can then employ these custom controllers in stand-alone applications with no host. See PROM Stand-alone example in Section 10.

# ARCHITECTURE OF THE CY512 STEPPER CONTROLLER

The CY512 architecture may be partitioned into several functional subsystems:

1. Input data subsystem
2. Output data subsystem
3. Program parameter storage
4. Mode flags and pins
5. Program storage buffer
6. Instruction selection, decoding, and control mechanisms.
7. Position Register

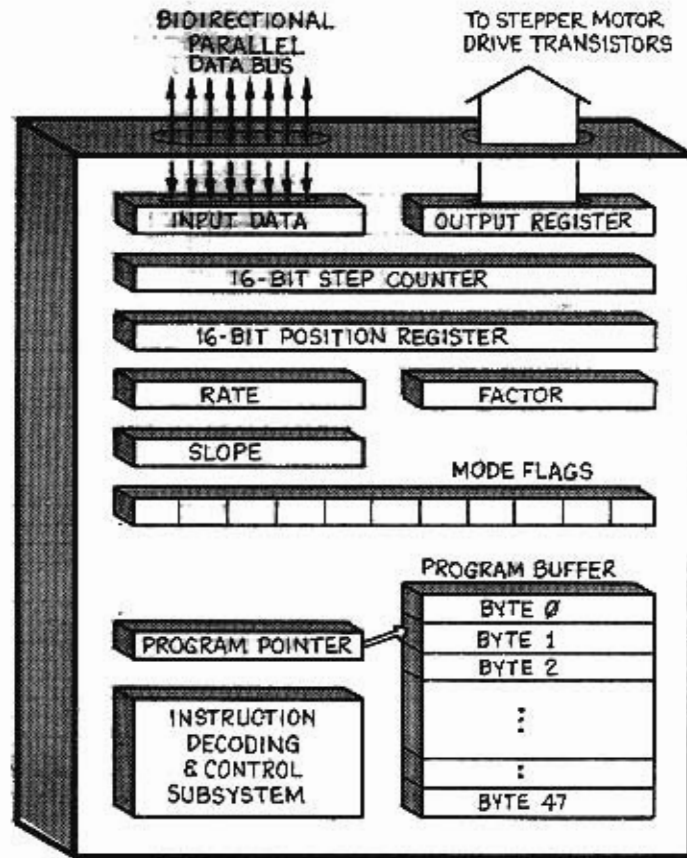


Figure 1.2 Schematic diagram of the architecture of the CY512 Intelligent Positioning Stepper Motor Controller.

## Input and Output Data Subsystems

The input data subsystem accepts commands and the output data subsystem holds the output control signals to the stepper drive circuitry and includes the associated direction and pulse timing lines.

## Program Parameter Storage

The program parameter storage subsystem is used to store the step rate parameters, ramp rate parameter, and to maintain a 16-bit position register. The position register is incremented (or decremented) when stepping in the clockwise (or CCW) direction. The position register is used when absolute position commands are specified. The 16-bit step counter is used when relative commands are employed. The contents of the position register change with every step, while the step counter register contents remain unchanged until a specific command is used to change them.

## Mode Flags and Pins

The mode flags and mode select pins are used during command execution to perform the appropriate action or to interpret data or input signals correctly.

## Program Storage Buffer

The CY512 contains a program buffer that allows the user to store a sequence of instructions that can be executed upon command. This provides all of the benefits of stored program execution that have made computers such powerful tools.

## Instruction Decoding and Control

This subsystem performs the actual execution of commands.

## Position Register

The CY512 contains a 16-bit position register that can be read to determine the current location. The CY512 will accept relative and absolute position commands; however, the position register always indicates absolute position.

# ABSOLUTE VS. RELATIVE POSITION

The CY512 default mode is the relative position mode, in which total travel is specified relative to the current position via the Number (of steps) command,  $N\ n$ , where  $0 < n < 2^{16} - 1$  (65535). In this mode an internal counter is decremented for each step (or half step if appropriate) and stepping continues until the count reaches zero (or another Halt condition is detected). If the Position mode command,  $P\ p$ , is received, the target position "p" is interpreted as absolute position with respect to the zero location declared by the Athome command. The CY512 calculates the stepping direction and the number of steps to take to reach the specified target position. **After moving to the specified position, the system reverts to the relative mode.** Note that the relative mode is selected by the G command, and the absolute mode is selected by the P command. When an actual stepping operation is in progress, both a number of steps value and a target position are used to internally execute the step command. The current position register is updated in both the relative and absolute modes, so motions may be mixed between the modes. In relative mode, the target position will be calculated, and in absolute mode, the number of steps to take will be calculated. After that, the two modes use the same internal stepping routines. Acceleration and deceleration work in both stepping modes, with the CY512 also calculating the position at which to start deceleration in order to return to the starting step rate when the motion is completed.

## COMMUNICATION WITH CY512

Commands can be issued to the CY512 using a parallel data format. In parallel operation, complete handshaking operation occurs via the use of a Busy/Ready line on the CY512, and the I/O Request strobe line from the host. The handshake protocol is shown below.

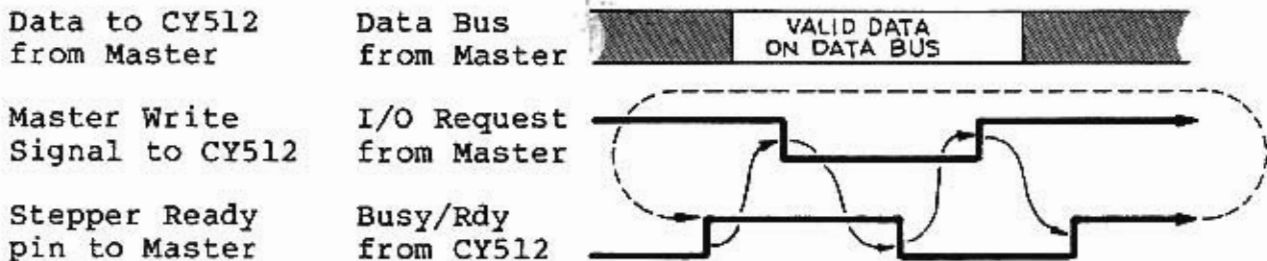


Figure 2.1 Handshaking protocol for CY512 parallel input.

## KEYBOARD PROGRAMMABLE DEVICE

The CY512 Stepper Motor Controller offers Hi-Level Language programming with an ASCII keyboard. This design allows the user maximum utility via the closest possible coupling and facilitates interactive prototype development and debugging. Note that the stored program capability makes it possible in many cases to perfect the operation of the stepper motor completely decoupled from the host computer. In such cases, the host processor is required to do little more than load the programs at appropriate times. Of particular importance in many applications is the dynamic stability of the system. By programming a range of test conditions through the keyboard, the designer may exercise the system over broad ranges and thus characterize the system dynamically. Of course, any designer with access to an easy-to-use, interactive host computer can achieve everything that the keyboard user can, and more. Lacking such systems, the designer will appreciate the extreme power of keyboard programming during prototyping phases, thus postponing until final systems integration the slower, costlier, host computer programming associated with all host-controlled controller devices.

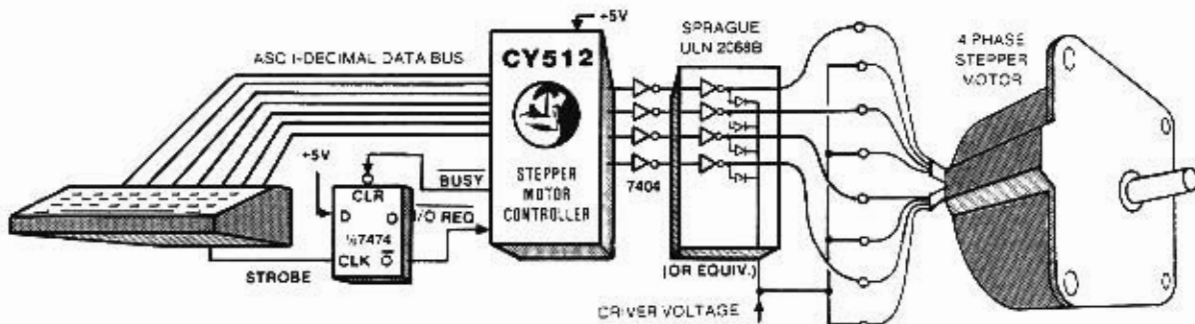


Figure 2.2 Simplest Prototype Development System.

## SYNCHRONIZATION MECHANISMS

Most stepper motors are employed as parts of functional systems. These systems often must synchronize the behavior of the various subsystems to each other or to a real-world occurrence, such as an operator input. The CY512 has been designed with both signal emitters and detectors to allow easy synchronization of the device to neighboring (interacting) subsystems.

The motor interface for the CY512 is very simple, consisting of six output signals. Since the controller is designed for four phase motors, there is a signal line for each phase. The patterns necessary to operate the motor in a full step or a half step mode, including sequencing for proper direction, appear on the phase outputs. A simple L/R type driving circuit may be connected directly to the phase outputs, so the motor can be run from the controller signals. Alternatively, the user could drive a more sophisticated pulse-to-step translator, using the CY512 Pulse and Direction outputs. The Pulse line gives one pulse at the beginning of each step, while the Direction line always indicates the current stepping direction.

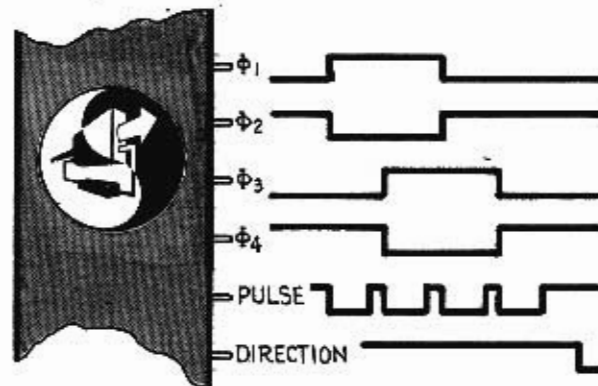


Figure 2.3 Motor Interface

The computer or data interface of the CY512 is also very simple. Commands and parameters are passed from the command source to the CY512 on an eight bit, bidirectional data bus. Direction of data is determined by the level on the I/O Select line, allowing the CY512 to not only receive commands, but also to be interrogated for the current values of its parameters and contents of the program buffer. Data transfer between the command source and the CY512 is controlled by a standard two-line handshake protocol. The master processor waits for the CY512 Busy/Ready line to go high, indicating that the CY512 is ready for the next command byte. Data may then be placed on the bus, and data available is indicated by a high-to-low transition of I/O Request. Data should remain stable until the CY512 indicates data accepted by a high-to-low transition of the Busy/Ready line. During this busy time, the CY512 is processing the character just received. The master processor should then raise I/O Request and wait until the CY512 is ready for the next data byte. Data transfers from the CY512 to the master processor are handled in a similar way, with

the master requesting the next byte using I/O Request, and the CY512 indicating data available using Busy/Ready. The simplicity of the data transfer handshake, combined with the ASCII command structure of the CY512, allows the commanding device to be any of a number of things, including a microprocessor or other computer, a keyboard for manual command entry, or a ROM for fixed, stand-alone applications. The keyboard is especially useful during prototype development or system characterization.

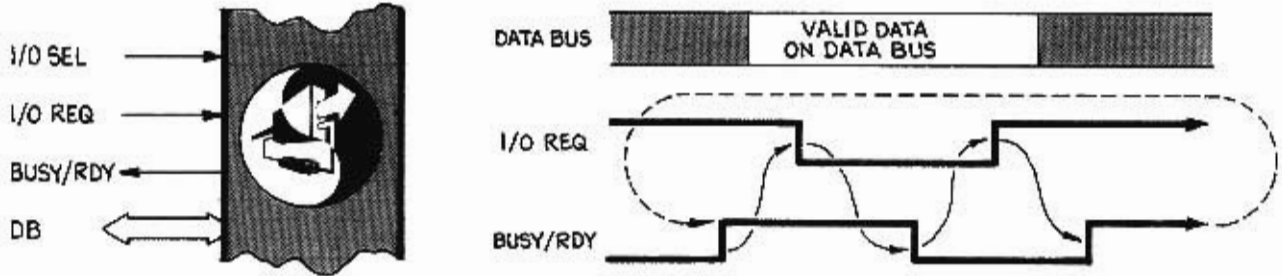


Figure 2.4 Data interface and handshake waveform

Since most stepper motors are parts of functional systems, requiring that various parts of the system stay synchronized with each other, the CY512 has been designed with a number of secondary input and output control lines. These signals may be used to modify and control the stepping behavior of the device, or indicate certain conditions within the controller. Two inputs control the stepping behavior directly. While Step Inhibit is high, the controller will not step. Stepping is resumed when the signal goes low again. This signal may be used to halt a motion under emergency conditions, or to slow the step rate if the motor cannot keep up. Conversely, the Terminate/Abort signal is used to speed up the rate (Terminate), or to cause a deceleration to the starting rate (Abort), at which point the motion may be stopped. Two other inputs modify the way a program is executed. The Wait line is used to suspend a program until the signal level on that line is in a certain state. Commands allow the program to wait for either a high level or a low level, making it possible to synchronize on either transition of the line. The DOWHILE input is used with the conditional loop command. While the line is low, the CY512 will loop back to the beginning of the program, repeating the program section over and over. When the line goes high, the controller will continue with the rest of the program.

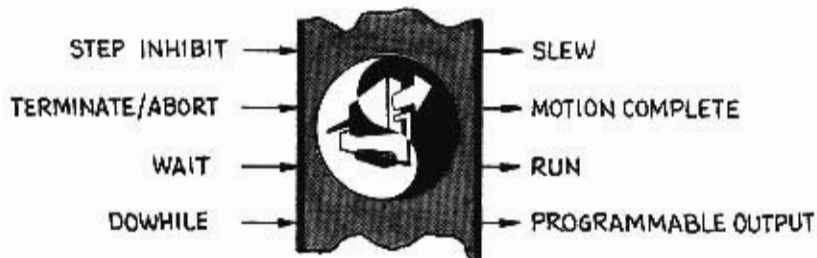


Figure 2.5 Secondary control inputs and outputs

The CY512 also provides a number of output signals which may be used by other parts of the system. While stepping, the Slew line indicates that the CY512 has reached the maximum programmed step rate, and is not accelerating or decelerating. When the CY512 has stepped for the number of steps specified, the Motion Complete signal indicates the end of the current motion. Run is used to indicate that a program is executing. In addition, the CY512 provides an uncommitted output, Programmable Output, which the user may apply as needed. The level on this output is controlled by two commands, one for a high output, and the other for a low output.

## CY512 PINOUT DIAGRAM

The CY512 pinout diagram is shown below, followed by the table of pin definitions.

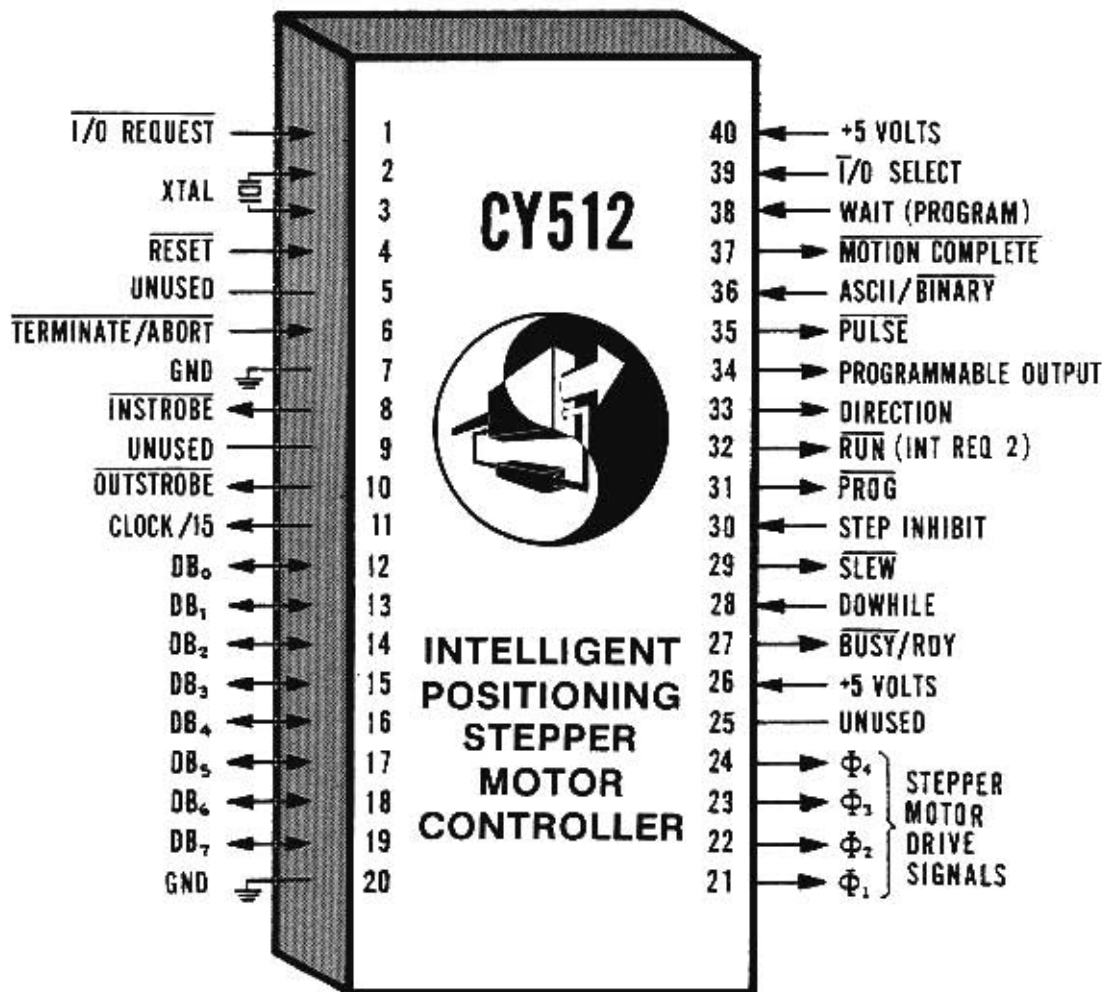


Figure 2.7 CY512 Pin definition

TABLE I

CY512 PIN DESCRIPTION

DESIGNATION	PIN#	FUNCTION
V <sub>CC</sub>	40	+5 volt power supply.
V <sub>DD</sub>	26	+5 volts.
V <sub>SS</sub>	7,20	circuit GND potential.
DB <sub>0</sub> -DB <sub>7</sub>	12-19	bidirectional parallel data bus.
Φ <sub>1</sub> -Φ <sub>4</sub> (output)	21-24	stepper drive signals.
Direction (output)	33	indicates current stepping direction and is affected by +, -, and "P" commands (Hi = CW, low = CCW).
Pulse (output)	35	low when step begins, high when step ends or is externally terminated by Terminate/Abort line.
Motion Complete (Int Req 1) (output)	37	signal to interrupt host at end of stepping.
Wait (input)	38	program Waits for this pin to go LOW when "Until" command is executed, and waits for a High signal when "Wait" command is executed.
Terminate/Abort (input)	6	Low during step Terminates (completes) step prematurely. Rising edge of Pulse may be used to bring line high again (this combination may be used with Step Inhibit to maximize step rate). If held low, the controller Aborts high speed stepping, ramps down and continues stepping to target position at slowest step rate.
Reset (input)	4	initializes controller to power-up state.
I/O Request (input)	1	strobe to initiate command input when writing to CY512. Initiate data output when reading from CY512. Interpretation of pin 1 is a function of I/O Select (pin 39).
I/O Select (input)	39	indicates direction of data on the data bus. Low = input to CY512. Hi = output from CY512, which can only be generated if CY512 has received "V" command.



TABLE I

## CY512 PIN DESCRIPTION

(continued)

DESIGNATION	PIN#	FUNCTION
Unused	5,9,25	must remain disconnected.
$\overline{\text{Busy/Ready}}$ (output)	27	handshake line for command data input. Host must wait until Ready state is indicated by a high level before transferring command or data to CY512. If Run (pin 32) is low, Ready is invalid, since CY512 can not be written to while program is executing.
$\overline{\text{Prog}}$ (output)	31	indicates program entry mode. Commands are entered and saved, but not executed, while pin 31 is low.
$\overline{\text{Run}}$ ( $\overline{\text{Int Req 2}}$ ) (Program Complete) (output)	32	indicates program execution mode. Commands cannot be entered while program is executing (pin 32 = low).
$\overline{\text{Slew}}$ (output)	29	goes low when maximum specified stepping rate has been achieved. Will return high when deceleration begins.
Dowhile (input)	28	is tested when "T" command is encountered in program. If low, program will continue looping; if high, program will fetch next instruction.
Step Inhibit (input)	30	inhibits stepping while held high.
Programmable Output (output)	34	user programmable output pin.
ASCII/ $\overline{\text{Binary}}$ (input)	36	selects ASCII-decimal or binary mode of operation.
Xtal <sub>1</sub> -Xtal <sub>2</sub> (input)	2,3	inputs for crystal or external clock (not TTL). See Clock Circuits section.
Clk/15 (output)	11	This output represents the crystal frequency divided by fifteen. The pulse width is at least 300 nanoseconds.
$\overline{\text{Instrobe}}$ (output)	8	occurs during data input. The data on the bus must be valid until the trailing edge of Instrobe occurs.
$\overline{\text{Outstrobe}}$ (output)	10	used with "V" command. Trailing edge indicates valid data output by CY512 on data bus.

# 3 OVERVIEW OF COMMAND LANGUAGE 3

## "BIN-ASCII"™ FEATURE

The CY512 user-orientation has been accomplished without the expense of complicating the host programming job. For example, the ASCII-decimal integers typed by the user at the keyboard may not be readily available in the host programming language. For this reason the CY512 can be placed in a binary mode in which binary number parameters are used instead of ASCII-decimal. This allows any computer with binary integer arithmetic to send commands and binary information to the controller. The CY512 is placed in either the binary or the ASCII-decimal mode via a mode-select input pin setting.

The use of ASCII instruction and ASCII-decimal integer parameters allows the user to type commands in familiar high-level language formats, as shown below:

```
N 738) ;set Number of steps = 738
G) ;Go (begin stepping)
```

where "N" is the ASCII command specifying NUMBER of steps to take. The ASCII space character is shown as a space, and the decimal number "738" is then entered, followed by the carriage return key, ")"= 0DH which terminates the commands. The GO command is entered as "G)". The controller then steps the motor for 738 steps. Other parameters, such as rate, may be specified in similar fashion.

Although the use of ASCII-decimal numbers is ideal for the user employing BASIC or other languages that can output ASCII-decimal numbers, it is, of course, desirable that the controller accept binary number parameters from binary computations. For this reason, the CY512 Stepper Controller may be placed in a BINARY mode via a strap, or mode-select, pin. In this mode, all numbers are interpreted as binary data (as are all commands). See the section on binary data mode for details.

## HIGH-LEVEL LANGUAGE DESIGN FACILITATES PROGRAMMING

The primary advantage of all hi-level languages is their problem-oriented nature, as opposed to the device-oriented nature of machine languages. A secondary characteristic is their ASCII representation, and a third characteristic of most hi-level languages is their use of the ASCII-decimal numbering system as natural numbers. In all of these aspects, the CY512 qualifies as a single chip Hi-Level Language Device. The combination of hi-level language and ASCII-keyboard programmability is designed to maximize user ease and convenience.

Every instruction entered in the ASCII decimal mode of operation consists of one of the following forms:

1. Alphabetic ASCII character followed by the "}" (RETURN) key.
2. Alphabetic ASCII character followed by space, then ASCII decimal number parameter, then "}" = 0DH.

Examples of type one are as follows:

NAME	COMMAND	INTERPRETATION
Athome	A}	Declare absolute zero location
Bitset	B}	Set programmable output line
Clearbit	C}	Clear programmable output Line
Doitnow	D}	Do program (begin running program)
Enter	E}	Enter program mode

Examples of type two are as follows:

NAME	ASCII COMMAND	INTERPRETATION
Number	N n}	Declare number of steps to be taken (relative)
Rate	R r}	Declare maximum rate parameter
Slope	S s}	Declare ramp rate
Factor	F f}	Declare rate modification factor
Position	P p}	Declare target position (absolute)

TABLE II

CY512 COMMAND SUMMARY

ASCII CODE	NAME	INTERPRETATION
A	Athome	Set current location as absolute zero
B	Bitset	Set programmable output line high
C	Clearbit	Reset programmable output line low
D	Doitnow	Begin program execution
E	Enter	Enter program code
F	Factor	Set factor parameter for step rate
G	Go	Begin relative stepping operation
H	Halfstep	Set halfstep mode of operation
I	Initialize	Turn off step drive lines, reset controller
J	Jump	Go to specified program buffer location
L	Loop	Repeat program segment for specified count
N	Number	Set number of steps to be taken (relative)
O	Offset	Set next stepper drive signal value
P	Position	Set and step to target position (absolute)
Q *	Quit*	Stop saving program, enter command mode. also quit stepping. <u>*Never followed by "</u> "
R	Rate	Set step rate parameter
S	Slope	Set ramp rate for slew mode operation
T	Loop Til	Loop "Til" dowhile line goes high
U	Until	Stop execution until wait line is low
V	Verify	Verify internal buffer contents
W	Wait	Stop executing until wait line is high
X	eXpend	Time delay for specified milliseconds
+	CW	Set clockwise direction
-	CCW	Set counterclockwise direction
∅	Command	Stop program execution, enter command mode

## DESCRIPTION OF COMMANDS

The command format shown in the following descriptions indicates the way commands are stored in the program buffer, as well as showing the binary values of the command letters. Note that in Binary mode the user must insert a data count between the command letters and parameters, if any. See section 10 on Binary Data Mode. Also note that 16 bit parameters (number and position) are entered least significant byte first in the Binary mode. In the ASCII mode, command letters are separated from parameters by a single space, and the parameters are entered as ASCII decimal numbers. ASCII mode commands are terminated by a carriage return, as indicated in the leftmost column of the command description.

A) **ATHOME** 0100 0001 1 byte

The ATHOME instruction defines the "Home" position. This position is set to absolute zero, and is the reference for all POSITION commands. The ATHOME command may be used at any time to define or redefine position zero.

B) **BITSET** 0100 0010 1 byte

This instruction causes the programmable output pin (#34) to go HIGH. This is a general-purpose output that may be used in any fashion.

C) **CLEARBIT** 0100 0011 1 byte

This instruction causes the programmable output pin (#34) to go LOW. The user can signal locations in a program sequence to the external world via B and C instructions.

D) **DOITNOW** 0100 0100 1 byte

This instruction causes the CY512 to begin executing the stored program. If no program has been entered, the controller will stay in the Run mode unless a Stop Operation is executed. If the program exists, the controller will begin execution of the first instruction in the program buffer. If the run (DOITNOW) command is encountered during program execution, it restarts the program (however, the initial parameters and modes may have been redefined later in the program) and may be used for looping or cyclic repetition of the program.

E) **ENTER** 0100 0101 1 byte

This instruction causes the CY512 to begin the program entry mode of operation. All commands following the ENTER command are saved in the program buffer in sequence until "Q" is entered. The PROG line (# 31) goes low to indicate this mode.

F f) **FACTOR** 0100 0110 2 bytes  
a7 a0

The factor, *f*, is a number from 1 to 255 and is used to further define the step rate. See the Rate Equation in Section 8 for details.

G) **GO** 0100 0111 1 byte

The GO command causes the stepper motor to step as specified by the rate, direction, etc., commands entered prior to the GO command. Stepping will be in the relative mode, with the number of steps defined by the N command.

H) **HALFSTEP** 0100 1000 1 byte

The HALFSTEP command causes the CY512 to enter the halfstep mode and remain in that mode until the device is reset or reinitialized.

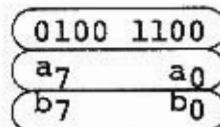
I) **INITIALIZE** 0100 1001 1 byte

The INITIALIZE command causes the CY512 to enter the command mode. None of the distance or rate parameters are altered. Any commands following "I" will be executed with the parameters specified prior to "I". The INITIALIZE command, when encountered during program execution, halts the program execution and returns the system to the command mode. This command de-energizes the stepper motor coils, erases the program, if any, sets the direction to CW, and sets the full step mode (turns off half step).

J a) **JUMP** 0100 1010 2 bytes  
a7 a0

The JUMP command will branch program execution to the program buffer location specified by the argument, which represents the byte number in the program buffer, starting with zero. Program execution continues from the specified byte number after the jump is executed. When the JUMP command is issued from the Command mode, it enables the Run (Program execution) mode, and begins executing from the specified byte number. It is the user's responsibility to insure that the number specified with the JUMP command is the correct value for the desired starting point. The first byte after the "E" command is location zero. The byte count specified in this section determines the number of bytes used by each instruction. Note that "J 0)" is equivalent to "D)".

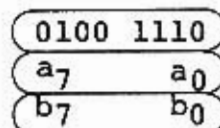
L c,a) LOOP



3 bytes  
COUNT  
LOCATION

The LOOP command uses the first argument as a repetition count, and the second argument as a jump location. Each time the LOOP command is executed, the count is decremented by one. If the count is nonzero after the decrement, program execution will jump to the specified address, which is the second argument of the command. The jump address specifies the location to which execution will loop, and the count represents the number of times the loop is to be repeated. When the count reaches zero, program execution continues with the instruction immediately following the LOOP command. In ASCII mode, the two arguments may be separated by either a comma, or a single space. LOOP commands may not be nested one inside the other.

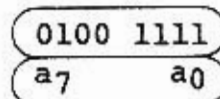
N n) NUMBER



3 bytes  
LSbyte  
MSbyte

The NUMBER command is used to specify the number of steps to be taken in the "Relative" mode of operation. The argument may be any number from 1 to 64K-1 (65,535). Note that this parameter is stored as 2 bytes in the program buffer.

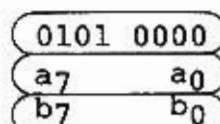
O o) OFFSET



2 bytes

The OFFSET command specifies the next step pattern to appear on the STEPPER MOTOR DRIVE SIGNALS, pins 21-24. This command may be used to synchronize these outputs with the motor when the desired pattern is known from the motor's position. See "Half Step or Full Step" discussion at the end of Section 7.

P p) POSITION



3 bytes  
LSbyte  
MSbyte

The POSITION command declares the "Absolute" mode of operation. The argument is treated as the target position relative to position zero. The ATHOME command is used to define position zero. Stepping to the target position begins when the POSITION command is executed. No "G" command is required. Direction to the target position is also determined and set automatically.

**Q**    **QUIT** (Programming)                    0101 0001                    1 byte

**NOTE:** The QUIT command is self-terminating, and should NOT be followed by the Linend ")" or data count. Such termination may result in incorrect operation.

The QUIT command causes the CY512 to exit the "Programming" mode of operation, wherein instructions are stored in the program buffer in the order received; and causes the CY512 to return to the "Command" mode of operation, in which instructions are executed as they are received. It may also be used as an emergency stop operation during a step command.

**R r)**    **RATE**                                    0101 0010                    2 bytes  
    a7            a0

The RATE instruction sets the rate parameter that determines the step rate. The rate parameter, r, varies from 1 to 255 corresponding to step rates from 50 to 4350 steps/sec (assuming a rate factor of 1 and a 6 MHz crystal). The rate is non-linear and can be computed from the rate equation or from Table VII. For crystals other than 6 MHz the step rate should be multiplied by  $f_{cy}/6\text{MHz}$ , where  $f_{cy}$  is the crystal frequency.

**S s)**    **SLOPE**                                    0101 0011                    2 bytes  
    a7            a0

The SLOPE or slew mode of operation is used when high step rates are required and the initial load on the motor prevents instantaneous stepping at such rates. In such cases, the load is accelerated from rest to the maximum rate and then decelerated to a stop. The user specifies the distance of total travel (via "N" instruction), the maximum rate (via "R") and the ramp rate or change in factor parameter from step to step. The CY512, starting from rest, decreases the factor parameter by "s" with each step until the maximum (slew) rate is reached. The device computes the "re-entry" point at which it begins decelerating (with acceleration = -s) until it reaches the final position.

**T)**    loop **TIL**                                    0101 0100                    1 byte

The "T" command provides a "Do...While..." capability to the CY512. This command tests pin 28 and, if low, it executes the DOITNOW command, i.e., it runs the program from the beginning (although using the latest rate, position, etc., parameter). If pin 28 is high, the instruction following the T command is fetched and executed.



U) wait-UNTIL (0101 0101) 1 byte

The wait-UNTIL instruction is used to synchronize the program execution to an external event. When the "U" instruction is executed, the WAIT pin (pin #38) is tested. When the WAIT pin goes low, the next instruction is fetched from the program buffer and execution proceeds.

V v) VERIFY (0101 0110) 2 bytes  
(a7 a0)

The VERIFY command allows interrogation of the internal CY512 buffers, including the rate, slope, number of steps, and current position registers. This command is also used to examine the current contents of the CY512 program buffer. The parameter "v" specifies which internal register group is to be read. VERIFY should only be executed from the command mode. Reading data during run mode may not work properly. See "Verify Mode" in Section 6 for details.

W) WAIT (0101 0111) 1 byte

The WAIT instruction is the opposite of the U command. WAIT tests the WAIT pin (pin #38) for a high level. The program will stop until the pin is high, then it will continue with the next command. Note that the U command may be used to detect the falling edge of the WAIT line signal, and the W command may be used to detect the rising edge. Thus, it is possible to synchronize program execution to either one or both of the transitions.

X x) EXPEND (0101 1000) 3 bytes  
(a7 a0) LSbyte  
(b7 b0) MSbyte

The EXPEND command will time delay for the number of milliseconds specified by its argument. The delay is calibrated in milliseconds, using an 11 MHz crystal. Other frequencies will require a linear scaling for the actual delay time. Since this is a 3 byte command (16 bit argument) the delay time can range between 1 msec and about 65.5 sec at 11 MHz. This command is useful in programming a delay time between stepping motions.

+ ) CLOCKWISE (0010 1011) 1 byte

Direction is set to clockwise by this command. Relative mode steps are taken in the direction last specified, so they will be clockwise until the direction is changed by the "-" or "P" commands. The current direction is always indicated on the DIRECTION line (pin # 33).

-> **CCW** (0010 1101) 1 byte

Direction is set to counter-clockwise by this command. Comments under CLOCKWISE also apply to the "-" command.

Ø) **COMMAND** (0011 0000) 1 byte

The CY512 is placed in the command mode and the next command is executed as it is received. Programs should be terminated by a "Ø" command, returning the CY512 to command mode at the end of program execution.

# 4 DETAILED EXAMPLES OF COMMANDS 4

## RESET COMMAND (INITIALIZE)

The "I" or Initialize command resets all pointers to the power-up state and restores the flags to this state. Specifically, the program is erased and the command mode entered. The direction is clockwise (CW). Halfstep mode is turned off. Note that this command de-energizes the stepper coils (phase outputs all go high). If this effect is undesirable, an external latch should be used to latch the four stepper control outputs using the pulse line (pin 35) to clock the latch. See Figure 4.1.

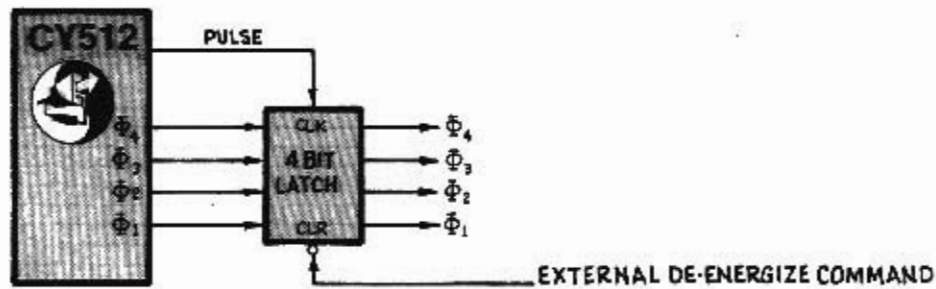


Figure 4.1 An external latch on the stepper control outputs prevents de-energizing of stepper drive coils when CY512 is reset via hardware or software and also allows an external control line to de-energize the coils independently of the CY512.

## PROGRAM EXECUTION MODE: "RUN" MODE OPERATION

Once a program is entered into the program buffer, it may be executed with a run or DOITNOW ("D") command. This code has been assigned the ASCII value "D" = 44H. It is the last command to be entered before program execution. It is a normal command in the sense that it is terminated with a carriage return, "}" = 0DH.

## HOME POSITION

The "Ahome" command, A, is used to declare the "Home" position, assigned absolute value zero. All positions specified with the POSITION command, P, are referenced to this zero position. On power-up, the absolute position is random. Therefore, the home position should be found, and the A command used, before the absolute position commands are utilized.

## PROGRAM LOOPING, ITERATION

One consequence of stored program execution is the use of program loops or program repetition. If the run (DOITNOW) command of the CY512 is included as a program instruction, the program executes again beginning with the first instruction (but using the latest

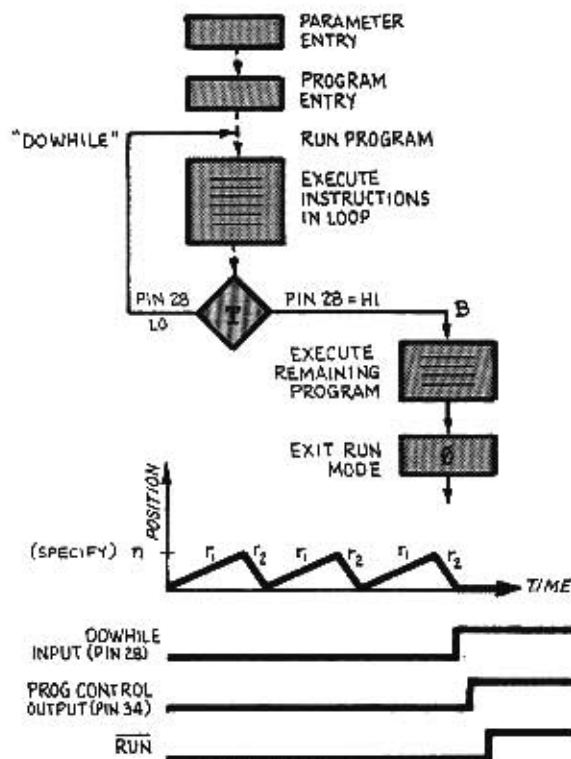
value of parameters set before the DOITNOW instruction was encountered). In this fashion, rather complex sequences of motions may be repeated without host intervention or interruption. Conditional looping may be accomplished with a "Do While" type instruction that continues looping until a condition is fulfilled. This may be combined with the JUMP command, for unconditional branching to various routines in the program buffer. Finally, a subsection of a program may be repeated a specific number of times by use of the LOOP instruction.

### Unconditional Program Looping

If the DOITNOW command, "D", is encountered in the program entry mode, it is stored in the program buffer with the rest of the program. When this instruction is encountered during the program execution, its effect is to begin program execution again, and therefore may be used to achieve cyclical looping if desired. However, program execution may be aborted via either the RESET line or the Stop Operation (Q command during stepping).

### Conditional Program Looping - (Do...While...)

The ability to repeatedly execute a program until an external event occurs provides a unique "Do...While..." capability for the CY512. The "T" command (loop TIL) is used as shown in the following example.



- N n) set number of steps = n
- E) enter program mode
- + ) set CW direction
- R r<sub>1</sub>) set rate = r<sub>1</sub>
- G) begin stepping
- ) set CCW direction
- R r<sub>2</sub>) set new rate = r<sub>2</sub>
- G) go (same "n")
- T) loop TIL pin 28 goes HI
- B) set control output line
- Ø) return to command mode
- Q quit program mode
- D) begin executing program

Figure 4.2 Conditional Loop

## Conditional Loop Embedded in an Unconditional Loop

"Do (the preceding program) While (Pin 28 is low)", then proceed to execute the remaining program instructions. Note that the program can (but need not) end with a DOITNOW instruction to provide a conditional loop inside of an unconditional loop:

```

R r1}    set rate parameter
E)       enter program mode
N n1}    set first distance
+ )      set CW direction
G)       take n1 steps
- )      set CCW direction
N n2}    distance parameter
G)       take n2 steps
T)       repeat TIL pin #28 = HI
B)       set output HI (pin 34)
R r2}    set new rate
D)       repeat program
Q        exit program mode
D)       begin executing program
    
```

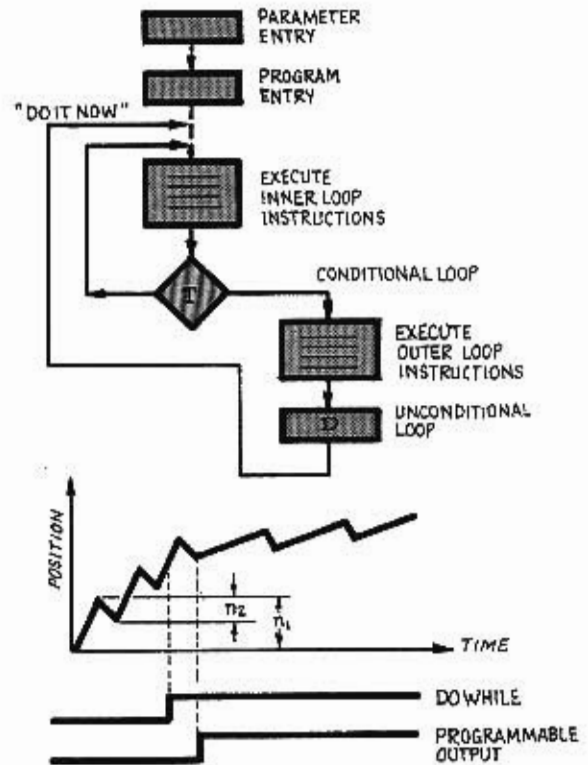


Figure 4.3  
Conditional Loop Imbedded  
in an Unconditional Loop.

The Wait line allows a program to be suspended until an external event occurs. As long as the Wait line is in one state, the program continually tests the line without executing any other instructions. When the line changes to the state being waited for, the program continues with the commands following the wait instruction. Figure 4.4 illustrates the wait Until command, for which the wait line must be low to continue.

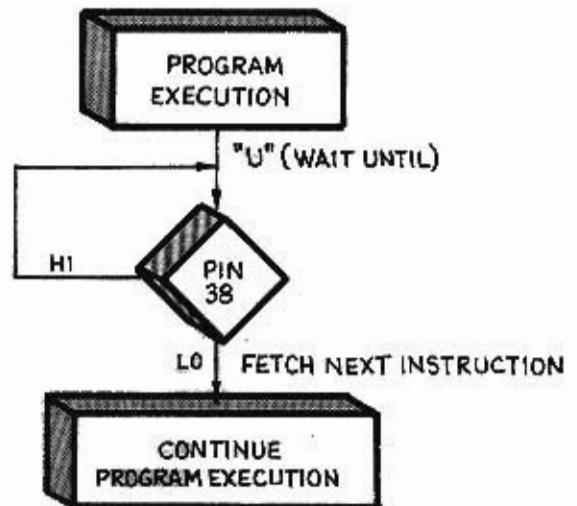


Figure 4.4 Wait UNTIL command use.

## The JUMP Command

Unconditional branching to any location in the program buffer may be accomplished by the Jump command. The single argument of the command specifies the program buffer location at which program execution will continue. This number is simply the byte number within the program buffer, with the first location designated as byte zero. Since the program buffer is 48 bytes long, arguments for the Jump command range in value from 0 to 47. The actual value used should correspond to the beginning of the instruction which is to be executed next. The byte numbers may be determined by adding the number of bytes used for each of the previous commands in the buffer. Byte counts are specified in Description of Commands in Section 3.

The following example contains a Jump command used to repeat a section of the program. The program steps to the home position at high speed, then repeats motions of 75 steps at a lower speed, waiting for a synchronizing signal before taking each motion. The Jump command may also be used in the Command mode, to start program execution at a location other than the first command of a program. This example assumes the home position has been previously defined, using the "Ahome" command.

```
E)   enter program mode
C)   lower programmable out pin
R 220) define fast step rate
P 0)  step quickly to home position
+)   CW direction for next steps
N 75) take 75 steps per motion
R 150) define slower step rate
B)   raise programmable out pin
W)   wait for high on pin 38
C)   lower programmable out pin
G)   take 75 steps
J 12) repeat from "B" command
Q    quit program mode

S 20) define acceleration slope
F 1)  define stepping factor
D)   begin running program
```

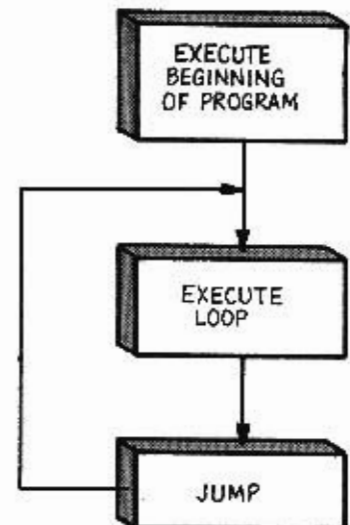


Figure 4.5 JUMP Command Use

## WELDING MACHINE EXAMPLE

Suppose we want a row of six equally spaced welds on a piece of metal. The welder should be turned on by the CY512 programmable output line when in position, and be turned off when finished. After completing six such welds, it will return as quickly as possible to its starting position, and wait for the next workpiece to come into position. It will then weld the next six spots, and continue in this manner until there are no more pieces to weld, at which time the program will stop and the CY512 will return to the Command mode.

<pre>R 180} S 35} F 9} } A) E) N 20} +) W) G) C) X 1000} B) L 6,5} P 0} T) Ø) Q D)</pre>	<pre> } define stepping parameters declare current position as home enter and save the following program take 20 steps between welds CW direction wait until workpiece is ready go 20 steps activate welder delay 1000 msec (1 sec) to weld turn off welder repeat six times from G command return to home position after 6 welds repeat program until no more pieces stop program, return to command mode exit program mode begin executing program</pre>	<p style="text-align: center; margin: 0;">CY512 PROGRAM BUFFER</p> <table border="1" style="width: 100%; border-collapse: collapse; text-align: center;"> <tr><td style="padding: 2px;">4E</td><td style="padding: 2px;">N</td></tr> <tr><td style="padding: 2px;">14</td><td rowspan="3" style="padding: 2px;">} 20</td></tr> <tr><td style="padding: 2px;">00</td></tr> <tr><td style="padding: 2px;">2B</td></tr> <tr><td style="padding: 2px;">57</td><td style="padding: 2px;">+</td></tr> <tr><td style="padding: 2px;">47</td><td style="padding: 2px;">W</td></tr> <tr><td style="padding: 2px;">43</td><td style="padding: 2px;">G</td></tr> <tr><td style="padding: 2px;">58</td><td style="padding: 2px;">C</td></tr> <tr><td style="padding: 2px;">E8</td><td style="padding: 2px;">X</td></tr> <tr><td style="padding: 2px;">03</td><td rowspan="2" style="padding: 2px;">} 1000</td></tr> <tr><td style="padding: 2px;">42</td></tr> <tr><td style="padding: 2px;">4C</td><td style="padding: 2px;">B</td></tr> <tr><td style="padding: 2px;">06</td><td style="padding: 2px;">L</td></tr> <tr><td style="padding: 2px;">05</td><td style="padding: 2px;">6</td></tr> <tr><td style="padding: 2px;">50</td><td style="padding: 2px;">5</td></tr> <tr><td style="padding: 2px;">00</td><td style="padding: 2px;">P</td></tr> <tr><td style="padding: 2px;">00</td><td rowspan="2" style="padding: 2px;">} 0</td></tr> <tr><td style="padding: 2px;">54</td></tr> <tr><td style="padding: 2px;">00</td><td style="padding: 2px;">T</td></tr> <tr><td style="padding: 2px;"></td><td style="padding: 2px;">Ø</td></tr> </table>	4E	N	14	} 20	00	2B	57	+	47	W	43	G	58	C	E8	X	03	} 1000	42	4C	B	06	L	05	6	50	5	00	P	00	} 0	54	00	T		Ø
4E	N																																					
14	} 20																																					
00																																						
2B																																						
57	+																																					
47	W																																					
43	G																																					
58	C																																					
E8	X																																					
03	} 1000																																					
42																																						
4C	B																																					
06	L																																					
05	6																																					
50	5																																					
00	P																																					
00	} 0																																					
54																																						
00	T																																					
	Ø																																					

Figure 4.6 Loop Command Use.

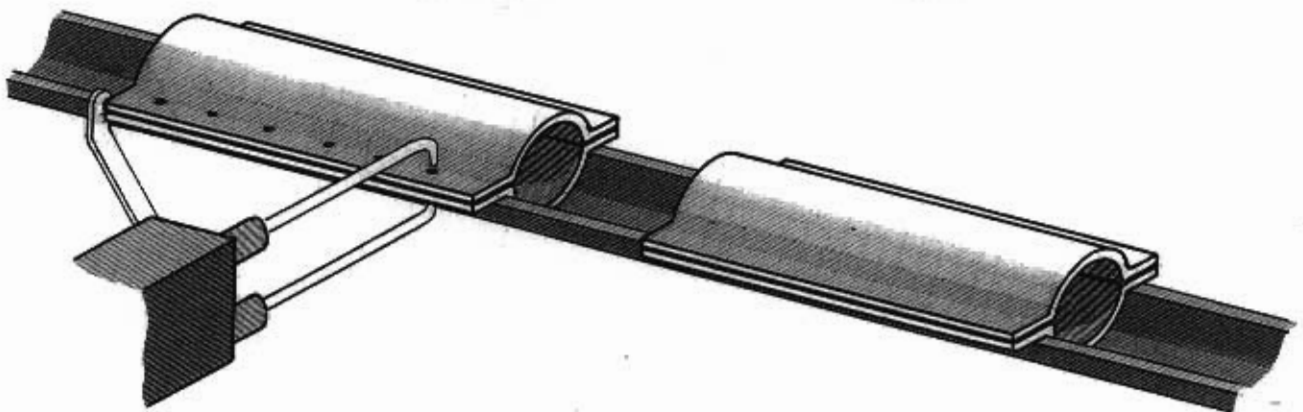


Figure 4.7 Welding Example.

In the welding example, relative mode stepping is used to move the welder from one spot to the next, while absolute mode stepping is used to bring the welder back to the home position, ready to start the next workpiece. The Wait line is used to indicate that a workpiece is in position, and the Dohile line indicates when there are no more pieces to weld. A real application may be more complex than what is illustrated, but the program indicates the level of problems which the CY512 can solve without help from a host computer. The program uses 19 of the 48 bytes available in the CY512 program buffer.

## OPERATIONAL MODE SUMMARY

TABLE III		OPERATIONAL MODE SUMMARY	
MODE DESCRIPTION	MODE 0*	MODE 1	MODE SELECTION VIA
DATA TYPE	ASCII DECIMAL	BINARY	(PIN 36 = HI/LO) (ASCII/BIN)
POSITION TYPE	RELATIVE**	ABSOLUTE**	'N' COMMAND SELECTS RELATIVE, 'P' COMMAND SELECTS ABSOLUTE
STEP MODE	FULL-STEP***	HALF-STEP	'H' COMMAND SELECTS HALFSTEP
GATED OPERATION	TRIGGERED	NON-TRIGGERED	PIN 30 LO IF NO TRIGGERING, STEP ON HI-TO-LO TRANSITION
EXECUTION	COMMAND	PROGRAM	'D' COMMAND SELECTS DO PROGRAM, '0' SELECTS COMMAND MODE

\*MODE 0 IS DEFAULT MODE IF DEFAULT EXISTS

\*\*ABSOLUTE MODE SET VIA EACH 'POSITION' COMMAND, ELSE RELATIVE MODE IN EFFECT.

\*\*\*RETURN TO DEFAULT MODE ONLY BY RESET (HARDWARE) OR 'INITIALIZE' COMMAND (SOFTWARE).



# 5 BINARY DATA MODE OF OPERATION 5

## BINARY DATA MODE

To facilitate microprocessor control using binary arithmetic, the CY512 can be placed in the BINARY data mode of command execution by applying a low voltage to pin 36. The possibility of the QUIT command occurring in the binary data necessitates the use of a data count sent after each command byte. In binary mode, the QUIT command, "Q" = 51H, may be inadvertently transmitted, since some of the binary Position or Rate data may assume this value. For this reason it is necessary to specify the number of binary data bytes to be sent to the CY512. In this mode, the data count and data values are specified in binary form, while the command letters retain their equivalent ASCII values.

Commands are issued by first sending the command letter, which has the same value as in ASCII mode. This is followed by a binary value data count. The data count represents the number of data bytes to follow the command byte. If the command is a single letter with no parameter, such as "A", "B", or "T", the data count will be zero, indicating the end of the command. This is similar to sending the command letter and a carriage return in ASCII mode. Note that the data counts are not ASCII characters, they are binary values. Commands with parameters in the range of 1 to 255 will have a data count of binary 1, since these values can all be specified in a single byte. Rate, Slope, etc. listed in Table IV are in this category. The data count is then followed by the single byte which is the binary value desired for that parameter. Commands such as Loop, Number, etc., listed in the table, will have a data count of binary 2, since their parameters cannot be specified in a single byte. The data count is then followed by the two bytes which represent the 16-bit value for the parameter, or the two parameters used by the Loop command. Note that 16-bit values are sent least significant byte first, while the Loop command parameters are sent as count then address, the same order as specified in the ASCII mode. All commands except QUIT are of the form:

COMMAND BYTE	COUNT	DATA BYTE 1	DATA BYTE 2
A,B,C,D,E G,H,I,T,U W,+,-,Ø	0	....	....
F,J,O,R S,V	1	Factor,Rate,etc.	....
L,N,P,X	2	Number of Steps,Target Position,etc. Least significant byte first	

Note that the QUIT command is not followed by a data count in the Binary mode, just as it is not followed by a carriage return in the ASCII mode. Also, it is possible to load an entire program with a single byte count value. To do this, issue the ENTER command with a data count value of zero, followed by the first command character of the program. Instead of following this character by the normal data count, use a count equal to the remaining total characters of the program, up to, and including the "Ø" command. Do not include the QUIT command in the count. The "Q" command should then be issued separately, ending the program entry mode and reverting to command mode. When this method of program loading is used, the "Ø" command must have a binary value of zero, not the ASCII character "Ø". The program may also be loaded as separate commands, with a normal data count for each command. The following example illustrates both options for loading a program in Binary mode:

ASCII Command	Binary with Separate Data Counts	Binary with Single Data Count	CY512 Buffer Contents
E}	{ 45.....	45	
	{ 00.....	00	
R 150}	{ 52.....	52.....	52
	{ 01.....	0C	
	{ 96.....	96.....	96
	{ 4E.....	4E.....	4E
N 300}	{ 02		
	{ 2C.....	2C.....	2C
	{ 01.....	01.....	01
+}	{ 2B.....	2B.....	2B
	{ 00		
G}	{ 47.....	47.....	47
	{ 00		
	{ 4E.....	4E.....	4E
N 750}	{ 02		
	{ EE.....	EE.....	EE
	{ 02.....	02.....	02
-}	{ 2D.....	2D.....	2D
	{ 00		
G}	{ 47.....	47.....	47
	{ 00		
Ø}	{ 30.....	00.....	00
	{ 00		
Q	{ 51.....	51	

The program buffer in the CY512 will contain the same 13 bytes no matter which byte sequence is used. In ASCII mode, it takes 31 characters to define the program, including the "E" and "Q" commands. In Binary mode, with a separate data count for each command, the program may be defined in 24 bytes. By using a single data count for the program, this number may be further reduced to 17 bytes. Note that the Binary mode values and the program buffer contents are shown as hex numbers.

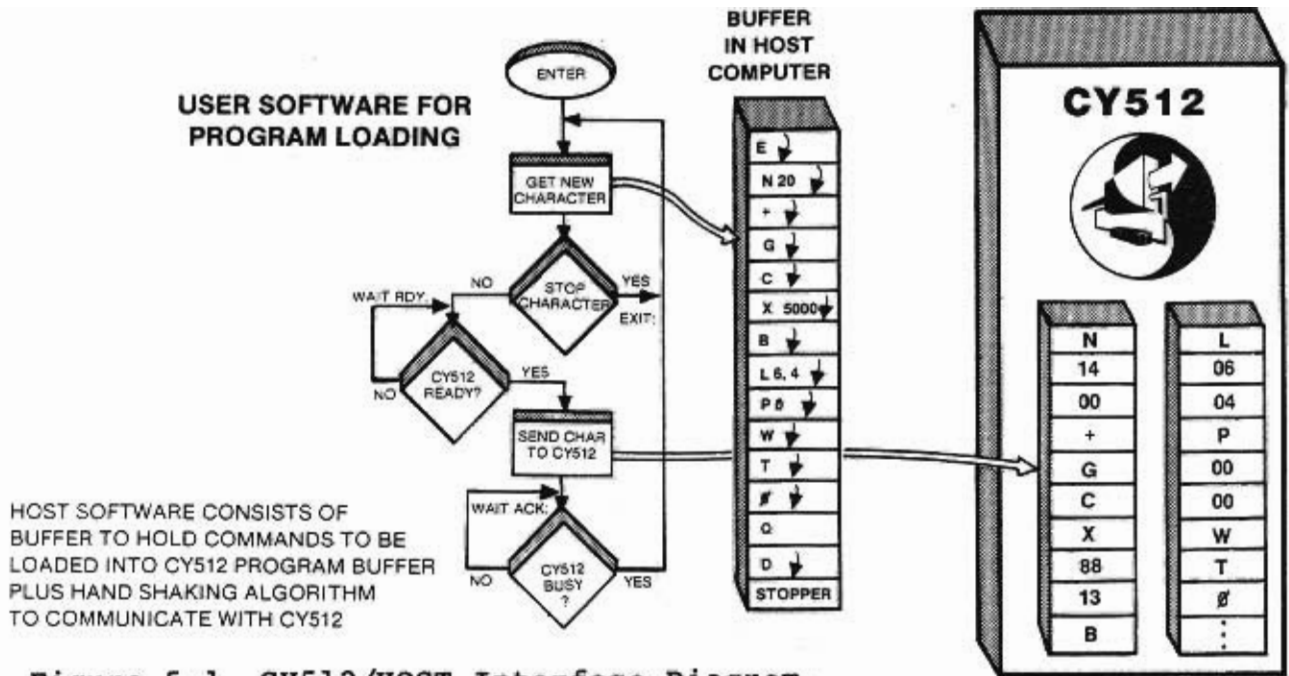
## INTERNAL PROGRAM STORAGE

The CY512 program buffer can contain 48 bytes of program commands and data. The Description of Commands contains the length of each command and is summarized in Table V. Note that program parameters set in the command mode do not require any space in the program buffer. If the internal storage is exceeded, the effects on operation will be unpredictable. For optimal operation, the CY512 is treated as a co-processor, with "subroutines" loaded and executed using Interrupt Req #2 (pin 32) to inform the host when a given routine has finished executing.

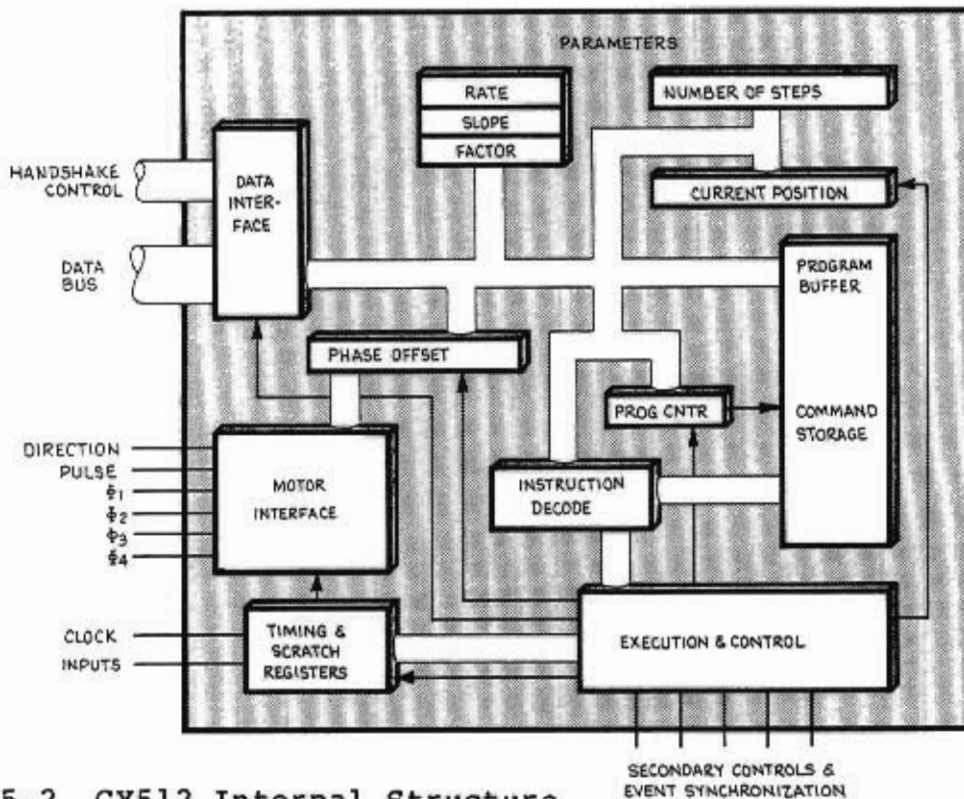
TABLE V INSTRUCTION LENGTHS AND PARAMETER CHARACTERISTICS				
COMMAND	BYTES	PARAMETER	RANGE	*TIME $\mu$ sec
ATHOME	1			375
BITSET	1			375
CLEARBIT	1			375
DOITNOW	1			380
ENTER	1			400
FACTOR	2	rate modifier	1-255	240+c+v
GO	1			660+v
HALFSTEP	1			375
INITIALIZE	1			375
JUMP	2	byte address	0-47	100+c
LOOP	3	count and address	1-255, 0-47	150+c
NUMBER	3	travel distance	1-65,535	180+c
OFFSET	2	drive signal output	0-7	80+c
POSITION	3	target location	1-65,535	550+c+v
QUIT	*			175
RATE	2	rate parameter	1-255	80+c
SLOPE	2	acceleration	1-255	80+c
TIL	1			380
UNTIL	1			380+v
VERIFY	2	buffer pointer	0-3	125+c
WAIT	1			380+v
EXPEND	3	msec time delay	1-65,535	100+c+v
+	1			380
-	1			380
Ø	1			600

**\*NOTES:**

Command execution times are for command mode. Program mode times are much shorter.  
 Maximum I/O Request time is 660  $\mu$ sec for "P" & "G" commands.  
 The "L" & "T" commands are normally used only in a program.  
 c = ASCII to Binary parameter conversion time.  
 v = variable time depending on parameter values.



The following illustration shows the internal structure of the CY512, including the data paths between the various parts of the device. Note that all parameters are stored in registers which are separate from the program and command buffer, so parameters which do not change may be defined in the Command mode, and require no space in the program buffer.



## INTERFACE EXAMPLE

In the following, it will be assumed that the 8080/8085 transmits data to the CY512 via output port 0DCH. The string of ASCII commands is stored at BUFFER and terminated by a terminal symbol 0FFH. The D-E register pair will be used to access the character string.

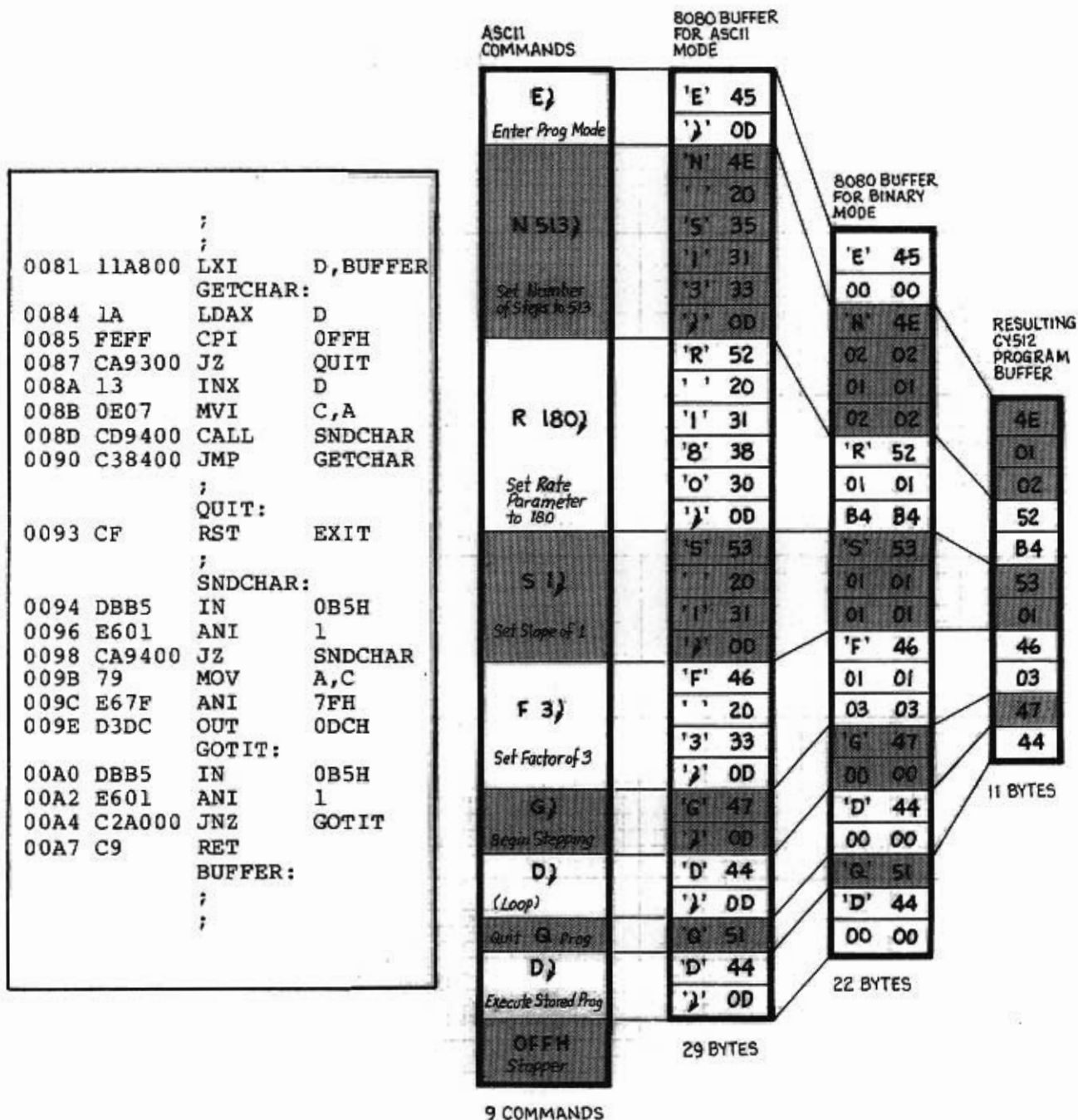


Figure 5.3 8080/8085 Interfaced to Stepper Motor through CY512 Stepper Motor Controller.

**VERIFY MODE OPERATION**

The Verify mode of operation allows the user to examine the internal register contents of the CY512. This is useful in determining the current state of the CY512, and in verifying parameters before or after critical operations, especially if communications between the CY512 and the host system are enacted in an electrically noisy environment.

The internal contents are divided into four groups, as specified by the parameter in the Verify command. Before reading the contents of the CY512, the user must issue a Verify command to set the internal pointer to the desired group. The contents are then read back one byte at a time, using the sequence described in CY512 Timing and Control Information. As each byte is read out, the internal pointer is advanced to the next byte value, allowing the specified group to be read back by repeated single-byte transfers. Note that I/O SELECT, pin # 39, should be high while the group is being read.

**TABLE VI VERIFY GROUP SUMMARY**

PARAMETER	GROUP	# of BYTES	DESCRIPTION
0	POSITION	2 or 5	current position Binary or ASCII
1	PROGRAM	0 to 48	program buffer contents
2	STATUS	6	pointers and internal flags
3	PARAMETER	5	N, S, R, F parameter values

v 0)

As indicated in Table VI, issuing the Verify command with a parameter value of 0 will set the internal pointer to the current position. The position can then be read back as either a two-byte quantity in Binary mode, or a five-byte quantity in ASCII mode. The mode in which the Verify command is issued will determine the format of the position output. The binary quantity will be presented most significant byte first, and the ASCII quantity will be presented most significant digit first. Internally the position is always maintained in binary, so in ASCII mode, it is converted to the ASCII-decimal equivalent before being output by the CY512. The ASCII position is always a five digit integer quantity, with leading zeroes as required. Note that position is the only quantity converted to ASCII decimal when in ASCII mode. All other Verify outputs are presented in binary, independent of the current command mode. If the current position value is 750, the five ASCII characters would be "00750" in left to right order. The binary mode equivalent would be 02EE, sent as two bytes, first the 02, then EE.

### V 1)

With a Verify parameter of 1, the CY512 will output the contents of the program buffer. The maximum program size is 48 bytes, representing the longest program which may be read back. The actual number of bytes which have any meaning will depend on the length of the current program. Output starts with location 0 of the buffer, the front of the program. Commands are stored in the program buffer with the format indicated in Description of Commands except that the 0 command has a binary value of zero. Note that two-byte parameters are stored least significant byte first in the program buffer. Several examples in this manual illustrate the program buffer contents. All would be read back in the order shown, from the front of the program through the end. If additional bytes are read back, they may not have any meaning, since most programs will not use the entire buffer.

### V 2)

The Status group, accessed with a verify parameter of 2, consists mostly of pointers and flags used by the internal operations of the CY512. This group is provided mainly for device testing, and is not expected to be of general interest, except for two flags, which both occur in the sixth byte of the group. If the fifth bit (DB<sub>4</sub>) of the sixth byte is high, the current direction is set CCW. Direction is CW if the bit is low. Note that the sense of this bit is opposite that of the DIRECTION line, pin # 33. If the eighth bit (DB<sub>7</sub>) of the same byte is high, it indicates that half step mode is enabled. A low value indicates full step mode.

### V 3)

The final group accessed by the Verify command is the parameter group, pointed to when the Verify argument is 3. This is a five-byte group, consisting of the parameters which may be specified by the user. The first parameter output is the number of steps, as specified by the N command. This is a two-byte value, with the most significant byte output first. Next is the slope parameter, as specified by the S command. This is followed by the current rate, set by the R command. Finally, the factor, as specified by the F command, is output. Slope, rate, and factor are all single-byte values. With N 513), S 25), R 180), and F 3), the bytes read back would be 02, 01, 19, B4, and 03 (HEX), for N, S, R, and F respectively.

The timing sequence of Figure 7.1, in CY512 Timing and Control Information, was generated by sending the command "V 1)", and then reading the first four bytes of the program buffer. The other groups may be accessed in an identical manner, by substituting the desired group number in place of the 1 in the V command. An example subroutine for reading back a desired number of bytes is shown in Figure 6.1. The routine is written in 8080

Assembly Language. It assumes that the V command has already been sent. A routine such as the SENDPARALLEL subroutine, shown in Figure 11.6 could be used to send the desired V command. The RCVBYTE routine is entered with the B register set to the number of bytes to read, and the DE register pair pointing to a RAM buffer which will hold the data.

```

;
RCVBYTE: ;READ CY512 IN VERIFY MODE, V CMD ALREADY SENT
;B = # OF BYTES TO READ, DE = BUFFER POINTER
003B DBED IN STATUS
003D E620 ANI READY ;LOW IF BUSY
003F CA3B00 JZ RCVBYTE ;WAIT FOR READY
;
0042 3E03 MVI A,IOSELOUT
0044 D3EF OUT A4CNTRL ;I/O SELECT SET HIGH FOR READ BACK
;
NEXTCHAR:
0046 3E00 MVI A,IOREQ
0048 D3EF OUT A4CNTRL ;I/O REQUEST LOW TO REQUEST A BYTE
;
WAITDATA:
004A DBED IN STATUS
004C E620 ANI READY ;LOW WHEN BUSY
004E C24A00 JNZ WAITDATA ;BUSY MEANS CY512 HAS OUTPUT A BYTE
;
0051 DBEC IN DATA ;READ THE BYTE
0053 12 STAX D ;SAVE IN BUFFER
0054 13 INX D ;POINT TO NEXT BUFFER LOCATION
;
0055 3E01 MVI A,NOIOREQ
0057 D3EF OUT A4CNTRL ;I/O REQUEST HIGH TO ACK BYTE RECEIVED
;
WAITCLR:
0059 DBED IN STATUS
005B E620 ANI READY
005D CA5900 JZ WAITCLR ;WAIT FOR READY AGAIN
;
0060 05 DCR B ;CHAR COUNT
0061 C24600 JNZ NEXTCHAR ;MORE BYTES TO READ IF NOT ZERO
;
0064 3E02 MVI A,IOSELIN
0066 D3EF OUT A4CNTRL ;I/O SELECT SET LOW FOR NEXT COMMAND
0068 C9 RET
;
;

```

Figure 6.1 Verify mode read subroutine.



# 7 TIMING AND CONTROL INFORMATION 7

## CY512 HANDSHAKE TIMING INFORMATION

With the parallel interface to the CY512, the user must wait for the CY512 BUSY/RDY line (pin #27) to be high before applying I/O REQUEST strobe to pin #1. Note that no data set-up time is required, so the data may appear on the data bus at the same time that the I/O REQUEST write strobe goes low. This is especially convenient in the ASCII mode as bit 7 of the ASCII data byte can be used to generate the write strobe (figure 10.8). The data is read into the CY512 from the data bus by a low going read strobe, INSTROBE, appearing on pin #8. The data should be valid at the trailing edge of INSTROBE. INSTROBE may be used to enable the data onto the data bus from an external device. The data may be removed at any time following the occurrence of INSTROBE, however the I/O REQUEST line should be held low until BUSY/RDY acknowledges the transfer by going low. The simplest interface ignores INSTROBE and uses BUSY/RDY only (Figure 2.1). I/O SELECT must be low while commands are being sent to the CY512, and the maximum active time for I/O REQUEST is 660 usec for P and G commands.

Timing for the Verify mode, in which the internal contents of the CY512 may be examined, is similar to that described above. In order to read the internal contents, the I/O SELECT line must be high. This will put the CY512 in an output mode. When the BUSY/RDY line is high (ready), the user should strobe I/O REQUEST (pin # 1) low. The CY512 will then write the next byte value onto the data bus. This is indicated by a low going write strobe, OUTSTROBE, appearing on pin #10. The data will be latched and valid on the trailing edge of OUTSTROBE, which may be used to latch the byte into the user's input port. After OUTSTROBE, the CY512 will go busy, indicating that the data is available on the bus. Data will remain valid until I/O REQUEST is again set high by the user. Note that the user may read the data directly, after the CY512 goes busy, before raising I/O REQUEST. When the CY512 detects I/O REQUEST high, the data bus will be put back into a high impedance state, and data will no longer be valid. This operation will be indicated by a second OUTSTROBE pulse. The CY512 will then indicate ready again, awaiting the next command or another verify read, as indicated by the I/O SELECT line. See Figure 7.1 for the waveforms.

To enable all CY512 features, the user should connect all 8 lines of the data bus to his I/O ports, and generate I/O REQUEST and I/O SELECT as separate lines. I/O SELECT should be changed only while the CY512 is ready (BUSY/RDY is high), and may be used to determine the direction of data on the data bus (low=into CY512, high=out from CY512). Since the data bus is bidirectional, the user must turn off (tri-state) the command output port during the Verify mode, allowing the CY512 to drive the data bus lines. This may be done by I/O SELECT, or the command port could be tri-state at all times except during the INSTROBE pulse.

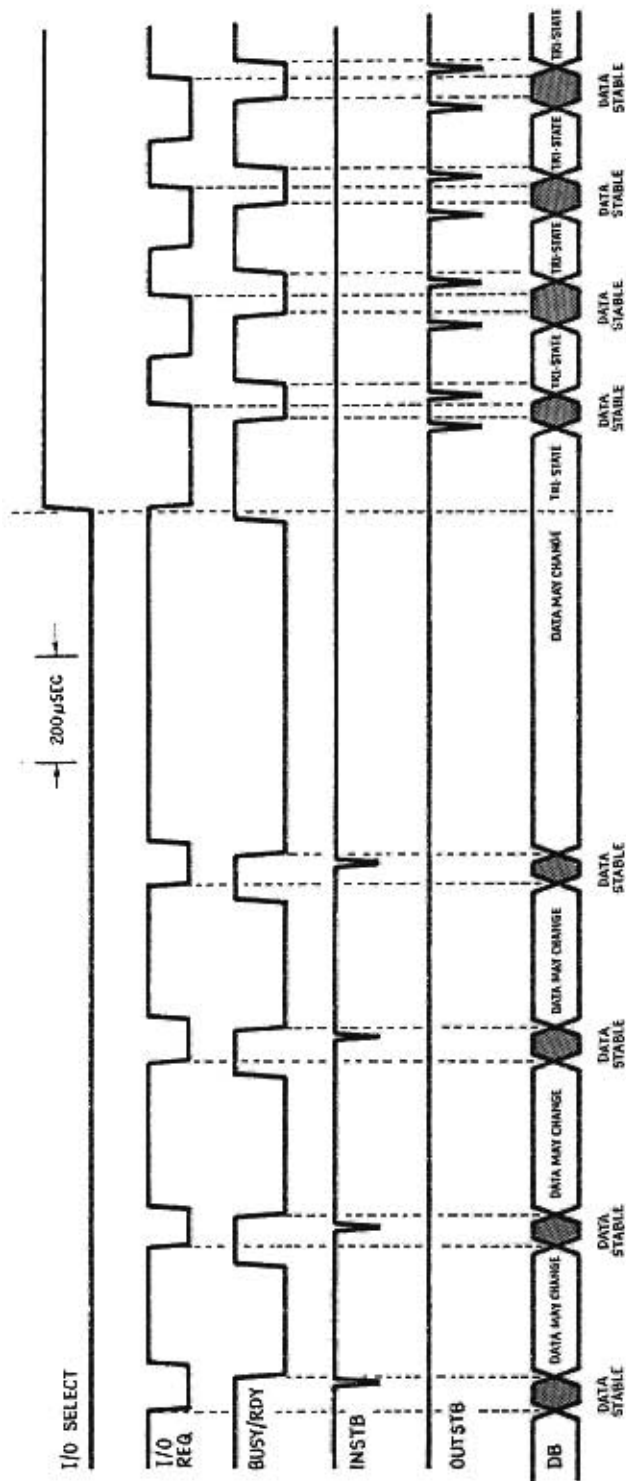


Figure 7.1 Parallel Handshake Timing Sequence.

## DOITNOW "RUN" TIMING

After entering program code into the CY512 program buffer, and exiting the program entry mode via the "Q" command, the host computer should wait for the PROG line (pin 31) to go high, indicating that the CY512 is no longer in the program entry mode. After testing the RDY line to be sure that it is high, the host computer can send the "DOITNOW" command (D), as shown in Figure 7.2. Using a 6MHz crystal, the CY512 will be busy approximately 150 microseconds with the "D" command and approximately 400 microseconds after detecting the end-of-command code ( $\mu=0DH$ ). The CY512 will then lower the RUN line (pin 32), raise the RDY line (pin 27) and begin executing the program. If the first program instruction is BITSET (and pin 34 has been previously cleared) the Programmable Output line (pin 34) will go high in less than 100 microseconds from the rising edge of BUSY/RDY.

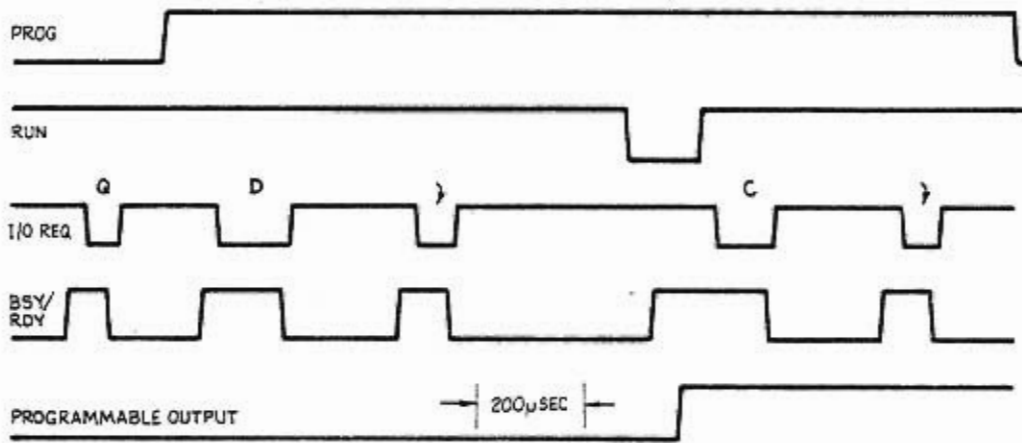


Figure 7.2 DOITNOW (Run) Timing Diagram.

### STEP INHIBIT PIN

In the triggered mode of operation, the GO command initiates the stepping sequence if the Step Inhibit pin is low. If the Step Inhibit pin (pin #30) is high, the controller simply waits for a low level on this pin and then takes the step. The level of the pin is tested just before every step is taken.

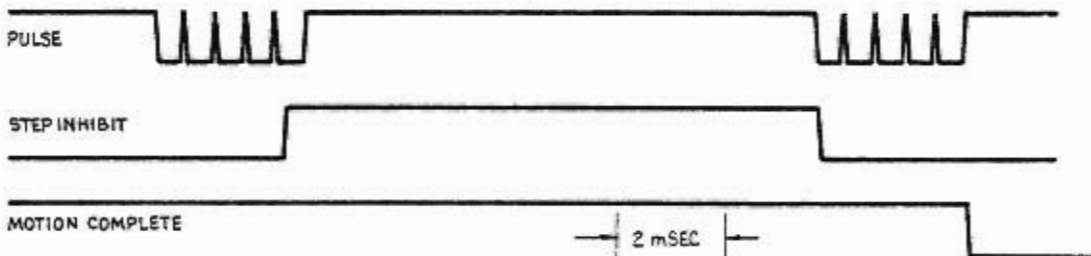


Figure 7.3 Step Inhibit Timing.

### DIRECTION CONTROL

In the Absolute Mode of operation, in which target positions are specified with the "P" command, direction is determined automatically. If the specified position is greater than or equal to the current position, the direction is set to clockwise (CW), and if the specified position is less than the current position, the direction is set to counter-clockwise (CCW). The current setting of direction is indicated by the DIRECTION pin (pin # 33). HIGH corresponds to CW, and LOW corresponds to CCW. In the Absolute Mode, the DIRECTION pin is set just before stepping begins.

In the Relative Mode of operation, in which the number of steps to take from the current position is specified with the "N" command, and stepping is activated with the "G" command, the user may control direction with the "+" (CW) and "-" (CCW) commands. The current setting of direction is still indicated by the DIRECTION pin, and stepping will occur in the direction last set when the "G" command is issued. The system powers up in the clockwise direction. Note that the direction commands are separate commands; they are terminated by the carriage return character,  $\text{)}=0DH$ . Thus, to specify 100 steps in the counter clockwise direction it is necessary to send two commands:

```
-)
N 100)
```

instead of sending:

```
N -100)
```

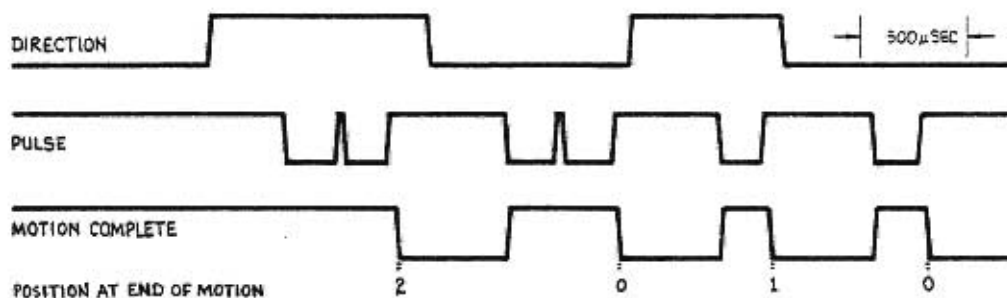


Figure 7.4 Direction indication. Program: P 2) P 0) +) G) -) G)

## STEP TIMING SIGNALS

The PULSE output (pin #35) may be used as a step timing signal. When the CY512 is not stepping, this output is high. It goes low at the beginning of every step, and stays low for the duration of the step. When the step time is over, PULSE goes high again. PULSE remains high for about 40 microseconds between steps. This time has been included in the rate equation.

The PULSE line also interacts with the TERMINATE/ABORT line (pin #6). If TERMINATE/ABORT goes low while pulse is low, it is considered a terminate function, in which the current step is ended before the nominal step time expires. PULSE will go high to indicate that the step is over. The rising edge of PULSE should be used to clear the TERMINATE input. If TERMINATE/ABORT is low while pulse is high (between steps), it is considered an abort function, in which the down ramp mode is immediately enabled. The step rate will decrease to the minimum value for the current settings of rate, slope, and factor. Then stepping will continue to the target position at this slower rate. Note that the ABORT input should be gated on by a high level on PULSE,

so that it will not be confused with the TERMINATE input. A suggested circuit is shown in Figure 7.6. The TERMINATE input is sampled once every 15 usec while PULSE is low, and the ABORT input is sampled once, about 7.5 usec after PULSE goes high.

By combining the ABORT, STEP INHIBIT, and Stop Operation functions of the CY512, it is possible to bring a motor to a safe halt, under emergency conditions, from a high slew rate. ABORT is used to enable the down ramp mode, bringing the motor to a slower step rate from the high slew rate. STEP INHIBIT may be used to stop the motor once the rate is sufficiently slow to accomplish this without losing steps due to system inertia. Then the Stop Operation function, using the QUIT command, could be used to end the current step command before the target position is attained.

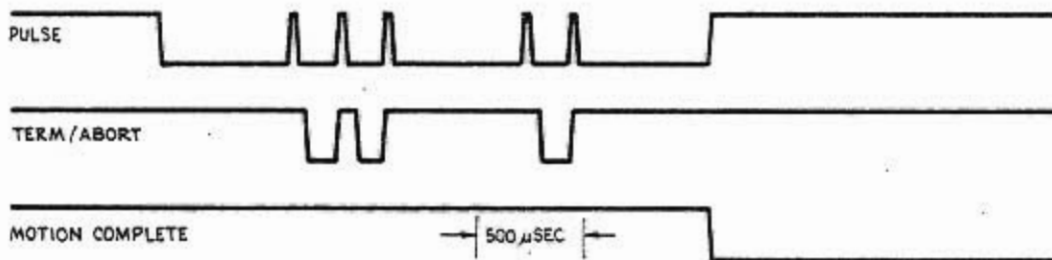


Figure 7.5 Step Timing Signals for terminate.

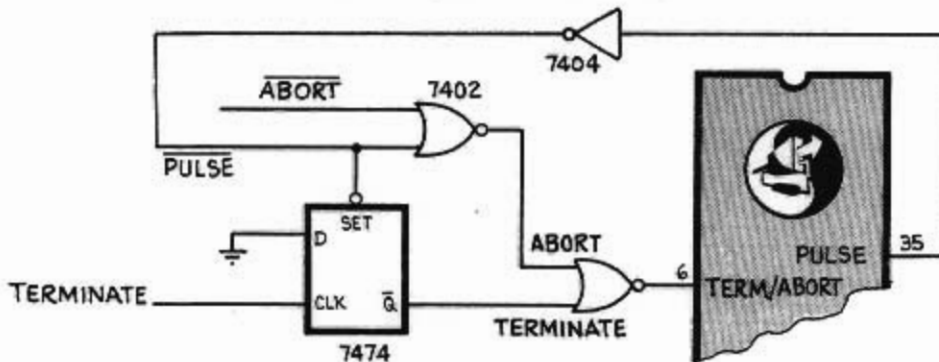


Figure 7.6 ABORT/TERMINATE separation logic.

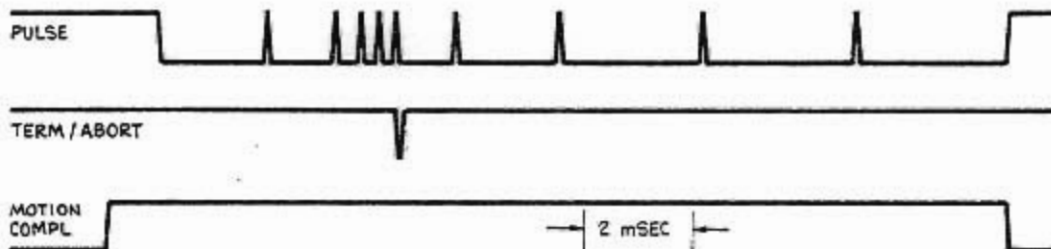


Figure 7.7 ABORT sequence.

## STOP OPERATION

If it becomes necessary to halt a step sequence immediately, the STEP INHIBIT line may be used to halt stepping, as explained before. As long as the line is HIGH, the CY512 will not step. However, the CY512 will continue to monitor the line, and will continue the step sequence when the line is LOW again. In some cases it is desirable to stop the motion and quit the current step command. This capability is provided in the following way. To stop the motion, use the STEP INHIBIT line. Then, to quit the command, issue a "Q" command to the CY512. When STEP INHIBIT is activated, a delay equal to the step time must be executed before the "Q" command is issued. The necessary delay may be computed from the current step rate, or the rising edge of the PULSE line may be used to indicate end of step. Then the "Q" command should be issued by placing the ASCII "Q" on the Data Bus lines and pulsing I/O REQUEST low for 150 microseconds. This will insure that the CY512 will see the "Q" command. The current step command will then stop, and the CY512 will go back into Command Mode, waiting to execute the next command entered. If the CY512 was running a program when the "Q" command was issued, it will also quit the Run Mode and go back into Command Mode. However, the RUN line will still be LOW, so a "Ø" command (followed by a carriage return) should also be issued in this case. This feature allows the user to quit any stepping operation, whether in the Command Mode, or the Run Mode, regain control over the CY512, and issue new commands. Note that this operation occurs while the CY512 is in the "busy" state, so the normal BUSY/READY handshake is not used. See the listing of an example Quit routine (figure 7.8) and the timing diagrams for further details.

The Emergency Stop Operation should be used to halt the stepping in exceptional cases only. It is not designed as a normal means of halting the CY512. After using the Stop Operation, it is best to redefine the program, using the "E" and "Q" commands. Some internal pointers, which are incremented by the Stop Operation, are not reset until an "E" command is executed. If too many such operations are done before another "E" command, operating parameters and modes will be altered during normal command entry, causing faulty operation of the CY512.

## SOFTWARE ABORT

```

;
; STOPOP: ;SPECIAL ROUTINE TO IMPLEMENT STOP OPERATION
;HALTS STEPPING AND QUILTS CURRENT COMMAND
0000 3E0B MVI A,STPINH
0002 D3EF OUT A4CNTRL ;STEP INHIBIT=HIGH TO STOP STEPPING
;
; WAITPLS:
0004 DBED IN STATUS
0006 E602 ANI PULSE ;LOW DURING STEP
0008 CA0400 JZ WAITPLS ;WAIT FOR END OF CURRENT STEP
;
000B 3E51 MVI A,'Q'
000D CD2600 CALL CMDABCHAR ;ISSUE SPECIAL Q CMD, NO HANDSHAKE
;
0010 DBED IN STATUS
0012 E604 ANI RUNBAR ;LOW IF RUNNING A PROGRAM
0014 C22100 JNZ ENDOP ;SKIP IF NOT RUN MODE
;
0017 3E30 MVI A,'0'
0019 CD2600 CALL CMDABCHAR ;ELSE ISSUE SPECIAL 0 COMMAND
001C 3E0D MVI A,CR ;TO INDICATE BACK IN COMMAND MODE
001E CD2600 CALL CMDABCHAR ;ALSO NEED CARRIAGE RETURN (CR)
;
; ENDOP:
0021 3E0A MVI A,TRIGGER
0023 D3EF OUT A4CNTRL ;RELEASE STEP INHIBIT, LOW TO STEP
0025 C9 RET
;
;
; CMDABCHAR: ;SPECIAL SEND CHAR DURING STOP OPERATION
0026 D3EC OUT DATA ;OUTPUT CHAR FROM A TO DATA BUS
0028 3E00 MVI A,IOREQ
002A D3EF OUT A4CNTRL ;LOWER I/O REQUEST FOR VALID DATA
002C CDEB03 CALL T150U ;DELAY 150 USEC FOR CY512 TO DETECT
002F 3E01 MVI A,NOIOREQ
0031 D3EF OUT A4CNTRL ;RELEASE I/O REQUEST--NO HANDSHAKE
;
; WAITRDY:
0033 DBED IN STATUS ;CY512 WILL SHOW READY WHEN STOP
0035 E620 ANI READY ;OPERATION IS COMPLETE
0037 CA3300 JZ WAITRDY
003A C9 RET

```

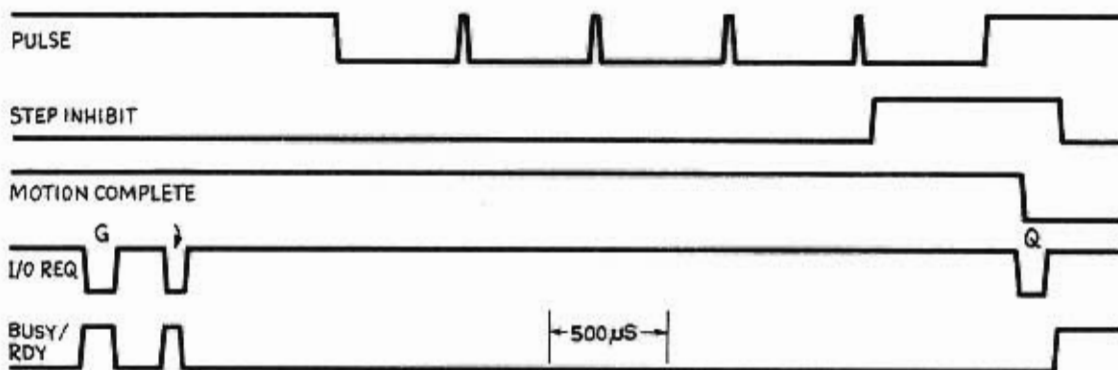


Figure 7.8 Software Abort timing

## HALF STEP OR FULL STEP

Steppers operate either in full step mode, with each step equal in torque output (proportional to current input), or in half-step mode in which the drive current alternates between 1 and 2 (n and 2n) coils. The half-step mode doubles the number of steps per revolution, thereby doubling the resolution. Although the torque is not maximum in the half-step mode, but varies, the variation tends to broaden the non-resonant bands; i.e., to diminish the region in which resonance occurs.

The tables and waveforms below indicate the sequential values assumed by the stepper drive signals. The tables also indicate how the step patterns correspond to the Offset parameter specified in an OFFSET (O) command. Note that the OFFSET command will set the internal step pattern pointer to that specified by the parameter indicated. The actual next step pattern, which will appear on the drive signal lines, depends on the direction in which the CY512 is stepping. When going CW, the next higher value will appear, and when going CCW, the next lower value will be used. Thus, if the command "O 2)" is issued, the internal step pattern pointer will be set to step number 3. If the CY512 takes its next step in the CCW direction, the next step pattern to appear on the drive signal pins will be that for step number 2. The OFFSET command does not change the current value output

on the drive signal pins, only the internal pointer setting. This change will not be noticed until the CY512 takes the first step of the next motion command. The OFFSET command is useful for switching back and forth between full-step and half-step modes without gaps in the stepping patterns.

FULL STEP DRIVER SIGNALS					
STEP	1	2	3	4	1
$\phi_1$	0	1	1	0	0
$\phi_2$	1	0	0	1	1
$\phi_3$	0	0	1	1	0
$\phi_4$	1	1	0	0	1
OFFSET	0	1	2	3	0

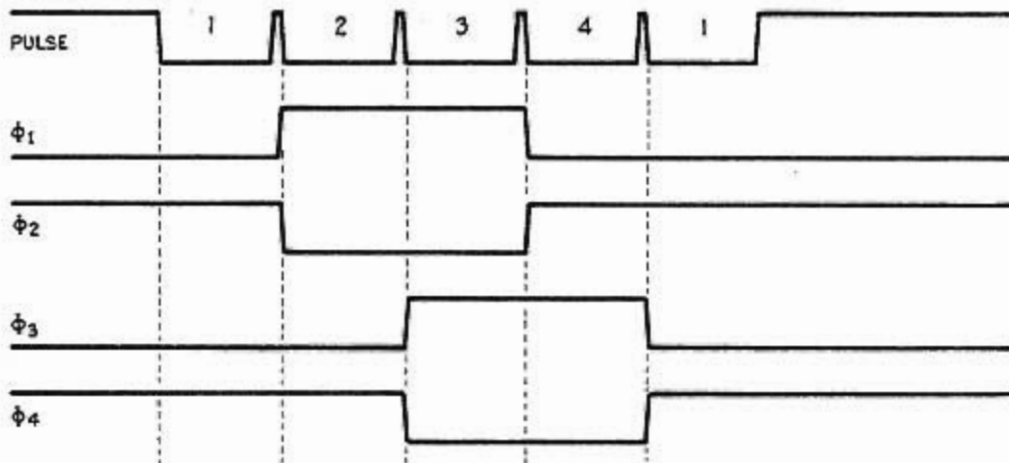


Figure 7.9 Full Step Control Outputs.



HALF STEP DRIVER SIGNALS									
STEP	1	2	3	4	5	6	7	8	1
$\Phi_1$	0	1	1	1	1	1	0	0	0
$\Phi_2$	1	1	0	0	0	1	1	1	1
$\Phi_3$	0	0	0	1	1	1	1	1	0
$\Phi_4$	1	1	1	1	0	0	0	1	1
OFFSET	0	1	2	3	4	5	6	7	0

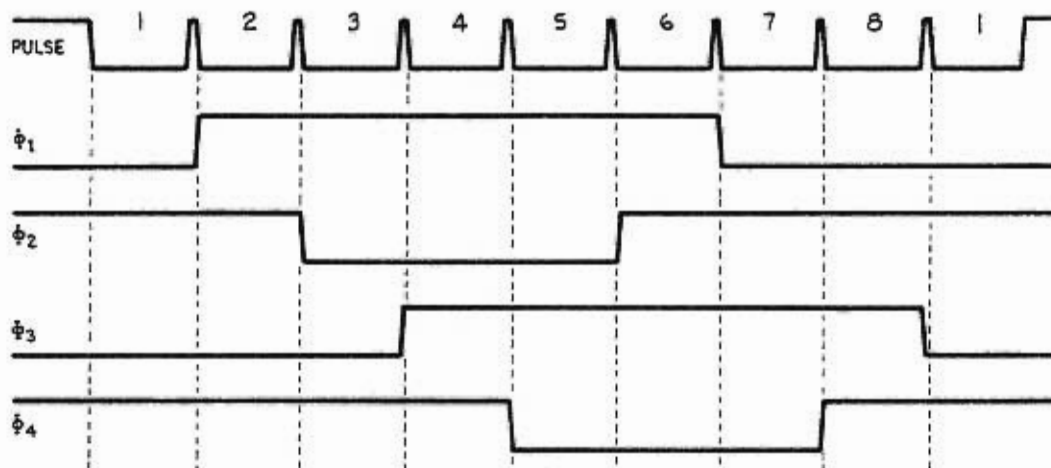


Figure 7.10. Half Step Control Outputs.

The stepping waveforms indicate that outputs  $\Phi_1$  and  $\Phi_2$  are paired together, as are outputs  $\Phi_3$  and  $\Phi_4$ .  $\Phi_1$  and  $\Phi_2$  should be connected to opposite ends of the same winding of a four phase motor, and  $\Phi_3$  and  $\Phi_4$  should be connected to the other winding of the motor. The user should study the waveforms and the motor requirements carefully to determine the proper connection between the phase outputs and the wires coming from the motor.

The phase outputs are considered on, or energized, when they are low (0) and off, or de-energized, when they are high (1).

### RATE CONTROL

Along with flexible and precise position control, the CY512 allows the user to define the step rate and acceleration rates in very fine increments (10 microsecond changes with a 6 MHz crystal). By proper choice of the rate, slope, and factor parameters, it is possible to control the starting rate, the number of steps from the starting rate to the slew rate, and the final slew rate. In addition, by controlling the STEP INHIBIT and TERMINATE lines, the user can achieve precise rate control via an external pulse source or feedback from the motor assembly.

When external controls are not used to define the step rate, the CY512 uses an internal timer algorithm to control stepping. The algorithm may be modeled by the following equation:

#### CY512 Rate Equation

$$\text{steps per second} = \frac{1}{(256-r)*80\mu\text{sec} + z*10\mu\text{sec} + 75\mu\text{sec} \pm 7.5\mu\text{sec}}$$

The equation and this discussion use several variables which are defined as follows:

- r = Rate parameter as set by the RATE command
- s = Slope parameter as set by the SLOPE command
- f = Factor parameter as set by the FACTOR command
- y = Integer ((255-f)/s)
- z = Internal factor value

The variable y represents the number of steps required to ramp up from the starting rate to the slew rate, and the number of steps to ramp down again before stopping.

The internal factor, z, changes in value for each step while ramping. When accelerating,  $z = 255 - n*s$ , where n is the step number from the start of motion, beginning with  $n = 1$ . When slewing,  $z = f$ , giving the maximum step rate for any particular settings of r and f. While decelerating,  $z = f + n*s$ , where n is the step number from the start of deceleration.

The above relationships indicate that the starting rate is given by  $1/((256-r)*80 + (255-s)*10 + 75 \pm 7.5)$ , and the slew rate is given by  $1/((256-r)*80 + f*10 + 75 \pm 7.5)$ . Note that the  $\mu\text{sec}$  times indicated in the equations are correct for a 6 MHz crystal, and must be linearly scaled for other crystal frequencies.

The rate equation applies as specified when  $1 \leq r \leq 254$ . In the case where  $r = 255$ , the rate will not follow the equation, but

will differ, depending on the direction of motion and full step or half step mode. The appropriate rate for each case is indicated below:

Step Rates for r = 255		
	CW	CCW
Full step	$1/(218\mu\text{sec} + z*10)$	$1/(232\mu\text{sec} + z*10)$
Half step	$1/(224\mu\text{sec} + z*10)$	$1/(230\mu\text{sec} + z*10)$

The " $\pm 7.5 \mu\text{sec}$ " term in the rate equations represents an uncertainty between the exact instant the step time expires and the point at which it is detected. The time out is tested once every 15 microseconds. For any given setting of the parameters, the step rate will be a constant within the  $\pm$  range specified. The CY512 will not exhibit a jitter from step to step.

Table VII is a listing of the range of possible slew rates for a given setting of r. The slowest step rate in each range uses the value f = 255, and the fastest rate is obtained when f = 1. Values are indicated for three different crystal frequencies; 2 MHz, the slowest rate, 6 MHz, the medium rate for which most timing in this manual is specified, and 11 MHz, for the fastest possible rates.

TABLE VII RATE TABLE						
Rate r	2 MHz		6 MHz		11 MHz	
	f=255	f=1	f=255	f=1	f=255	f=1
1	14	16	43	48	79	89
25	15	17	47	53	86	98
50	17	20	52	60	95	110
75	19	22	58	68	107	125
100	22	26	66	79	121	145
125	25	31	76	94	139	173
150	30	38	90	116	165	214
175	36	50	109	152	201	279
200	46	73	140	219	258	401
205	49	80	149	240	273	440
210	52	88	158	265	290	486
215	56	99	169	297	310	544
220	60	112	181	337	333	618
225	65	129	195	389	359	714
230	70	153	212	461	389	846
235	77	188	232	566	425	1038
240	85	244	256	732	469	1343
245	95	345	285	1036	523	1899
250	107	589	322	1769	590	3244
251	110	687	330	2061	606	3788
252	113	823	339	2469	622	4484
253	116	1025	349	3076	639	5494
254	119	1360	359	4081	658	7462

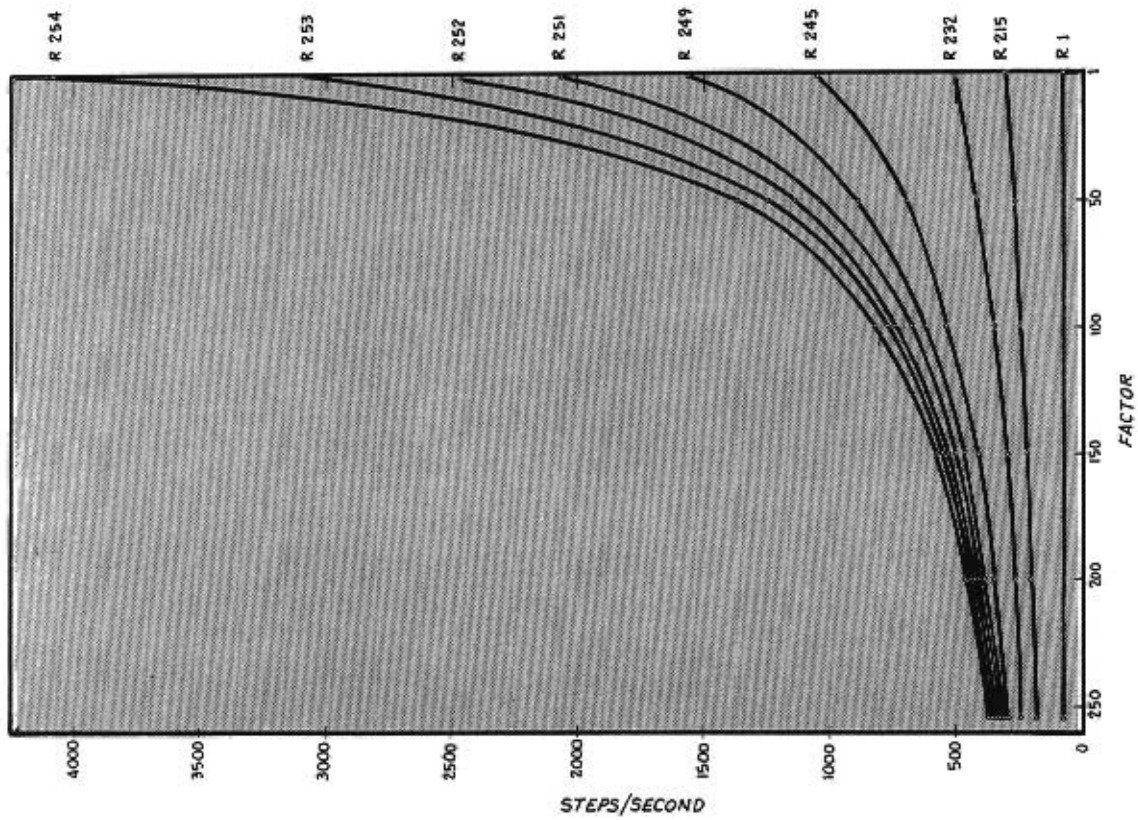
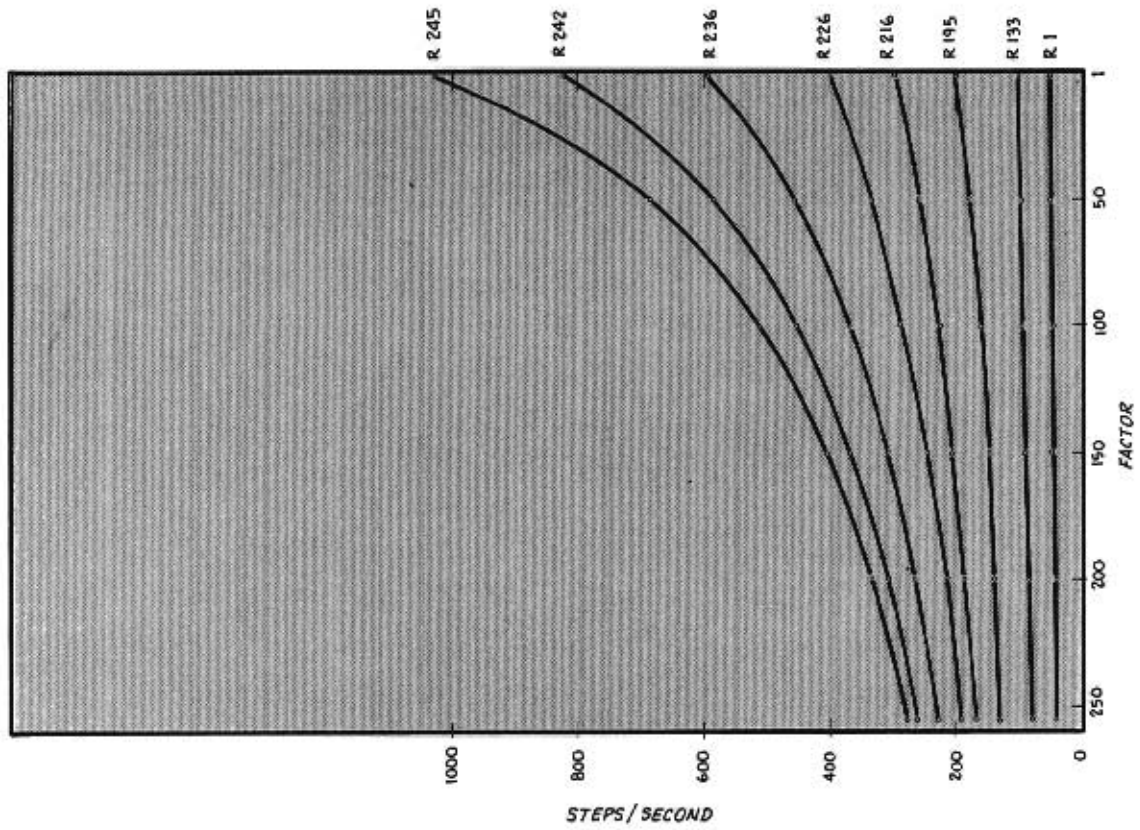


Figure 8.1 Step Rate vs. Rate Parameter with  $1 < f \leq 255$ .

## RAMPING MODE OF OPERATION

The maximum step rate is defined by a combination of the RATE and FACTOR parameters, as indicated in the step rate equation. When a motion begins, the step rate will not generally be at the maximum rate, but will ramp up to this slow rate from a slower value. The slope, or number of steps required to attain the slow rate, is defined by the SLOPE parameter. When stepping begins, an internal factor value (z) is set to 255. The value of the SLOPE parameter is then subtracted from this internal factor once per step, increasing the step rate as the factor decreases (see the rate equation). Note that the RATE parameter is not changed by this process, only the internal factor. When the internal factor becomes less than or equal to the value specified by the FACTOR command (f), the slow rate has been reached, and the internal factor is set so the specified slow rate is maintained. At this point the SLEW line (pin #29) is also brought low, indicating that the maximum specified step rate has been attained. The CY512 will continue slewing until it is time to down ramp to hit the target position. The point at which this occurs is determined automatically by the CY512. The SLEW line is then turned off (set high), and the value of the SLOPE parameter is added to the internal factor once per step, producing a slower step rate until the target position is reached, or the slowest rate is attained (internal factor=255).

If the number of steps to travel is less than the number required to reach the slow rate and then ramp down again, the CY512 will not ramp to the slow rate. It will just step to the target position at a rate equal to that given by the current RATE parameter and an internal factor of 255, the slowest step rate for the current RATE parameter. The CY512 does not partially ramp up, then ramp down if the target position is too close. The number of steps required to ramp up and down is given by  $2*y$ . The number of steps must exceed this value for the CY512 to work in the ramped mode.

Note that FACTOR must be the last of the SLOPE and FACTOR commands specified, since it performs a calculation based on the other values. If the slope is changed, the FACTOR command must be repeated after the new slope, to insure proper ramping operation.

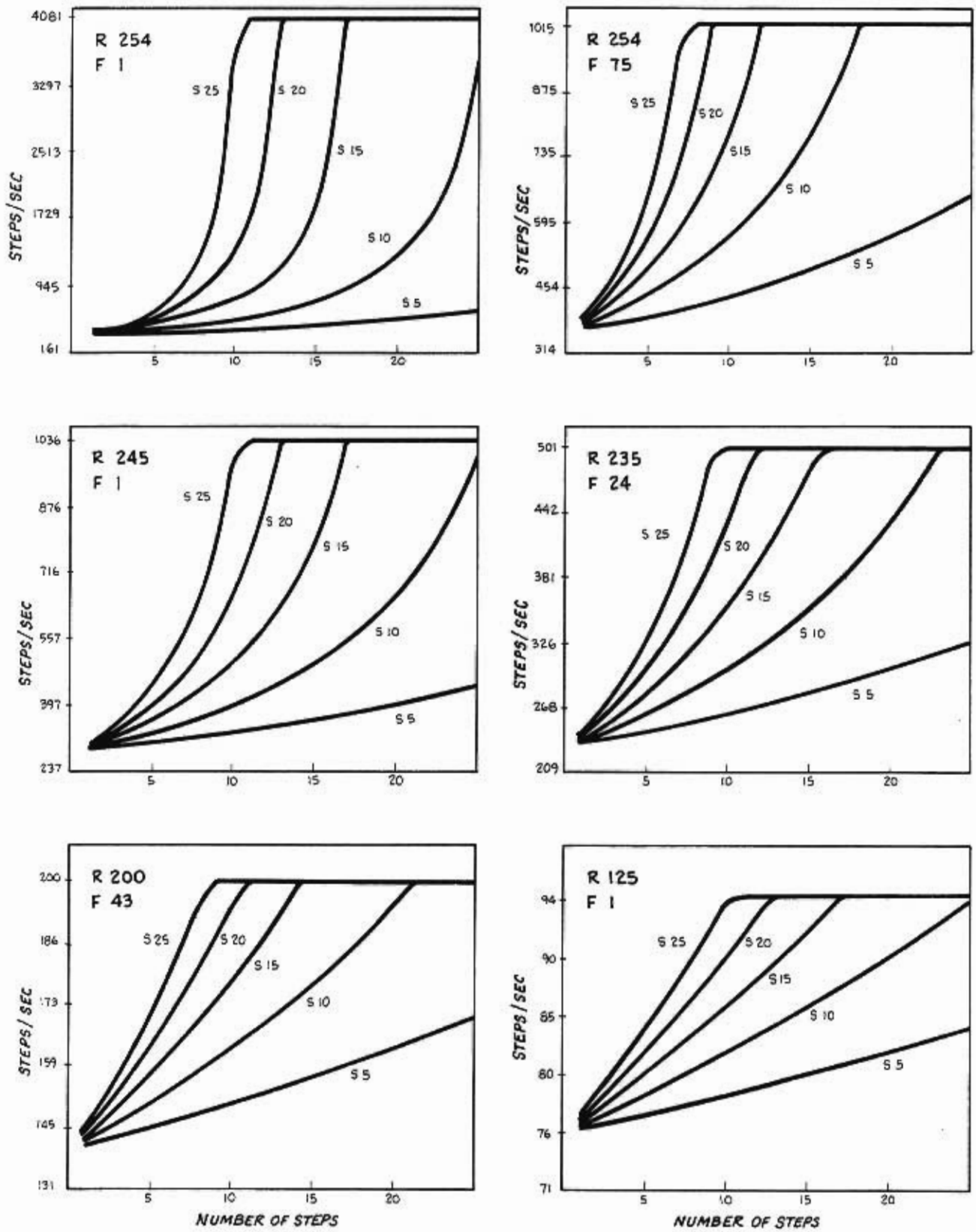


Figure 8.2 Variation of Rate Parameter with Position for five different slopes.

## DISPLAY OF RAMPED OPERATION

The display of several output lines is easily accomplished using the end-of motion signal (INTREQ1 = pin 37) to externally trigger the horizontal sweep circuits of an oscilloscope. The host computer sends the setup parameters to select the maximum rate and the number of steps to be taken and then sends the GO command. The following timing diagrams illustrate the effect of various slope parameters on the stepping behavior.

In the first example, figure 8.3, the parameters are chosen to give a five step acceleration before slewing is indicated. With a number of steps equal to 16, the CY512 will accelerate through five steps, slew for six steps, then decelerate through five steps. In the next example, figure 8.4, the number of steps is changed to six, while the other parameters are unchanged. Since five steps are required for ramping up, and another five steps for ramping down, the number of steps must be greater than 10 if the CY512 is to step in the ramped mode. Remember  $y = \text{Integer}((255-f)/s) = 5$ , and the number of steps must exceed  $2*y = 10$  for ramping to occur. Since six is not greater than ten, the CY512 takes the six steps in a nonramped mode, stepping at a slow, constant starting rate for all six steps. The third example, figure 8.5, changes the slope and number of steps. In this example, the CY512 takes only three steps to ramp to the slew rate. It then slews for two steps, then ramps down for three steps again. By changing the slope, the CY512 works in a ramped mode through a smaller number of steps than required in the first example. The final example, figure 8.6, maintains the same slope and number of steps, but varies the rate parameter. The CY512 ramps, as in the third example, but the step times are different, due to the new rate parameter. This is especially evident during the slew steps.

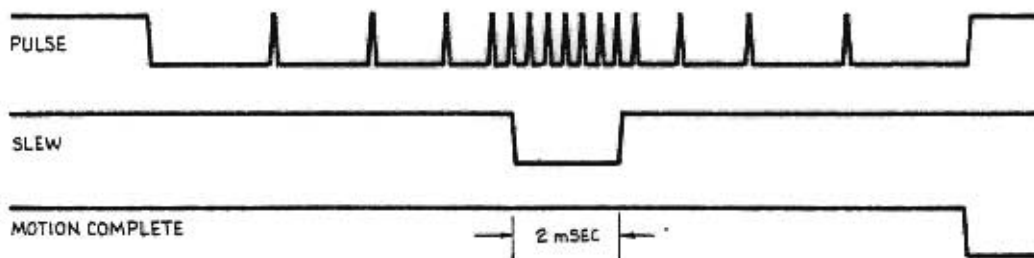


Figure 8.3 Ramp Rate Timing Example: R 253) S 50) F 1) N 16) G)

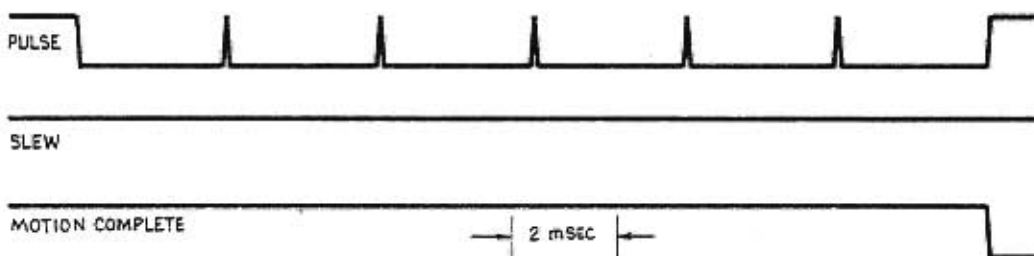


Figure 8.4 Ramp Rate Timing Example: R 253) S 50) F 1) N 6) G)

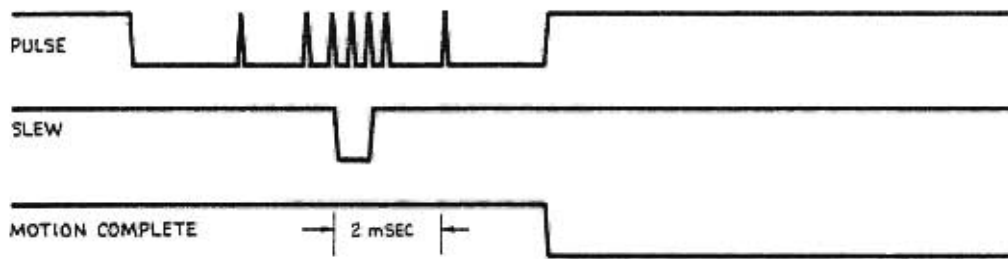


Figure 8.5 Ramp Rate Timing Example: R 253) S 80) F 1) N 8) G)

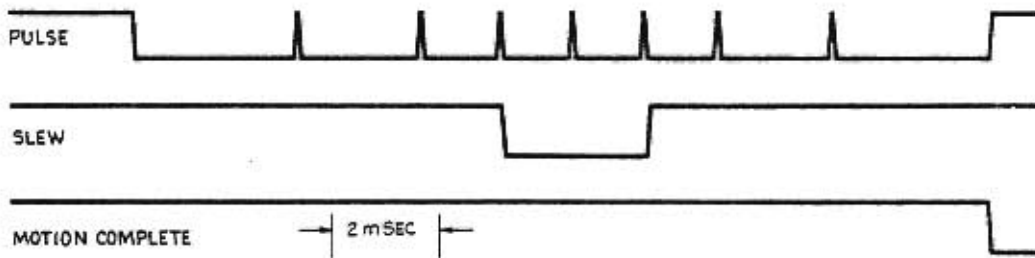


Figure 8.6 Ramp Rate Timing Example: R 240) S 80) F 1) N 8) G)

From the Rate Equation it is possible to derive Stepping Rate vs. Elapsed Time and Elapsed Time vs. Position relationships. The plots shown in Figure 8.2 indicate Stepping Rate vs. Position. The Rate Equation yields the step rate for each step, which is the inverse of the time duration of each step. In order to get elapsed time from the equation, it is necessary to sum the times for all the steps involved. During acceleration, the time duration of step  $n$  is given by:

$$t(n) = (256-r)*80\mu\text{sec} + 10*(255 - n*s)\mu\text{sec} + 75\mu\text{sec} \pm 7.5\mu\text{sec}$$

where  $n$  ranges from 1 to  $y$ , the number of ramp steps. This time will be in microseconds for the values shown, and is directly related to the Rate Equation. The step rate for step  $n$  would then be  $R(n) = 1/t(n)$ .

To determine the elapsed time for  $n$  steps, simply add the step times for each of the  $n$  steps. This may be done one step at a time, or by using the following equation:

$$T(n) = n*[(256-r)*80 + 75 + 2550 - (n+1)*5*s] \mu\text{sec} = \sum_{x=1}^n t(x)$$



The following plots show Rate vs. Elapsed Time, [R(n) vs. T(n)], and Elapsed Time vs. Position, [T(n) vs. n]. Various rates and factors are shown, with five different values for slope in each case.

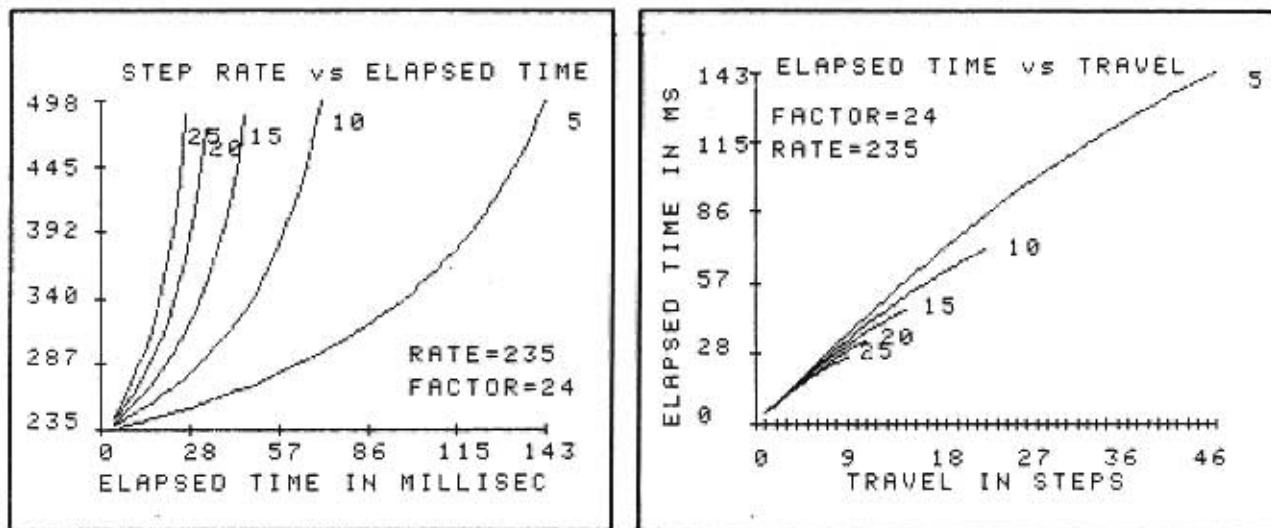


Figure 8.7 Acceleration Plot: R 235) F 24) for S = 5,10,15,20,25

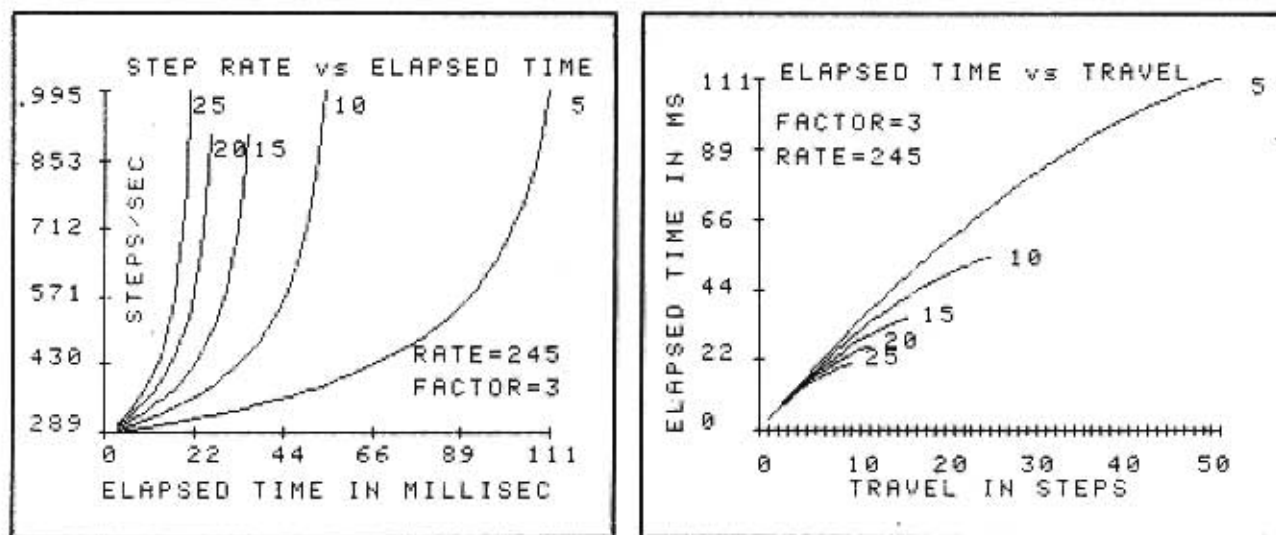


Figure 8.8 Acceleration Plot: R 245) F 3) for S = 5,10,15,20,25

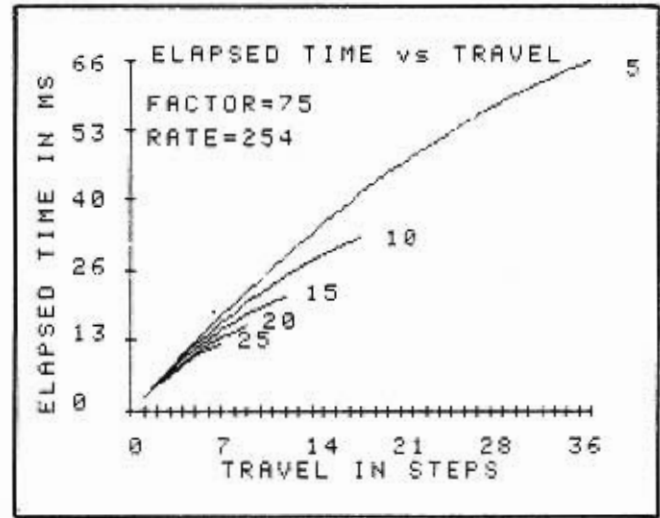
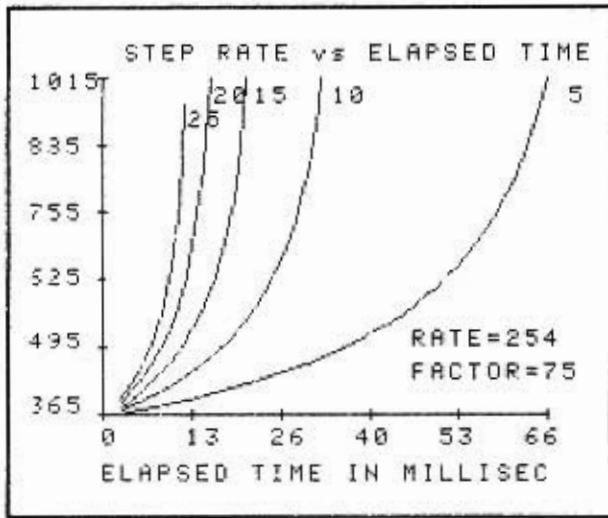


Figure 8.9 Acceleration Plot: R 254) F 75) for S = 5,10,15,20,25

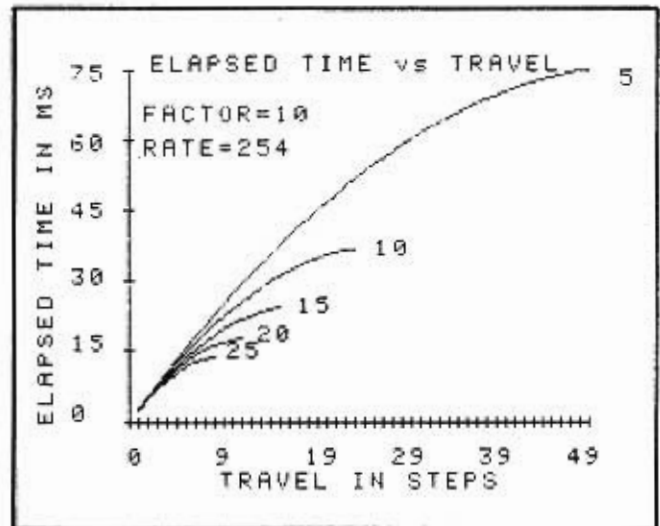
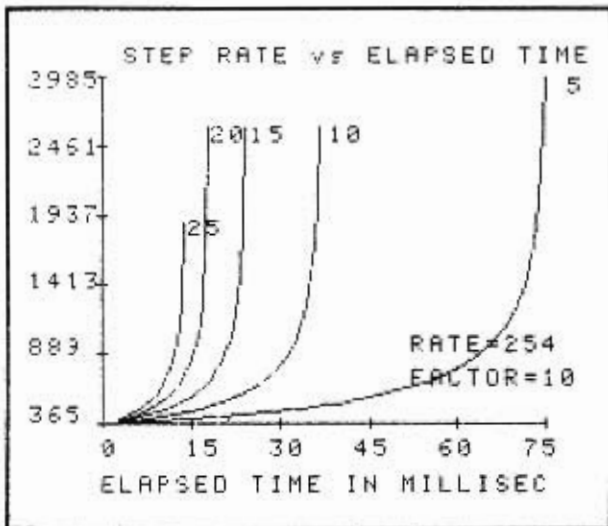


Figure 8.10 Acceleration Plot: R 254) F 10) for S = 5,10,15,20,25

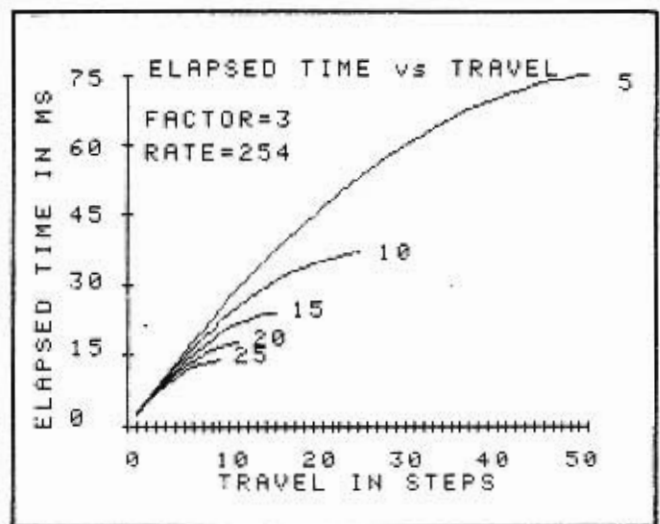
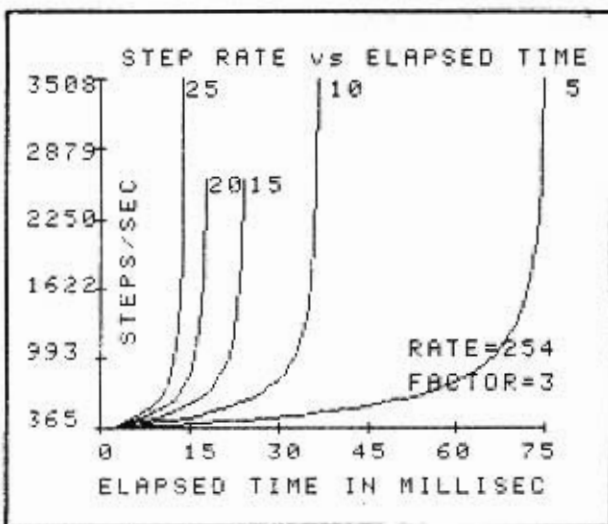


Figure 8.11 Acceleration Plot: R 254) F 3) for S = 5,10,15,20,25

## SLEW MODE OPERATION

Many applications require that the stepper move at its maximum stepping rate. In most situations, the mechanical constraints on the system prevent the motor from accelerating from zero speed to maximum velocity in one step. For this reason, it is desirable to accelerate or "RAMP UP" to step at the desired rate. The SLOPE command provides the "RAMP RATE", defined as the change-in-factor/step. The maximum rate desired is specified by the RATE and FACTOR commands as usual, and the number of steps desired is specified by the "N" command, as usual. A typical instruction sequence is shown below. Note that FACTOR should be the last of the SLOPE and FACTOR commands to be executed, as it computes an internal value based on the other parameters.

```
N 1095) ;set number of steps = 1095
R 220) ;set maximum rate = 335 steps/sec (see Table VII)
S 12) ;change in factor parameter per step
F 1) ;minimum factor parameter value
G) ;begin stepping
```

This instruction sequence sets the number of steps to be taken, the maximum rate of travel, and specifies the change in the factor parameter per step. The ramped operation is generally satisfactory at low or medium speeds but may require external timing at higher slew rates. Note that at the higher step rates, the dependence of step rate on rate parameter is non-linear as shown in Table VII. Also, the interaction between the various parameters is non-linear, as indicated in the various curve plots. Some combinations of values may accelerate your motor to higher slew rates than other combinations. The user is encouraged to experiment with various values for the parameters before resorting to a more complex timing scheme. An example of such timing is shown in Figure 8.12c. In this example the BITSET instruction precedes the GO instruction in the program. This causes pin #34 to go HI and the capacitor begins charging. The output of the V/F follows the voltage step. While this circuit does not provide the ideal deceleration, it is suggestive of the type of open loop external control possible.

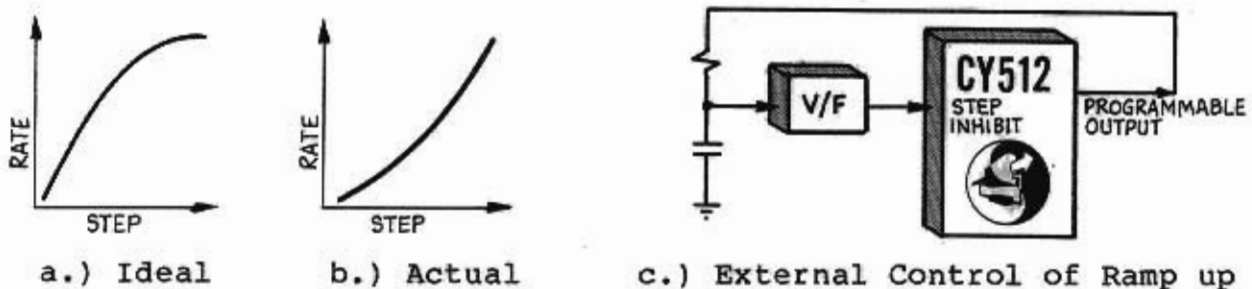


Figure 8.12 Actual and Ideal acceleration curves and example external ramp-up control circuitry.

### Slew Mode Example

Consider a command sequence to cause the CY512 controller to accelerate the motor with a slope that decreases the factor parameter by 5 with each step taken, until a maximum rate of 180 steps/sec is reached (this corresponds to  $r = 190$  and  $f = 20$ ...see Table VII) and then decelerate from this maximum speed to stop 513 steps from the start position. To enter this command sequence into the CY512 program buffer, we send "E" followed by ")", then the command string terminated by "Q" for QUIT. Parameter values may be set via commands prior to program loading, thus allowing all of the program buffer to be used for active instructions. Execution of the program begins when a "D" (DOITNOW) is sent to the CY512. Note that a single motion of this kind may also be executed directly, without loading the program buffer.

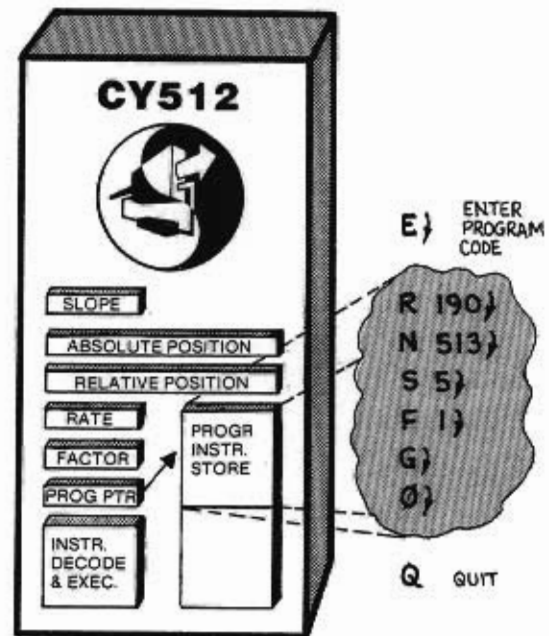


Figure 8.13  
CY512 Programming Example.

## CLOSED LOOP CONTROL

The constant slope acceleration implemented in the CY512 controller allows higher step rates than would be achievable with non-ramped stepping signals. The maximum rate attainable with a small slope may be sufficient, however the long acceleration times may be impractical for a given application. As is generally true, optimum performance can be obtained via closed loop feedback, and this is true for the CY512. Although the term "closed loop control" often indicates a rather high level of complexity and circuit analysis, the CY512 provides for closed loop operation with its corresponding optimal performance via the use of a slotted disk attached to the stepper shaft and an economical optical "interrupter" module of the type commercially available from several sources. The use of a "slotted disk" on the shaft to break a light beam allows a position signal to be fed back to the CY512. As shown in Figure 8.14, this signal may be converted to digital levels via a Schmitt Trigger and used to trigger each step. Thus at low rates, the motor accelerates normally and the rotor is in position when the next step signal

arrives. At higher rates, the next step signal may occur before the motor has completed the last step. In this case the optical feedback will cause the trigger input to be low and therefore prevent the step from occurring until the previous step is completed. Thus, maximum performance is obtained for the given stepper motor. Note that the standard shaft encoder would also work in place of the slotted disk assembly. The CY512 only requires some mechanism for controlling the STEP INHIBIT and TERMINATE lines. The circuit shown below runs the Step Inhibit line, which will slow the CY512 from a rate which is too fast for the motor. If it is also necessary to speed up the step rate, providing complete feedback control, the circuit below may be combined with the Abort/Terminate separation logic shown in Figure 8.15.

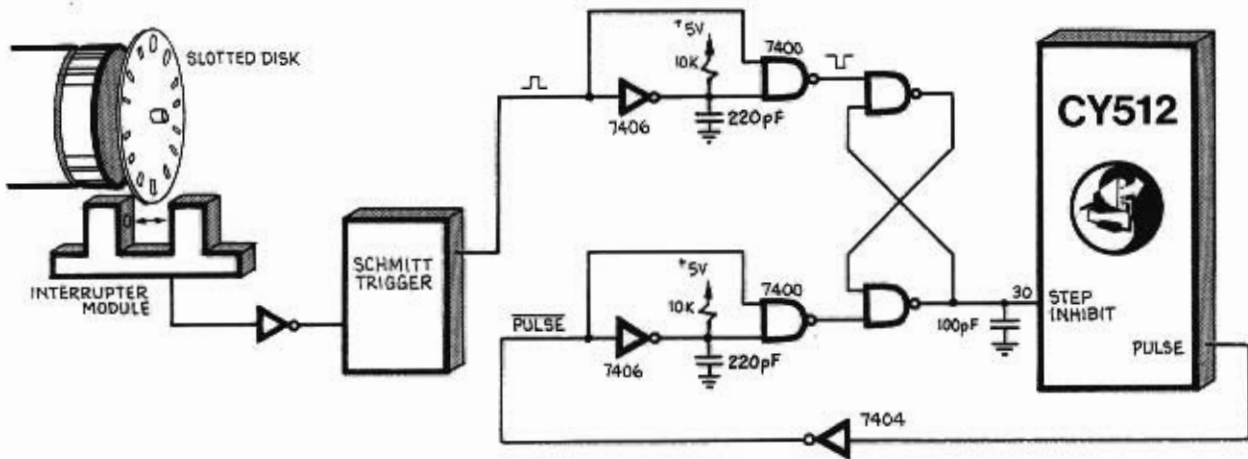


Figure 8.14 An interrupter module with slotted disk provides position feedback to assure that the maximum step rate never exceeds the motor's ability to keep up with the controlled fields, and the motor can accelerate at its maximum rate. Note that some interrupter modules include a Schmitt Trigger to produce a TTL output that can be input directly to the CY512.

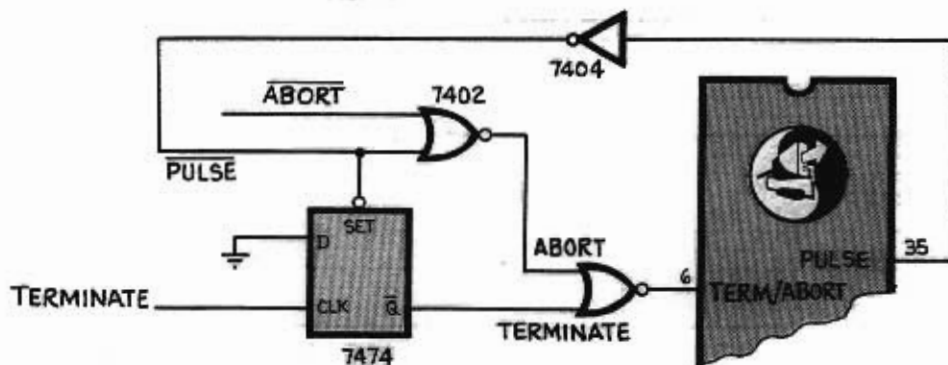


Figure 8.15 ABORT/TERMINATE Separation Logic.

**ABSOLUTE MAXIMUM RATINGS:**

Ambient Temperature under bias.....0°C to 70°C  
 Storage Temperature.....-65°C to +150°C  
 Voltage on any pin with respect to GND...-0.5V to +7V  
 Power Dissipation.....1.5 Watts

TABLE VIII		DC & OPERATING CHARACTERISTICS			
(T <sub>A</sub> = 0°C to 70°C, V <sub>CC</sub> = +5V±10%)					
SYMBOL	PARAMETER	MIN	MAX	UNIT	REMARKS
I <sub>CC</sub>	pwr supply current		100	mA	
V <sub>IH</sub>	input high level	2.0	V <sub>CC</sub>	V	(3.8V for XTAL <sub>1,2</sub> , RESET)
V <sub>IL</sub>	input low level	-.5	.8	V	(0.6V for XTAL <sub>1,2</sub> , RESET)
I <sub>LO</sub>	data bus leakage		10	µA	high impedance state
V <sub>OH</sub>	output hi voltage	2.4		V	I <sub>OH</sub> = -40 µA
V <sub>OL</sub>	output low voltage		.45	V	I <sub>OL</sub> = 1.6 mA
F <sub>CY</sub>	crystal frequency	2	11	MHz	see clock circuits

**ELECTRICAL CONVENTIONS**

All CY512 signals are based on a positive logic convention, with a high voltage representing a "1" and a low voltage representing a "0". Signals which are active low are indicated by a bar over the pin name, i.e., PULSE.

All input lines except I/O Request, Terminate/Abort, and I/O Select include 50K ohm pull-up resistors. If the pins are left open, the input signals will be high.

The data bus is bidirectional, and is tri-state during nonactive modes. Note that data bus signals are positive logic, and all command letters are upper case ASCII.

## RESET CIRCUITRY

The RESET (pin #4) line must be held low upon power-up to properly initialize the CY512. This is accomplished via the use of a 1  $\mu$ F capacitor as shown in Figure 9.1. RESET must be low for 10 msec after power stabilizes on power-up. Once the CY512 is running, RESET need only be low for about 15  $\mu$ sec (6 MHz crystal).

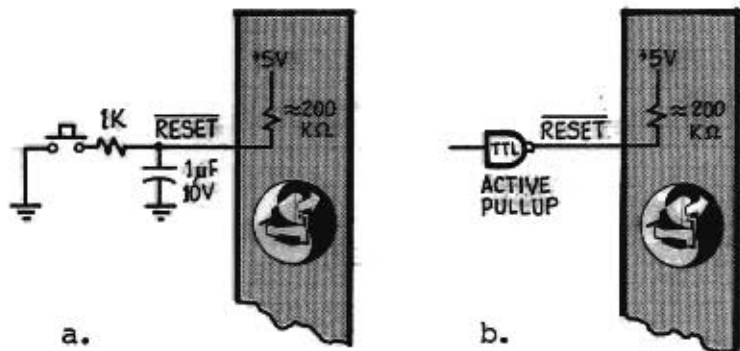
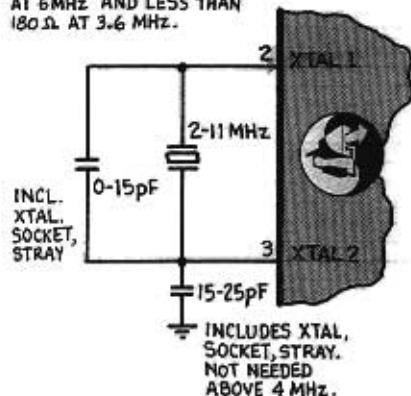


Figure 9.1 a) Reset Circuitry.  
b) External Reset.

## CLOCK CIRCUITS

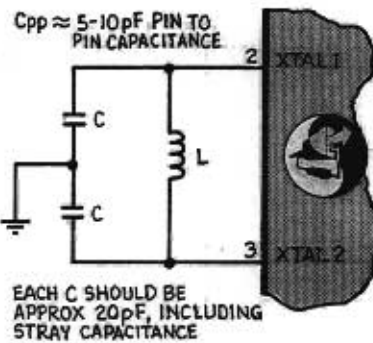
The CY512 may be operated with crystal, LC, or external clock circuits. These three circuits are shown in Figure 9.2. Unless otherwise specified, all timing discussed in this manual assumes a 6MHz series resonant crystal such as a CTS Knights MP060 or Crystek CY6B, or equivalent. The CY512 will operate with any crystal from 2 to 11 MHz, including a standard 3.58 MHz TV color burst crystal. All timing values specified in this manual will be changed by using different crystal frequencies. Time values must be scaled by  $6/f_{cy}$ , and stepping rates must be scaled by  $f_{cy}/6$ , where  $f_{cy}$  is the crystal frequency in MHz. Note, however, that the "X" command is calibrated for milliseconds at 11 MHz.

CRYSTAL SERIES RESISTANCE SHOULD BE LESS THAN  $75\Omega$  AT 6 MHz AND LESS THAN  $180\Omega$  AT 3.6 MHz.

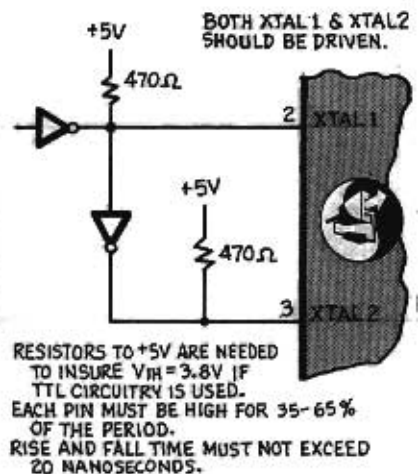


CRYSTAL

$$C' = \frac{C + 3C_{pp}}{2} \quad f = \frac{1}{2\pi\sqrt{LC}}$$



LC



EXTERNAL

Figure 9.2 Clock Circuits for CY512.

**CY512/KIT**

Since the CY512 contains all the logic needed to operate a stepper motor, including instruction decoding, parameter maintenance, and step timing, very little external circuitry is required to get a minimal stepper motor subsystem operating. A circuit indicating what is required is shown in Figure 10.2. As a convenience to our customers, Cybernetic Micro Systems has implemented such a circuit on a small printed circuit board. This board is made available as a kit, including all parts necessary to put the board together. A photograph of the CY512/Kit is shown in Figure 10.1.

The design consists of the CY512 with associated peripheral parts (crystal, socket, capacitors, etc.), buffers with LED indicators on CY512 output signals, switches for CY512 inputs, Abort/Terminate logic for closed loop motor operation, and a simple, unipolar driver circuit for the motor. Three edge connectors make the various signals available to the rest of the system. Signals are divided into a Data Interface, Secondary Control lines, and the Motor and Power Supply connections. Finally, about half of the board consists of a wire-wrap area, useful for special data interfaces or motor driver circuits.

The kit supplies all parts needed to assemble the board, with assembly requiring only a few hours. In addition to the kit, the user must have a power supply, a source of commands for the CY512 (keyboard or computer), and a four-phase stepper motor. Complete documentation is provided. The CY512/Kit is ideal for prototyping, allowing first time users of the CY512 to quickly and easily get their part into operation.

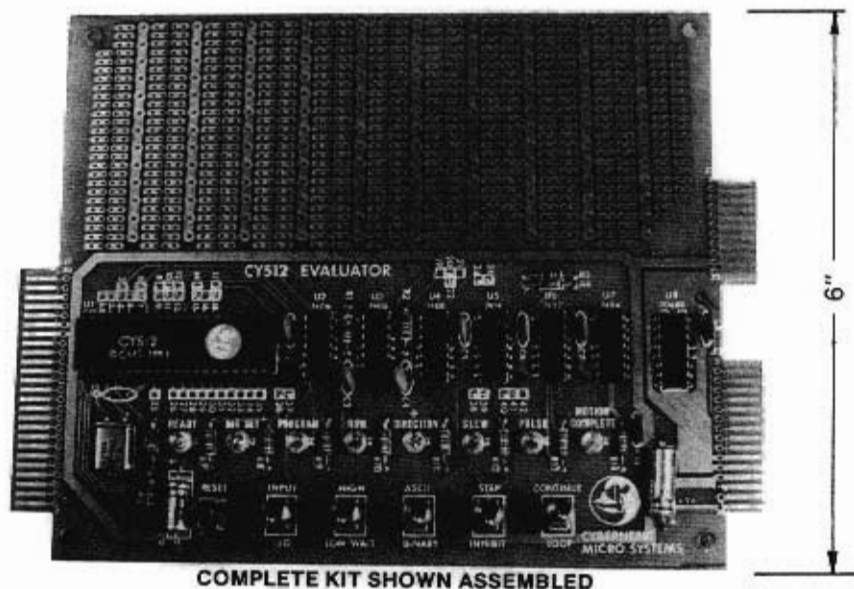


Figure 10.1 CY512/KIT board available for prototyping.



# TEST DEMONSTRATION CIRCUIT

The circuitry shown in Figure 10.2 provides a simple setup that allows use of an ASCII keyboard to control the CY512. It is generally helpful to use LEDs (light emitting diodes) on all output lines to visually display the state and state transitions. This is especially true when working through the detailed timing diagrams in the back of this manual. The programs used in the timing examples have slow rates specified, so that the transitions are visible to the user if LEDs are used on the outputs. A slower crystal frequency will also help. The CY512/KIT shown in Figure 10.1 is ideal for this purpose.

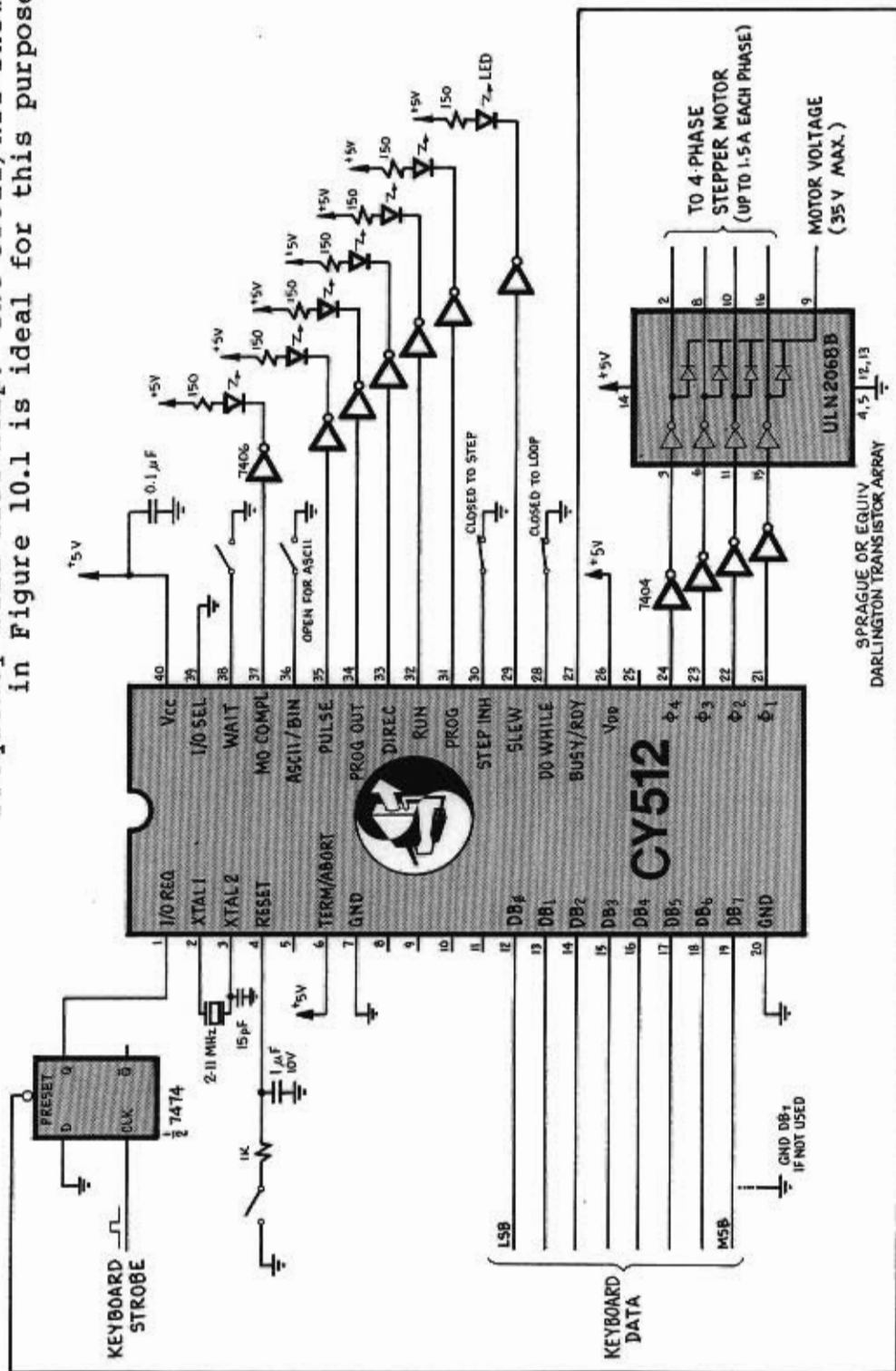


Figure 10.2 Test Demonstration Circuit

## DRIVER CIRCUIT CONSIDERATIONS

The CY512 provides the timing and logical signals necessary to control a stepper motor. However, to make a complete system, a driver circuit must be added to the CY512. This circuit will take the logical signals generated by the CY512 and translate them into the high-power signals needed to run the motor.

The user has two choices in the selection of driver circuits. Existing designs, usually in the form of pulse-to-step translators, may be used, or special designs may be created. Translators usually require a pulse and direction input, or two pulse streams, one for CW stepping and one for CCW stepping. The translator takes the pulse inputs and generates the proper four phase outputs for the motor. **Note that it is also possible to drive motors with this scheme which are not four phase designs.** Since the translator generates the actual motor driver signals, it only requires the pulse timing and direction information generated by the CY512 Pulse and Direction signals. **This allows the CY512 to control three and five phase motors as well as the standard four phase designs.**

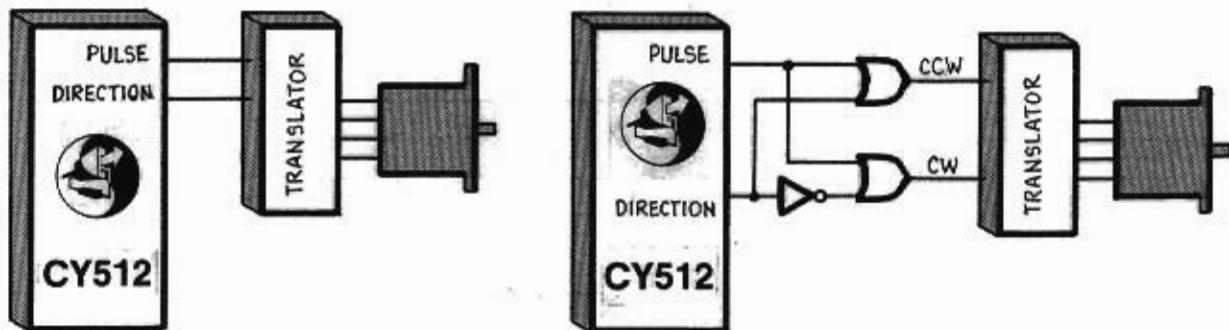


Figure 10.3 CY512 to Translator Driver connections.

If the user opts for his own driver design, the Pulse and Direction lines may be used, or the four phase outputs may directly control the driver circuits. This type of design makes full use of the CY512 signals. The following paragraphs are meant as a guide to various types of driver circuits, but should not be used as final driver designs. Detailed switching characteristics, transient suppression, and circuit protection logic have been omitted for clarity and simplicity.

Unipolar designs are the simplest drivers, and are generally useful when running at less than 600 steps per second. These designs require motors with six or eight leads, since the power supply is connected to the middle of each winding. The end of each winding is pulled to ground through a transistor controlled by one of the phase output lines from the CY512. Motor performance may be improved by adding a dropping resistor between the power supply output and the center tap of each winding. This decreases the field decay time constant of the motor, giving

faster step response. The performance increase is paid for by a higher voltage power supply and heat losses through the dropping resistors. This type of circuit is known as an L/xR circuit, where the x represents the resistor value relative to the winding resistance. An L/R circuit would not have any external resistors, while an L/4R circuit would use a resistor of three times the value of the motor winding resistance. Note that the power supply could be four times the nominal motor value with this circuit. Also note that this circuit requires only a single voltage and one transistor per phase.

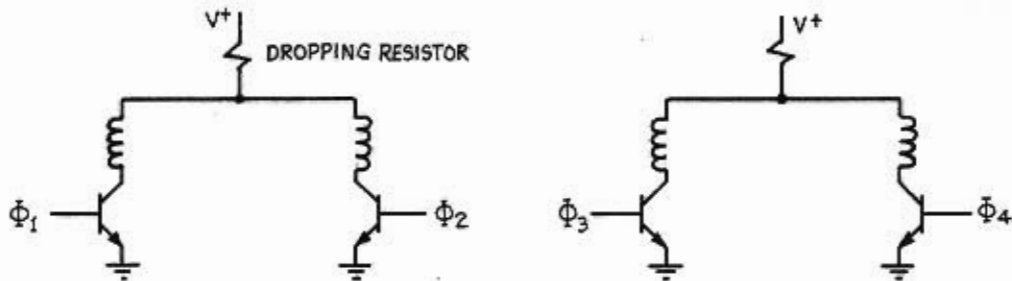


Figure 10.4 Unipolar driving circuits.

The second basic type of driver circuit is the bipolar design. In this case, the motor is driven only from the ends of each winding, with switching logic used to control the direction of current through the winding. These circuits may be implemented with a four lead motor, since only the ends of each winding are needed. Bipolar designs are more efficient in driving the motor, and result in higher performance than the unipolar designs. Two methods of switching the direction of current may be used. With a single voltage power supply, eight transistors are used, two per phase. Transistors are turned on in alternate pairs across each winding to control the current. The second alternative uses only four transistors, but requires a dual voltage power supply. In this case, one side of each winding is connected to ground, and the other side is switched between the positive and negative power supplies. In both designs it is very important to insure that both transistors on one side of the winding are not on at the same time, as this would short the power supply through the transistors, generally destroying the transistors in the process. Protection logic is usually included to insure that one transistor is off before the other is allowed to turn on.

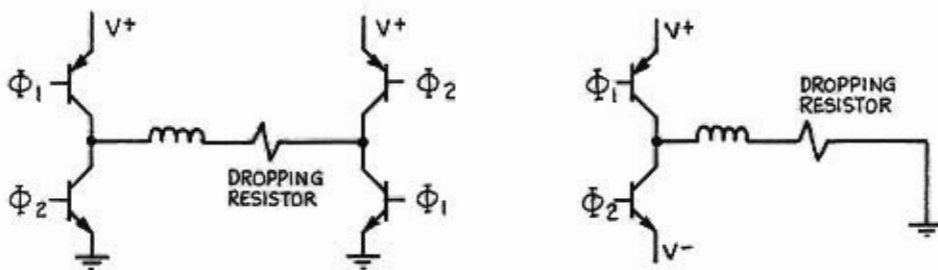


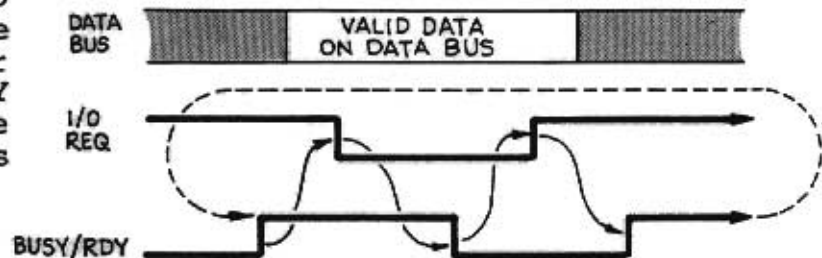
Figure 10.5 Bipolar driver designs.

The most advanced driver designs are variations on the unipolar or bipolar types, although they are generally implemented using the bipolar approach. These drivers are capable of the highest step rates attainable. They work by switching current or voltage through the motor at much higher than the rated value. This is done for only a short period of time, causing the magnetic field in the motor to change very quickly, without exceeding the maximum power dissipation of the motor. As long as the average dissipation does not exceed the motor rating, the motor will perform without problems. Once the maximum limit is reached, the motor may overheat and self destruct. One technique for increasing motor performance would simply apply a high voltage to the motor at the beginning of each step. This makes the motor react very quickly to the change in phase signals. After a short period of time, the voltage is switched to a lower value, allowing the motor to continue its motion without overheating. A second approach, known as a constant current design, senses the amount of current flowing through the winding, and adjusts the voltage applied to the motor to maintain the current at its maximum rated value. At the beginning of a motion, the voltage would be low, with a constant adjustment to a higher value as the motor speed increases, and back EMF decreases the current draw for a fixed voltage level. Another technique, known as chopping, may also be applied to these driver designs. This approach applies a voltage much higher than the rated value for a short period of time. The voltage is then turned off for another time period. This occurs many times per step, with the frequency of switching known as the chopping frequency. This frequency may be controlled by time, switching at a given rate, or it may be controlled by sensing the current flow through the motor, switching at a variable rate. The highest performance drivers are usually designed as bipolar chopper circuits.

The user should consult design guides available from the various motor manufacturers for additional information.

## HANDSHAKE PROTOCOL

All commands and data transmitted from the master processor to the CY512 peripheral processor are sent asynchronously with complete handshaking performed. The master processor waits for the CY512 READY line to go HIGH before sending the active LOW I/O REQUEST signal. The data may be placed on the bus at any time prior to the HIGH-to-LOW transition of I/O REQUEST. The data should be stable on the bus until the CY512 RDY line goes LOW, indicating that the transfer has been acknowledged and that the CY512 is BUSY processing the command or data. The master then brings I/O REQUEST to the HIGH state. The next transfer can occur as soon as BUSY/RDY returns HIGH. The sequence described is shown in Figure 10.6.



Example 8080/85 Driver: ASCII mode operation Bit 7 of data used as I/O REQUEST strobe, Routine entered with ASCII in C-register.

		SENDCHR:	
0069	DBED	IN	STATUS
006B	E620	ANI	READY
006D	CA6900	JZ	SENDCHR ;WAIT TIL READY
0070	79	MOV	A,C
0071	E67F	ANI	7FH
0073	D3EC	OUT	DATA ;WITH I/O REQ LOW
		:	
		BUSY:	
0075	DBED	IN	STATUS
0077	E620	ANI	READY
0079	C27500	JNZ	BUSY ;WAIT TIL BUSY
007C	3EFF	MVI	A,0FFH
007E	D3EC	OUT	DATA ;I/O REQ HIGH
0080	C9	RET	

Figure 10.6 Data Transfer Handshake Sequence

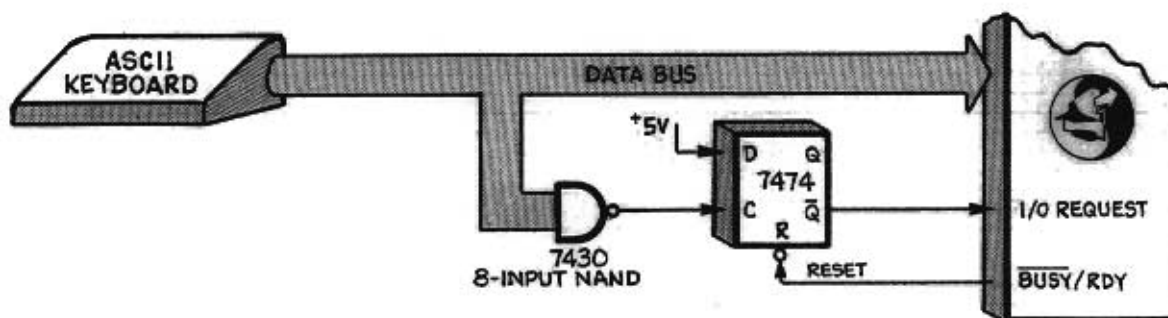


Figure 10.7 Write Strobe Generator for keyboards without strobe.

In the example shown in Figure 10.8, the CY512 is operating in the PARALLEL ASCII input mode. In this mode, bit 7 is always zero and  $b_7$  line of the CY512 data bus may be tied to ground. Since the user will normally transfer bytes of data from memory to the output port, the most significant bit of the data byte may be used to generate the I/O REQUEST strobe, thus allowing only one 8 bit output port to suffice. The "SENDCHR" routine, shown in Figure 10.6, demonstrates the coding used to achieve this. Of course, a separate port line may be used to generate I/O REQUEST, if this is desired. If the CY512 is operated in the PARALLEL/BINARY mode, all 8 data bus lines are used, and a separate I/O REQUEST line is required. Note that in the example shown, use is made of the fact that the data and the I/O REQUEST signal may be applied simultaneously in parallel operation. If Verify mode is to be used, all 8 bits of the data bus must be free to operate bidirectionally. In this case, it is generally best to make I/O REQUEST and I/O SELECT separate lines from the data ports. See Timing and Control Information in section 7.

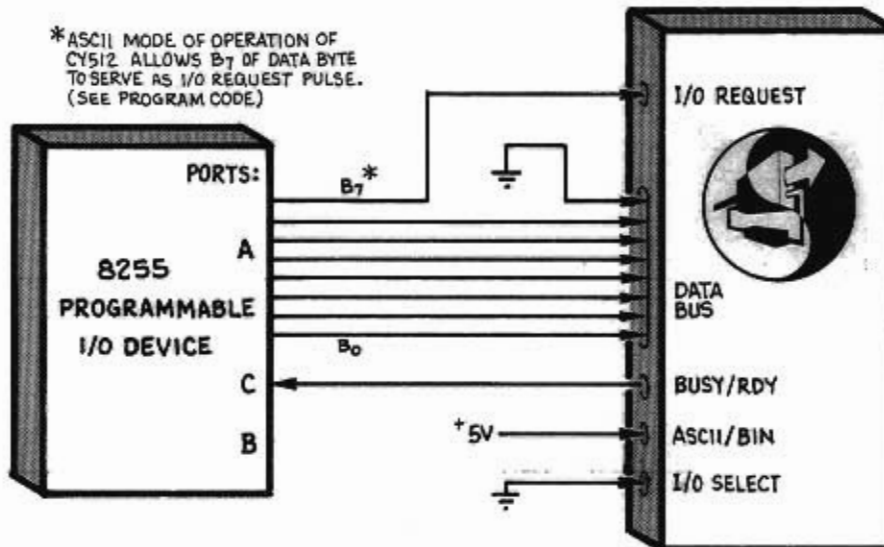


Figure 10.8 Example Interface to CY512 using 8255 PIO.

## OPERATION OF SEVERAL CY512S USING A COMMON DATA BUS

In systems where multiple CY512s are to be controlled by a host computer it is possible to use one eight-bit port to establish a common data bus for sending instructions to the CY512s. Each of the separate RDY lines (pin 27) of each CY512 must be monitored individually and each I/O REQUEST line (pin 1) must be activated separately. This technique effectively uses the I/O REQUEST line as a chip select (CS). A CY512 will ignore all bus information if its I/O REQUEST line is inactive.

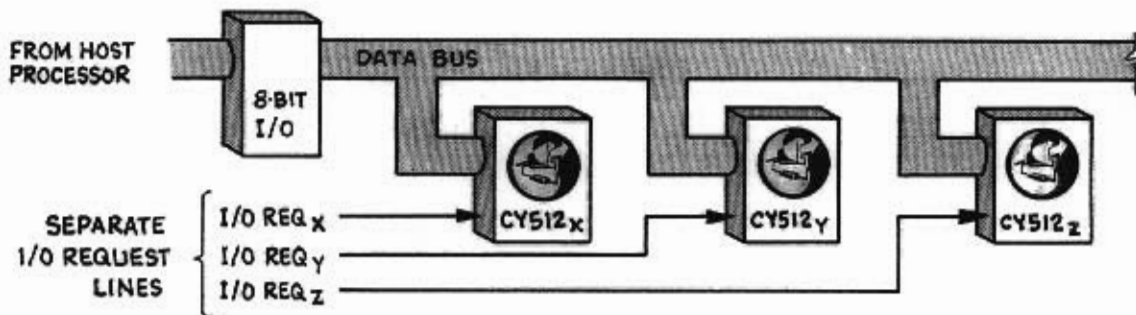
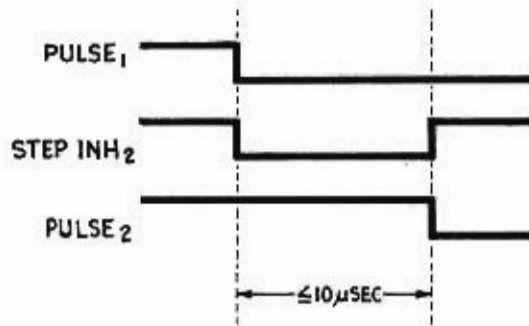


Figure 10.9 CY512s share common data bus by using separate I/O REQUEST lines for chip select.

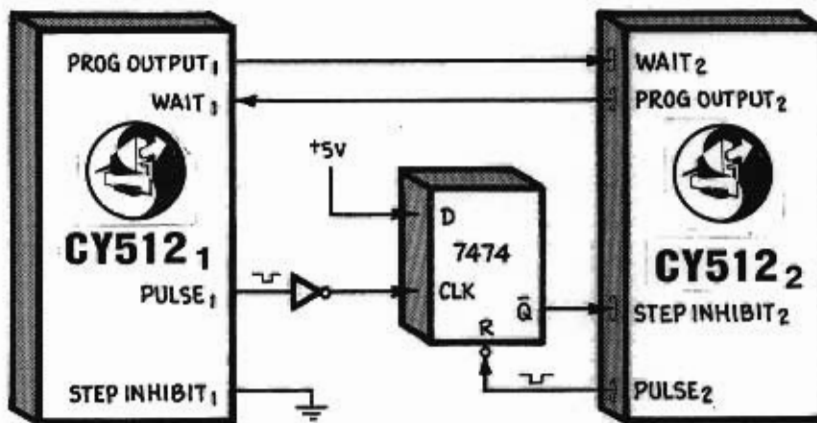
## SYNCHRONIZATION OF TWO CY512S

Two CY512s, executing the same program, may be synchronized as shown in Figure 10.10. The master controller can control the WAIT line of the slave CY512 via the BITSET or CLEARBIT commands. The slave CY512 is started first, with a DOITNOW command, and executes a WAIT command and waits until the wait line (pin #38) is driven low by the CLEARBIT command executed by the master CY512 when it receives the (second) DOITNOW command. Both CY512s

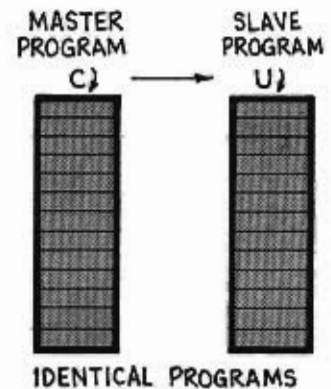
then proceed to the next instruction and are synchronized as shown in Figure 10.10a, to within approximately 10 microseconds. Note that when the two programs are not identical, the master can also wait for the slave to execute its own CLEARBIT instruction, and thereby achieve a more general synchronization.



a.) Timing Diagram



b.) Hardware



c.) Software

Figure 10.10 Synchronization of two CY512s.

## COORDINATION OF SEVERAL CY512S

Multiple CY512s may be synchronized to each other by use of the Programmable Output line, the Wait functions, DOWHILE signal, and time delays. These may also be combined with other signals, such as Direction, Slew, or Motion Complete, used to select the point in the motion when the signal is presented to the waiting controller. Consider a general parts handling function in which the part must be handed off between two controllers. The geometry of the parts and the arms used to carry the parts requires that the hand off be carefully synchronized between the two controllers. The one to receive the part waits at the receiving position until the CY512 which has the part signals that it has arrived. The two arms then move together in a coordinated motion, reaching a point at which the distance between them is a minimum. The part is exchanged and the arms move apart, again in a coordinated motion. Once a certain position is reached, the arms are free to move independently, and continue with their assigned programs. If the motion is repetitious, both controllers can work with the same program, always being resynchronized at the hand off. The following program illustrates such a motion.

```

H)    Use halfstep mode
A)    Declare current position as home
R 225) }
S 25)  } Define stepping parameters
F 5)   }
P 14)  Move to the receiving position
E)    Define hand off program
U)    Wait for a part to arrive
P 0)   Arms move together to handoff position
+)    Change direction
C)    Activate mechanism to transfer part
X 90)  Delay for part to actually transfer
P 14)  Move apart, back to receiving position
X 90)  Delay for part to stabilize, arms apart
P 108) Transport part to next handoff position
-)    Low DIR & PROG OUT indicates part arrived
P 122) Move together with receiving arm
B)    Release mechanism which holds part
X 90)  Delay for part transfer to receiving arm
P 108) Move apart, back to receiving position
X 90)  Delay for part to stabilize, arms apart
R 20)  Change step rate to slower rate
P 0)   Move empty arm back for next part
P 14)  Stay at the receiving position
R 225) Change rate back to faster rate
T)    Repeat program if DOWHILE low
Ø)    Else stop program
Q)    End of program
D)    Run program
    
```

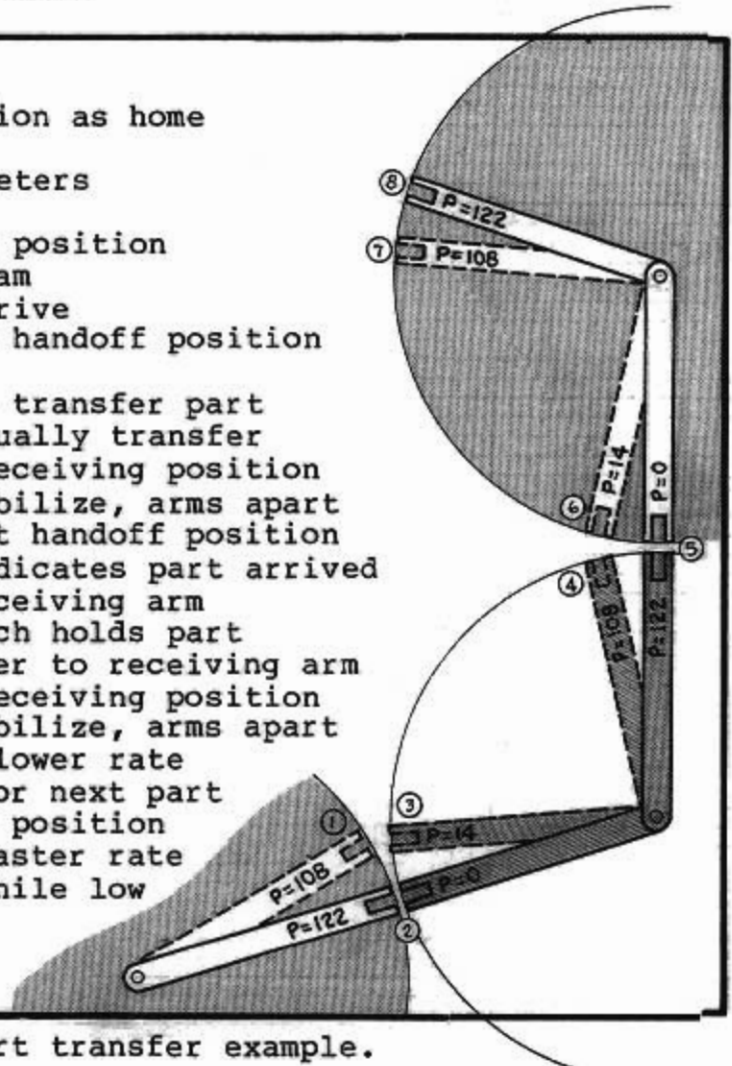
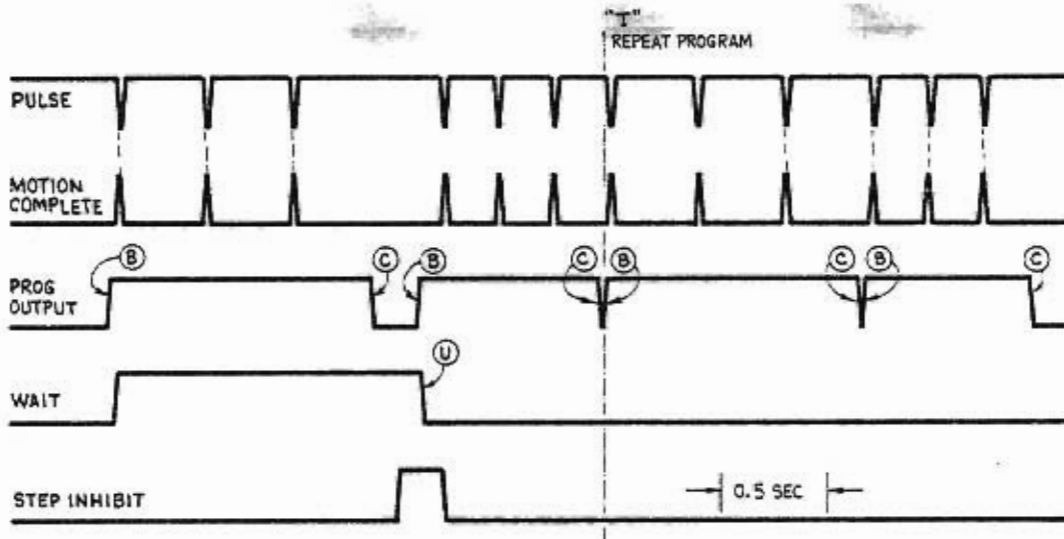


Figure 10.11 Synchronized part transfer example.



## EXAMPLE PROGRAMS AND WAVEFORMS



```
R 20)
F 80)
N 1)
C)
```

E)

```
B) ←
G) ←
X 500)
L 3,1)
C)
U)
B) ←
G) ←
X 330)
L 3,11)
C)
T)
Ø)
```

Q

D)

The timing sequence for a typical program is shown in Figure 10.12. In this example, the rate parameter, factor, and number of steps are present before entering the program-entry mode via the "E" command. These parameters are chosen to allow easy observation of the outputs using the test/demonstration circuit shown in Figure 10.2. The program entered sets the programmable output (pin #34), then takes three steps, clears the programmable line, and waits for the WAIT line (pin #38) to go low when the wait UNTIL instruction is executed. As shown, the STEP INHIBIT line has gone high, and the CY512 waits for this line to go low before stepping. The three-step motions are done one step at a time, using the LOOP command and a time delay between each step. The time delay is used to create a very slow step rate, which can be more easily observed. If the Dowhile line (pin 28) is low when the Til command is executed, the program will repeat from the beginning. When pin 28 is high, the program stops and the CY512 returns to the Command mode. Two program loops are shown in the waveforms.

Figure 10.12 Sample Program and Timing Diagram.

Figure 10.13 provides timing relations for a command sequence that inputs the parameters and executes a "G" command to begin stepping. The I/O REQUEST, BUSY/RDY, and INSTROBE signals are related to the data bus and several outputs are shown as a function of the STEP INHIBIT input.

```

COMMAND MODE INPUT SEQUENCE:

R 10)      set RATE = 10
S 255)     set SLOPE = 255
F 80)      set rate FACTOR = 80
N 4)       set NUMBER of steps = 4
G)         GO, begin stepping
  
```

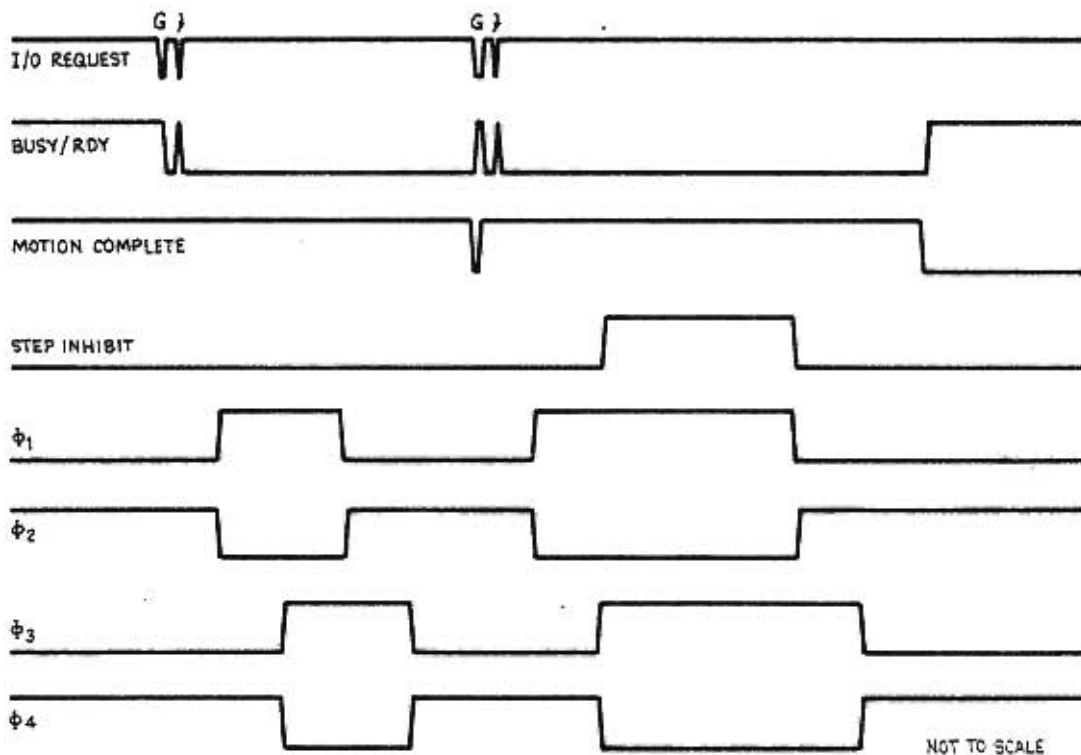
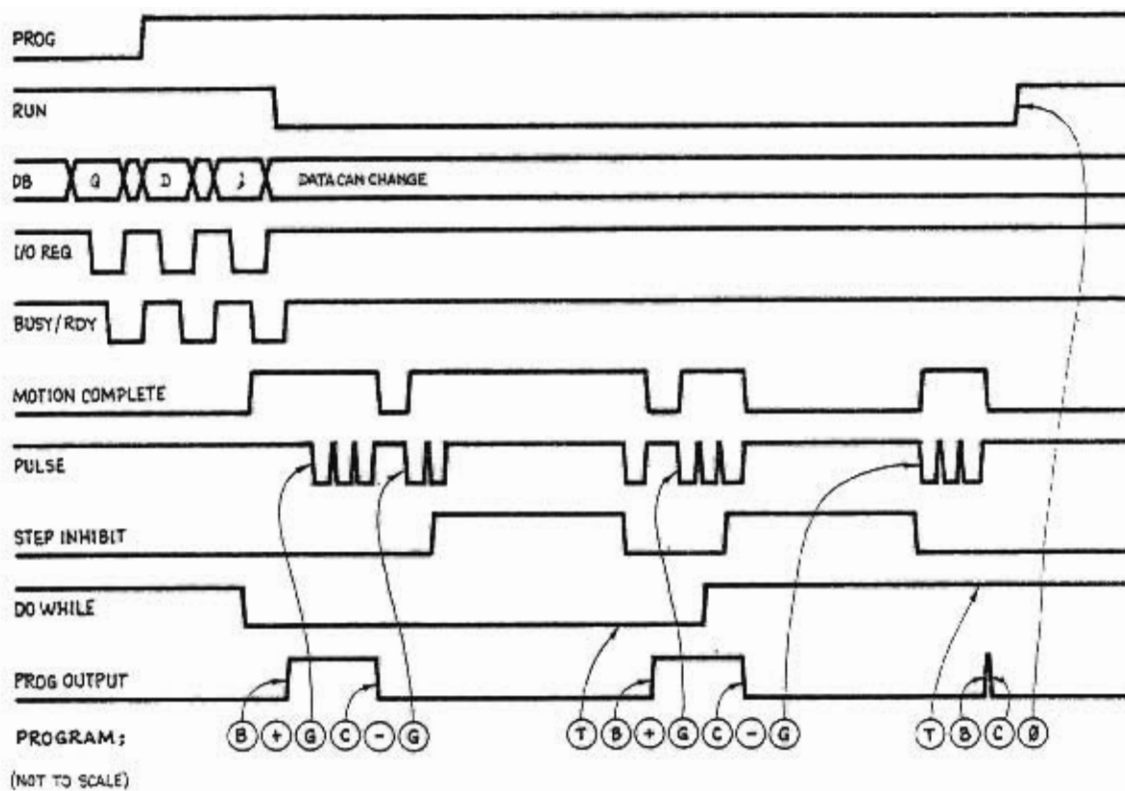


Figure 10.13 Timing Diagram for Commands.

The use of the "loop TIL" instruction is illustrated in Figure 10.14. The PROG and RUN outputs are also shown as a function of the "Q" and "D" commands and the "Ø" instruction. The program loops until the DOWHILE line (pin #28) goes high, then fetches the next instruction. The effect of the STEP INHIBIT input on the MOTION COMPLETE output is also shown.



PRESET: C) clear output line  
 R 10) set RATE = 10  
 S 255) set SLOPE = 255  
 F 50) set FACTOR = 50  
 N 3) set NUMBER steps = 3

ENTER PROG: E)  
 PROGRAM CODE B) set output line  
 +) set CW direction  
 G) GO, begin stepping  
 C) clear output line  
 -) set CCW direction  
 G) GO, begin stepping

T) repeat above prog Til DOWHILE = HI  
 B) set output line  
 C) clear output line  
 Ø) exit run mode, enter command mode

QUIT: Q  
 EXECUTE: D) DOITNOW

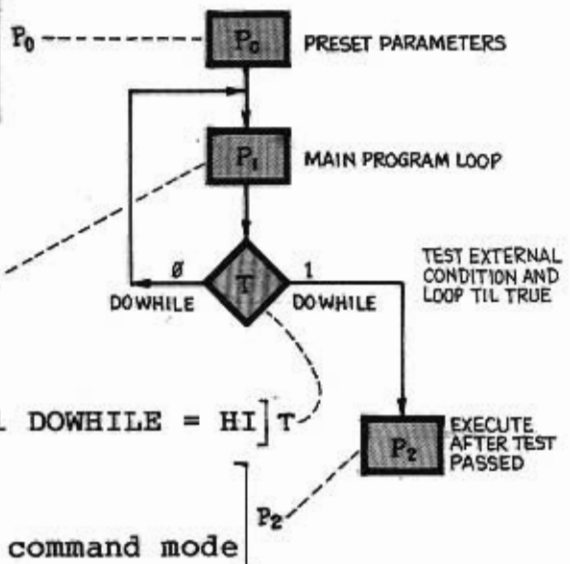


Figure 10.14 Timing and Control for Program Entry and Conditional Looping.

## RS-232-C RECEIVE ONLY INTERFACE DESIGN

When the user wishes to communicate with the CY512 over a serial data link, a special data interface, such as the RS-232-C design shown in this section, must be used. The main component of such a design is the UART (Universal Asynchronous Receiver Transmitter), which transforms the serial data from the data link into the parallel form required by the CY512.

The design shown here is a "receive only" type, meaning that it can only receive data, not transmit. This design will allow the user to send commands to the CY512, but will not allow the Verify mode to work. Bidirectional communication through a UART is very difficult with the CY512, because there is no direct control over the I/O SELECT line or the number of bytes to transmit from the CY512. Those who require the Verify mode must use a more sophisticated design to control the handshake protocol during the verify portion.

As shown in the schematic below, only two signals are needed from the RS-232-C lines. Transmitted Data contains the data sent by the host to the CY512, and Signal Ground is a reference for the data line. Since signals on the RS-232-C interface are not TTL compatible, the transistor circuit connected between Transmitted Data and the UART acts as a converter, generating the TTL equivalent of the data signal for the UART.

The type of UART shown is a single, 40 pin IC. It was chosen because the operating mode is set by connecting the control lines either high or low. Other types of UARTs require a command word to be written to an internal register which controls the mode, something the CY512 is not capable of doing. The type of UART shown is made by several manufacturers, and is readily available. The mode control lines should be connected so that the operating mode of the UART matches that of the host system. This is very important in getting data transmitted properly to the CY512.

Whenever the UART receives a character, the data available line (DAV) goes high. This signal runs I/O REQUEST, indicating to the CY512 that a command character is ready. As the CY512 reads the character, the INSTROBE signal is used to put the character onto the CY512 data bus, by controlling RDE, which brings the received data lines (RD1 to RD8) to their active state. BUSY/READY, connected to RDAV, then resets the DAV signal, clearing the I/O REQUEST. Thus, the standard signals from the UART fully implement the two-line data transfer handshake used by the CY512.

The rest of the circuitry is a baud rate generator. It creates the clock rates needed to operate the UART at most of the common data transfer rates. The 7404 and crystal circuit is an oscillator which runs at 2.4576 MHz. This frequency is an exact multiple of the popular baud rates used. The CD4040 is a CMOS, twelve stage counter. It takes the 2.4576 MHz clock rate and divides it through twelve binary stages, creating one half the frequency of the preceding stage in each case. The outputs are

labeled with the resulting data baud rate, although the actual signal frequency is sixteen times this rate. The clock inputs of the UART should be connected to the desired rate. It will do an internal divide by sixteen, generating the data rate needed by the interface.

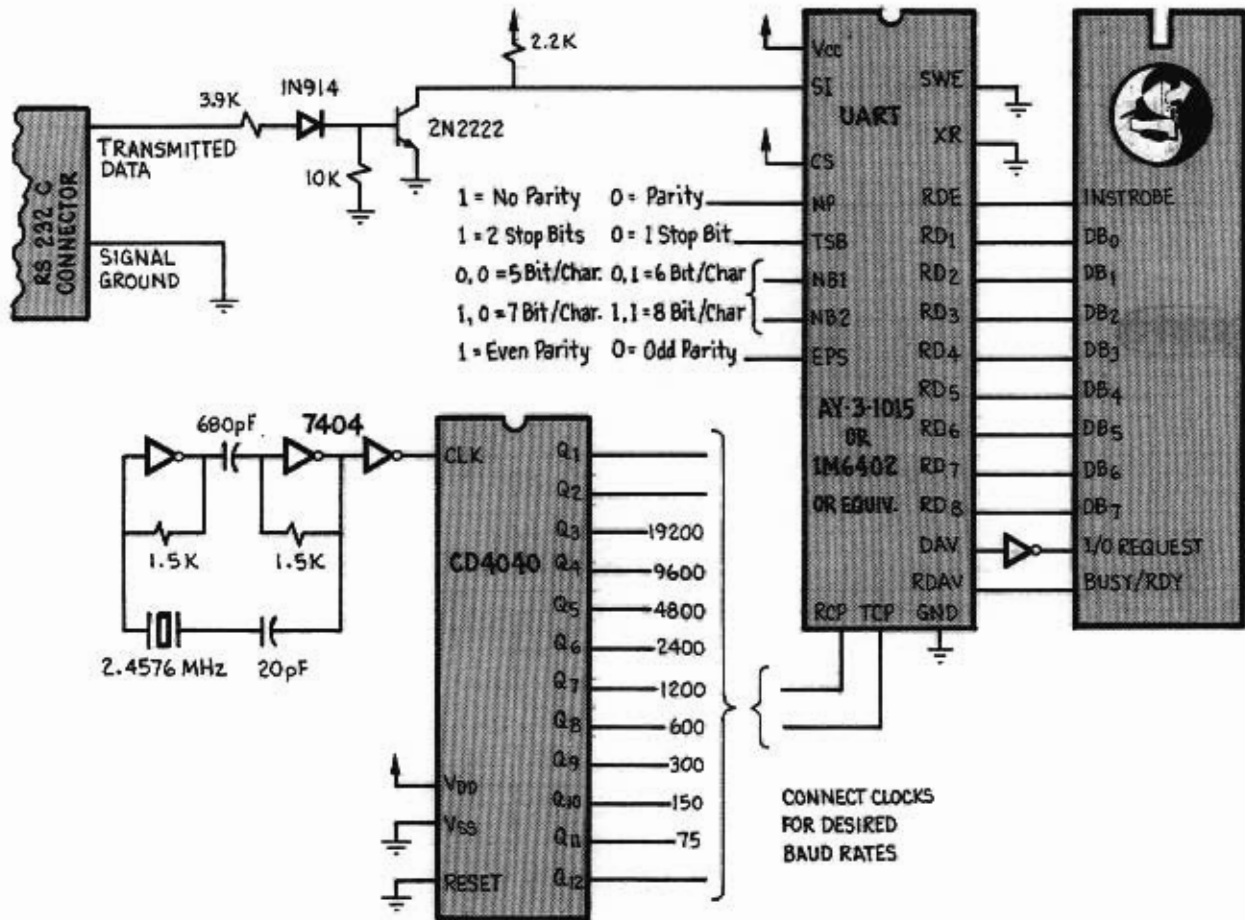


Figure 10.15 RS-232-C Interface Schematic.

## RS-232-C TRANSMIT/RECEIVE INTERFACE WITH CY232

The CY232 Parallel/Serial Network controller enables the user to both transmit and receive data from the CY512 parallel device via a serial RS-232-C port. The actual CY232 to CY512 interface is very easy as shown in the schematic below. However, since the CY232 gives the user the ability to address multiple devices on a network, the CY232 address lines should be tied high or low to provide the CY512 with a specific address, and this address should be used when writing to the CY232/CY512. Also, multiple CY512s can be addressed this way by preceding each with a separate CY232 with a different address or by connecting multiple CY512s to a single CY232. In the second case, the CY232 address decoding logic should be combined with the CY232 DAV to generate a unique I/O REQUEST for each CY512 (see also Figure 10.9). The CY232 manual gives complete details on this interface.

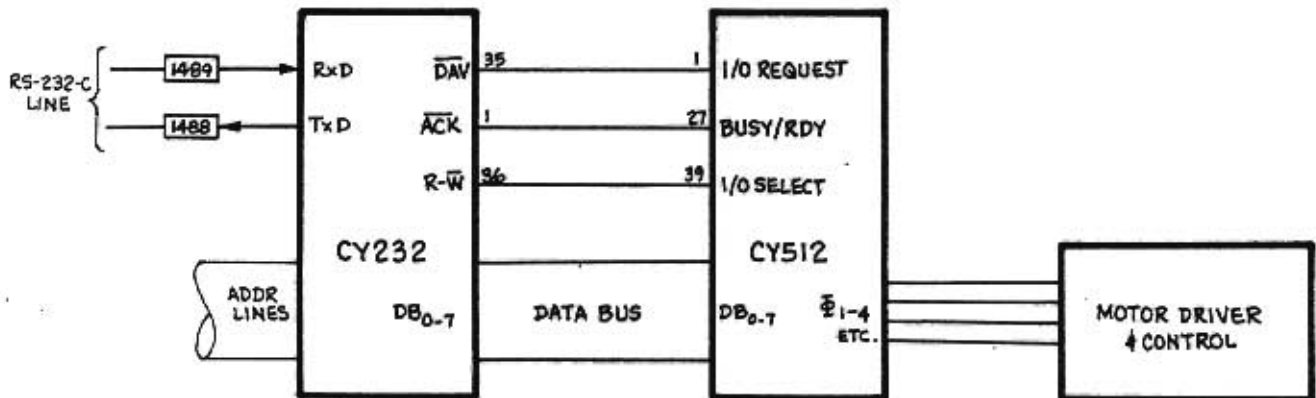


Figure 10.yy CY512 connections to CY232.

## PROM STAND-ALONE INTERFACE DESIGN

When the CY512 is to be used in specific applications, with fixed commands or a small number of different programs, the user may eliminate the need for a keyboard, which is prone to typing errors, and the need for a computer, which may not be justified for the application. By programming the CY512 commands into a PROM or EPROM, a stand-alone design may be generated, in which the program may be selected by switch position, and a push button is used to get things going. The BUSY/READY signal from the CY512 is used to advance the address counter of the PROM, and the hardware automatically loads the commands, one byte at a time, until the end of the program is reached. The end of program then inhibits further program loading until the procedure is restarted by setting the address to the front of a program again.

The circuit shown in this section is started by selecting the desired program starting address for the PROM. With the 74193 counters, any address may be chosen by setting the counter inputs and pulsing the load signal low. The schematic shows the load signal controlled from the CY512 RESET, but a separate load switch could be used. The outputs from the counters control the address inputs to the PROM. Each address corresponds to a single CY512 command character, so the PROM should be organized as eight data outputs per address. Many popular PROMs and EPROMs are organized this way, including 2708s, 2508s, and 6309-1s. Enough address lines must be provided to access the number of bytes required by the program or programs. The design shows eight lines, allowing for 256 bytes, but more could be added by simply cascading additional 74193s.

When the starting address is loaded, the PROM will output the first command byte to the CY512, so the data bus will have the byte ready when the CY512 reads it. When the CY512 becomes ready, with a high level on the BUSY/READY line, the 7400 NAND gate generates a low output to the CY512 I/O REQUEST line. This will tell the CY512 that a command byte is available. The CY512 will read the byte from the data bus and then go busy, indicated by a low level on the BUSY/READY line. This will generate a high level on I/O REQUEST, indicating that the byte transfer has been completed. The same signal also clocks the 74193 counters, advancing the PROM to the next byte location, and putting the next command byte on the data bus. When the CY512 has finished processing the last command byte, it will go ready again, generating another I/O REQUEST, and causing the CY512 to read the next command byte.

The above procedure continues until the PROM address reaches a value at which the data byte output is all bits high, 0FFH. This will generate a low output from the 7430, which will keep the CY512 READY signal from generating another I/O REQUEST. The circuit stops clocking at this point, and stays frozen with I/O REQUEST high and the 74193 counters set at the address which contains the 0FFH byte value. No more bytes will be transferred

until the address is changed by another load pulse to the 74193. This means that the user must end the program to be loaded into the CY512 with a byte containing the 0FFH. Note that the 0FFH is not read into the CY512, it is only used to stop the circuit from advancing any further. Since 0FFH is not a legal ASCII character, it may be used to end the program without fear that such a value might be part of the program, so long as the CY512 is operated in the ASCII mode. If the CY512 must be operated in the Binary mode, and the program to be loaded must contain an 0FFH data value, some other means of stopping the program must be found. In this case, the best approach would detect the end of program by a unique address from the 74193 counters. This would require the user to place the program in the PROM so that the last program byte occurs at the address just before this end of program address. Note that the same logic now used will work if the last address is 0FFH. In this case, the 7430 inputs connect to the 74193 outputs instead of the data bus. The last byte of the program should be at location 0FEH, one before 0FFH, since the byte at location 0FFH would not be read by the CY512. With this scheme, the starting address of the program would depend on the length of the program, and must be set properly before the load pulse is given to the 74193. The design shown in the schematic allows the starting address to be fixed, with the end indicated by the 0FFH data byte value.

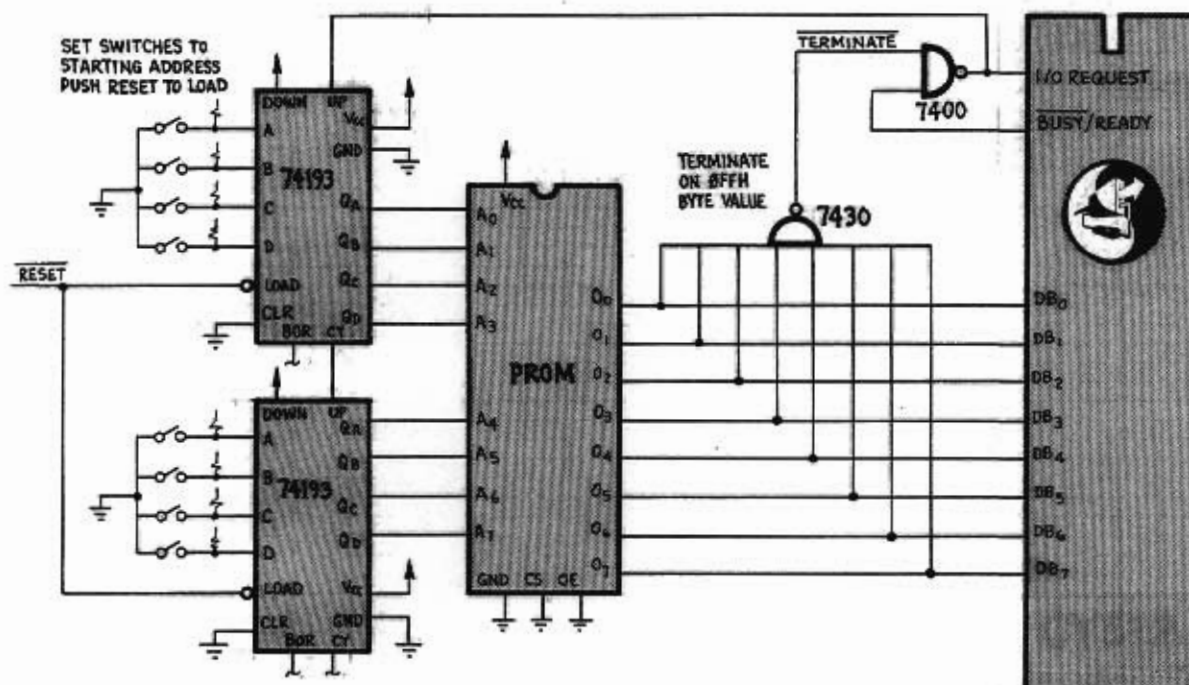


Figure 10.16 PROM Stand-alone Interface



# 11 COMPUTER CONTROL OF CY512 11

## COMPUTER CONTROL OF CY512

The ability to control all of the CY512 control inputs and monitor all of the CY512 outputs allows the designer to exercise the maximum control over the device. The following sections present information that may be used as a guide to interfacing the CY512 to a computer via the use of programmable I/O devices such as the Intel 8255. The programs are written for the 8080 microprocessor, but the general scheme will, of course, work with any computer using two parallel 8-bit output ports and one parallel 8-bit input port. For Verify mode, the data bus port may be bidirectional, or replaced by a tri-statable output port and another input port. The setup is as shown below:

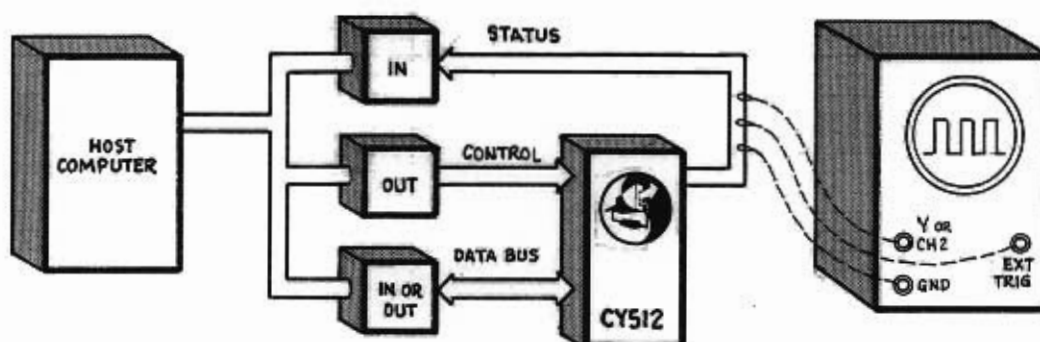


Figure 11.1 Example setup for Test/Display/Control of CY512 Stepper Controller.

By using a loop in the host computer (or in the CY512) the user can achieve a repetitive operation of the CY512 that allows easy display of CY512 signals on a standard oscilloscope. The use of externally triggered horizontal sweep circuits to synchronize the scope display is particularly convenient. The MOTION COMPLETE (INT REQ 1) output (CY512 pin 37) and the PROGRAMMABLE OUTPUT (pin 34) serve well as external triggers.

## ENTER/QUIT PROGRAMMING MODE

A key feature of the CY512 is the capability to accept and execute sequences of instructions; i.e., stored programs. The device powers-up in the "Command" mode of operation in which valid instructions are executed as they are received. If the ENTER command, "E", is given, the device initializes the relevant (internal) pointers and prepares to accept the program entered. All commands received prior to the receipt of the "Q" command are stored in the program buffer in the order in which they are received. Each command is entered just as in the command mode;

that is, the opcode is entered followed by either the "Linend" character "]" (carriage return) or a delimiter and parameter string terminated with the "]". The only command NOT terminated with a Linend (0DH) is the QUIT command, "Q"=51H. The Linend should not be used immediately following the "Q" character. The escape (QUIT) command terminates the program entry mode of operation, and returns the system to the command execution mode.

The maximum efficiency in use of the CY512 may be gained by presetting parameter values before entry and execution of the program. All parameter values have their own storage registers, so they need not occupy program buffer space, if the values stay constant during program execution. The host program may treat the CY512 program as a "Co-routine" that can be passed a set of parameters and invoked via the DOITNOW command. The host can then sample the RUN output (pin #32) or utilize this output in an interrupt mode to detect program completion and load new parameters or programs, as appropriate. This mode of operation is particularly well suited for inclusion in multi-tasking systems, when two or more CY512s are controlled by a single host.

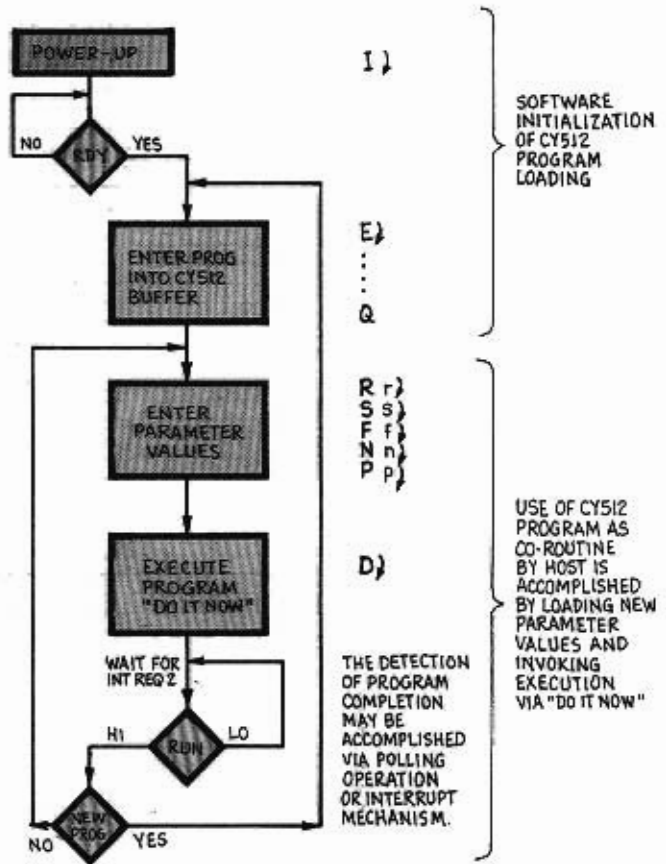


Fig. 11.2 CY512 used as "co-routine"

### CY512 STAND-ALONE APPLICATIONS

The CY512 receives data and commands from an 8-bit data bus. The source of data in most cases will be from an ASCII keyboard during prototype development and a microcomputer in the final system. The CY512, of course, does not know or care where the commands and data actually come from. This means that as long as the handshake protocol is properly implemented, the commands can be stored in a ROM, PROM, or EPROM and can be sequenced to control the CY512 with no host processor at all. For certain limited repertoire machines and stand-alone applications, this may be a very practical solution. A conceptual diagram of this type of system is shown in Figure 11.3. See also PROM Stand-Alone Interface Design in section 10.

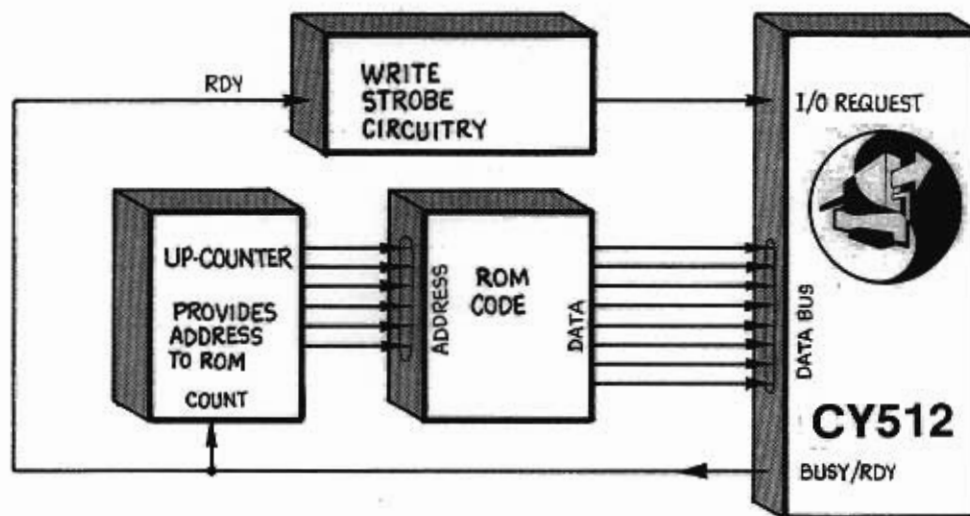


Figure 11.3 The CY512 can receive commands and data from a ROM sequencer for many stand-alone applications not requiring a host microcomputer.

## PROGRAMMING EXAMPLES

The following pages illustrate several programming examples, including waveforms and program listings. Programs are all written in 8080 Assembly Language, but the comments should allow those readers who are not familiar with the 8080 to understand what the various subroutines are doing. The programs were used on an SDK80 board, with the CY512 included in the wire wrap area.

We start with an equate table, indicating how the CY512 was connected to the SDK80 I/O signals. The names assigned to the various signals are used in the other routines. The table is followed by a Binary mode example, with the data buffer, BINBUFFER, showing the exact data bytes sent to the CY512 in this program. All bytes except the 0FFH at the end of the table are sent by the SENDPARALLEL program, which is shown next. This routine implements the basic data transfer between the SDK80 and the CY512, illustrating an example of the handshake protocol needed to transfer the bytes. It may be used in either Binary or ASCII mode. The ASCII mode example, which follows the SENDPARALLEL program, sends the same commands to the CY512 in ASCII mode as the Binary mode example shown previously, with ASCIIBUFFER containing the ASCII data bytes sent by this program. Finally, another Binary mode example is used to generate a repeating oscilloscope waveform.

TABLE IX

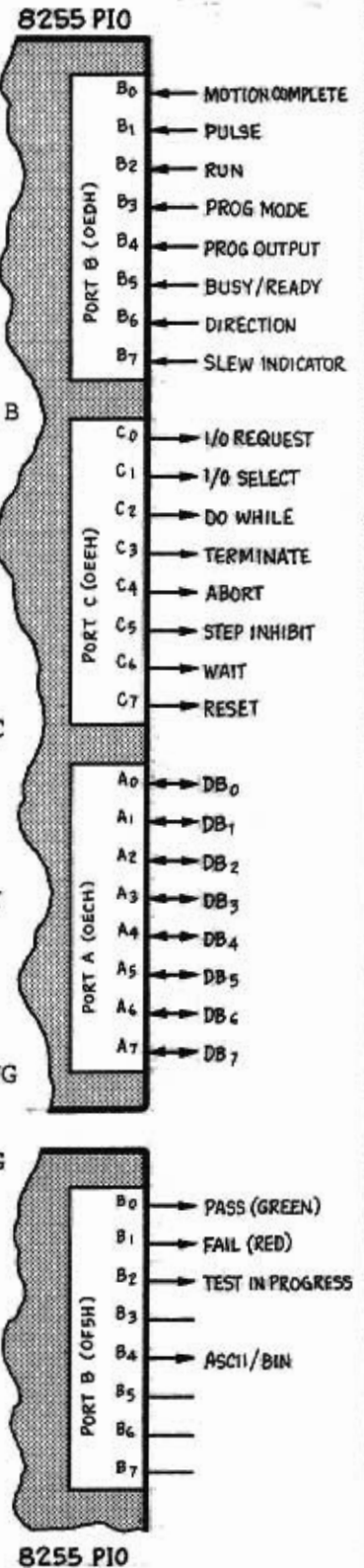
EQUATE TABLE AND 8255 PORT ASSIGNMENTS

The CY512 is connected to 8255-A4 ports 0ECH to 0EEH on the Intel SDK80 board.

```

;
; EQUATE TABLE
;
00F5 = LEDS      EQU 0F5H ;TESTING LEDS ON PORT B=F5H
;
0001 = GREEN    EQU 1     ;PASS
0002 = RED      EQU 2     ;FAIL
0004 = YELLOW   EQU 4     ;IN PROGRESS
;
00EC = DATA    EQU 0ECH  ;PORT A IS DATA BUS ON A4
;
00ED = STATUS   EQU 0EDH  ;CY512 STAT READBACK ON PORT B
;
0001 = MOTION   EQU 1     ;B0--MOTION COMPLETE
0002 = PULSE    EQU 2     ;B1--PULSE OUTPUT
0004 = RUNBAR   EQU 4     ;B2--RUN MODE PIN
0008 = PROGBAR  EQU 8     ;B3--PROGRAM MODE PIN
0010 = BITOUT   EQU 10H   ;B4--PROGRAMMABLE OUTPUT
0020 = READY    EQU 20H   ;B5--BUSY/READY PIN
0040 = DIRECT   EQU 40H   ;B6--DIRECTION INDICATOR PIN
0080 = SLEW     EQU 80H   ;B7--SLEW INDICATOR PIN
;
00EF = A4CNTRL  EQU 0EFH  ;CY512 INPUTS RUN FROM PORT C
;
0000 = IOREQ    EQU 0     ;C0--LOW I/O REQUEST
0001 = NOIOREQ EQU 1     ;C0--HIGH FOR NO REQUEST
;
0002 = IOSELIN  EQU 2     ;C1--LOW FOR COMMAND INPUT
0003 = IOSELOUT EQU 3     ;C1--I/O SELECT HI FOR VERIFY
;
0004 = DOWHILE  EQU 4     ;C2--LOW DOWHILE TO LOOP
0005 = NOLOOP   EQU 5     ;C2--HIGH FOR NO LOOP
;
0006 = TERMIN   EQU 6     ;C3--TERMINATE STEP IF LOW
0007 = NOTERM   EQU 7     ;C3--HI FOR NORMAL STEP TIMING
;
0008 = ABORT    EQU 8     ;C4--LOW FOR ABORT
0009 = NOABORT  EQU 9     ;C4--HIGH FOR NORMAL STEPPING
;
000A = TRIGGER  EQU 0AH   ;C5--LOW TO ALLOW STEPPING
000B = STPINH   EQU 0BH   ;C5--HIGH STEP INHIBIT
;
000C = LOWAIT   EQU 0CH   ;C6--LOW ON WAIT PIN
000D = HIWAIT   EQU 0DH   ;C7--HIGH ON WAIT PIN
;
000E = RESETLO  EQU 0EH   ;C7--HARDWARE RESET ON LOW
000F = NORESET  EQU 0FH   ;C7--HIGH TO RUN CY512
;
;
000D = CR       EQU 0DH   ;ASCII CARRIAGE RETURN CODE
;

```



## BINARY DATA PROGRAMMING EXAMPLE

The binary data mode is illustrated by the programs and timing diagrams that follow:

```

;
; TESTBINARY:
00A8 11C300    LXI    D,BINBUFFER
00AB CDD700    CALL   SENDPARALLEL
;
; RDYERROR:
00AE DBED     IN     STATUS
00B0 E620     ANI    READY
00B2 C2E803   JNZ    ERROR    ;FALSE READY
;
; TSTINTREQ1:
00B5 DBED     IN     STATUS
00B7 E601     ANI    MOTION
00B9 CAEE00   JZ     RDYERROR
;
;
00BC 3E01     MVI    A,GREEN
00BE D3F5     OUT    LEDES
00C0 C3A800   JMP    TESTBINARY
;
;
; BINBUFFER:
00C3 4300     DB    'C',0    ;CLEAR CY512 PIN 34
00C5 4200     DB    'B',0    ;SET PIN 34 HIGH
00C7 5201C8   DB    'R',1,200 ;SET RATE = 200
00CA 5301FE   DB    'S',1,254 ;SET SLOPE = 254
00CD 460101   DB    'F',1,1    ;SET FACTOR = 1
00D0 4E020500 DB    'N',2,5,0 ;SET 5 STEPS
00D4 4700     DB    'G',0    ;GO FOR 5 STEPS
00D6 FF       DB    0FFH   ;STOPPER
;

```

In the command mode, the BUSY/RDY output remains low after a GO command is received until the CY512 finishes the last of the "N" steps specified. This is indicated by the END-of-MOTION (INTREQ1) output (pin 37). The RDY line returns high approximately 30 microseconds after the INTREQ1 goes low. INTREQ1 rises when the next command is sent to the CY512.

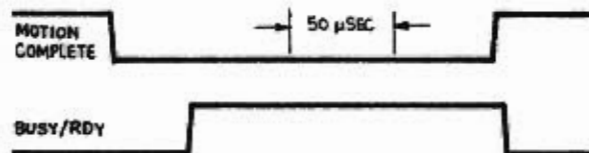
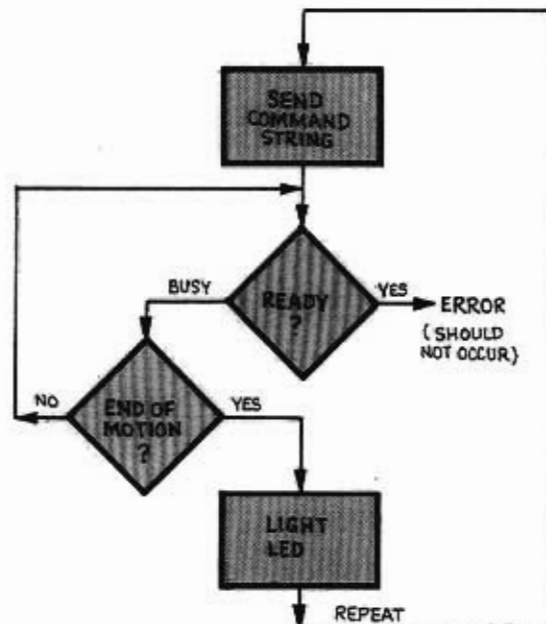


Figure 11.4 End-of-Motion Timing.



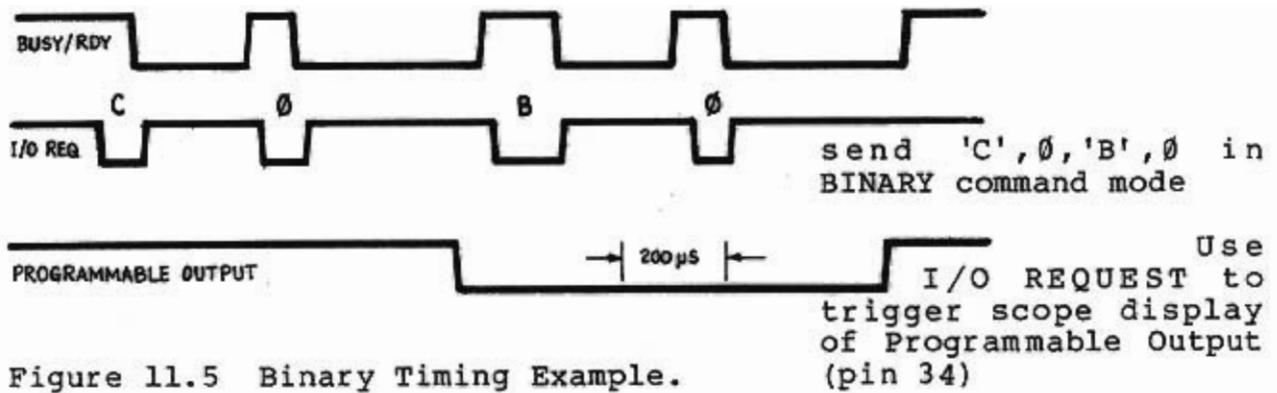


Figure 11.5 Binary Timing Example.

## HANDSHAKE SUBROUTINE

The parallel ASCII data is sent to the CY512 Stepper Motor Controller using the 8080 SENDPARALLEL code shown below. In this system the I/O REQUEST strobe is generated via a separate programmable control line and is removed after the data is acknowledged by the CY512.

```

;
SENDPARALLEL: ;ROUTINE TO SEND COMMAND BYTES TO CY512
;DE = POINTER TO BYTE STRING, 0FFH IS STOPPER
00D7 1A LDAX D ;GET NEXT BYTE FROM BUFFER
00D8 FEFF CPI 0FFH ;IS IT STOPPER?
00DA C8 RZ ;RETURN IF STOPPER, ALL BYTES SENT
00DB 13 INX D ;UPDATE POINTER
00DC 4F MOV C,A
00DD CDE300 CALL SENDCHAR ;SEND THIS BYTE TO CY512
00E0 C3D700 JMP SENDPARALLEL ;KEEP LOOPING TIL STOPPER
;
;
SENDCHAR: ;OUTPUT CHAR IN C TO CY512
00E3 DBED IN STATUS
00E5 E620 ANI READY ;LOW IF BUSY
00E7 CAE300 JZ SENDCHAR ;WAIT FOR READY
;
00EA 79 MOV A,C
00EB D3EC OUT DATA ;PUT CHAR ON DATA BUS
;
00ED 3E00 MVI A,IOREQ
00EF D3EF OUT A4CNTRL ;LOWER I/O REQUEST, DATA IS AVAIL
;
WAITBSY:
00F1 DBED IN STATUS
00F3 E620 ANI READY
00F5 C2F100 JNZ WAITBSY ;WAIT FOR BUSY (CY512 GOT DATA)
;
00F8 3E01 MVI A,NOIOREQ
00FA D3EF OUT A4CNTRL ;RAISE I/O REQUEST
00FC C9 RET
;

```

Figure 11.6 Example command output subroutine.

## ASCII DATA PROGRAMMING EXAMPLE

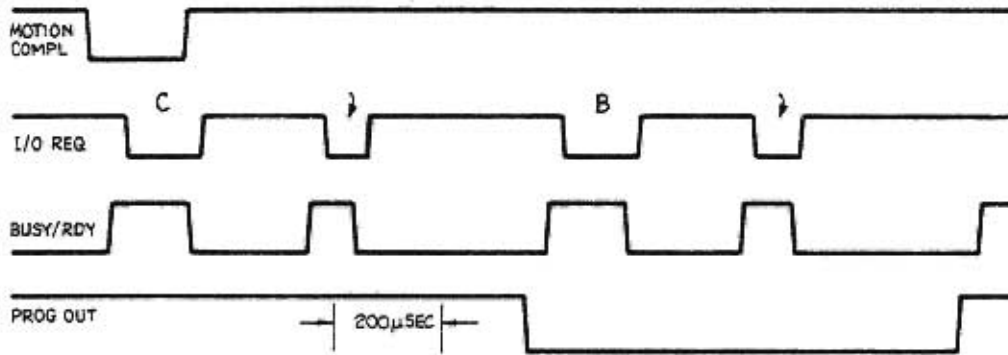


Figure 11.7 Expanded Handshake Timing Diagram.

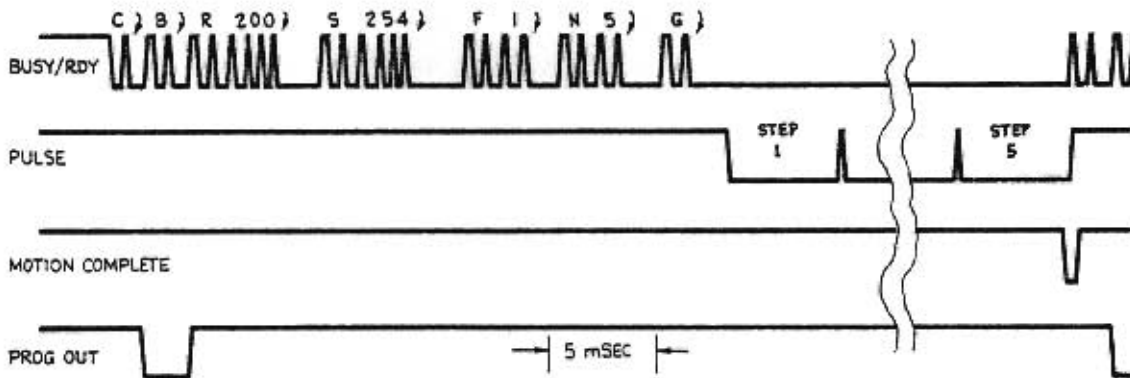


Figure 11.8 Complete Timing for Sample Program.

	;	TESTASCII:	;ASCII MODE
00FD 111801	LXI	D,ASCIIBUFFER	
0100 CDD700	CALL	SENDPARALLEL	;SEND THE COMMANDS IN THE BUFFER
	;		
	;	RDYLOOP:	
0103 DBED	IN	STATUS	
0105 E620	ANI	READY	
0107 C2E803	JNZ	ERROR	;SHOULD BE BUSY STEPPING
	;		
010A DBED	IN	STATUS	
010C E601	ANI	MOTION	
010E CA0301	JZ	RDYLOOP	;WAIT FOR MOTION COMPLETE
	;		
0111 3E01	MVI	A,GREEN	
0113 D3F5	OUT	LEDS	;LIGHT GREEN LED
	;		
0115 C3FD00	JMP	TESTASCII	;LOOP FOR SCOPE DISPLAY
	;		
	;	ASCIIBUFFER:	;COMMAND STRING FOR CY512
0118 430D	DB	'C',CR	;CLEAR PROGRAMMABLE OUTPUT (PIN 34)
011A 420D	DB	'B',CR	;PIN 34 HIGH
011C 52203230	DB	'R 200',CR	;SET RATE = 200
		300D	
0122 53203235	DB	'S 254',CR	;SET SLOPE = 254
		340D	
0128 4620310D	DB	'F 1',CR	;SET FACTOR = 1
012C 4E20350D	DB	'N 5',CR	;SET FOR 5 STEPS
0130 470D	DB	'G',CR	;GO FOR 5 STEPS
0132 FF	DB	0FFH	;STOPPER FOR SENDPARALLEL ROUTINE
	;		

Figure 11.9 Sample program.

## OSCILLOSCOPE DISPLAY EXAMPLE

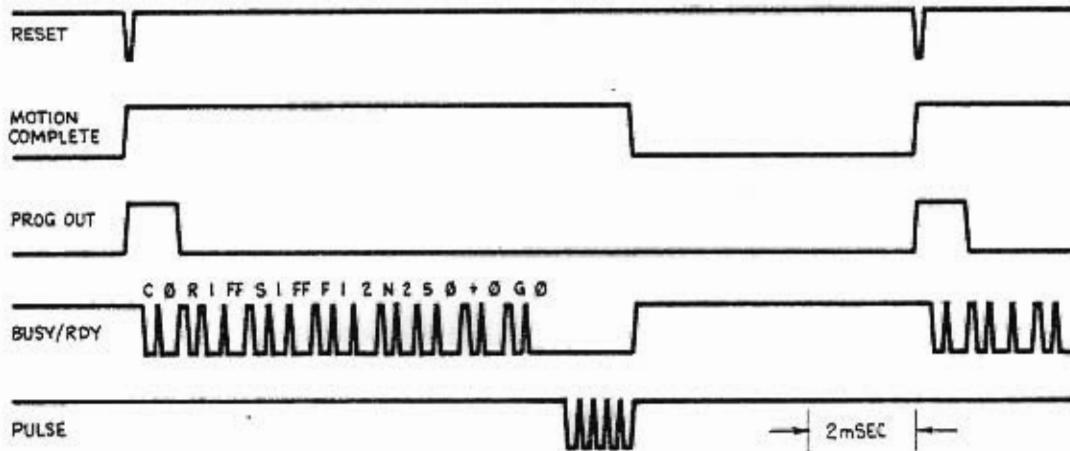


Figure 11.10 Timing for Program Shown Below.

The 8080 (or equivalent) sends the following commands and binary data to the CY512:

```

---      reset CY512 using pin 4
'C' 0    clear programmable output (pin 34)
          (used to trigger scope display)
'R' 1 FEH set rate parameter = 0FEH
'S' 1 FFH set slope parameter = 0FFH
'F' 1 2   set rate factor = 2
'N' 2 5 0 set number of steps = 5
+' 0     set CW direction (redundant)
'G' 0    begin stepping
  
```

After sending the above commands, the host computer polls the MOTION COMPLETE output (pin 37) and, upon finding it active, after the 5th step has been taken, the host delays a fixed time interval and then loops back, resets the CY512 and repeats this process. The programmable output may be used to trigger an oscilloscope.

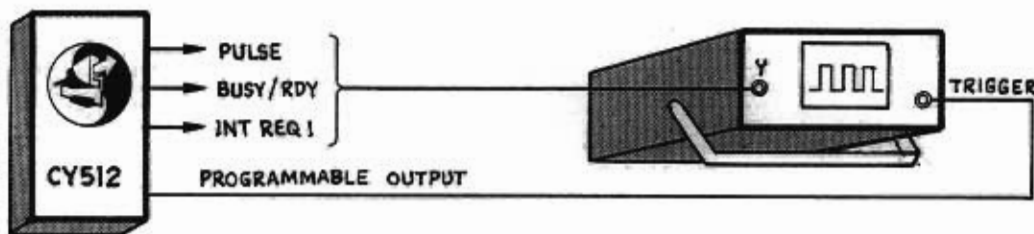


Figure 11.11 Test Setup.



## IEEE-488 INTERFACE

Using only a few SSI TTL gates, the CY512 can be made to work as a "LISTENER" on the IEEE-488 or GPIB (General Purpose Interface Bus). This section describes the timing and control involved in the GPIB interface and identifies the CY512 signal names with the appropriate GPIB signals. This implementation represents a simple GPIB interface. If a more complete bus interface is required, especially in a multi-instrument environment, the user should employ a separate GPIB interface device between the CY512 and the bus. This would allow the user to assign device addresses and communicate in both directions, using the CY512 Verify mode. A suitable GPIB interface device would be Fairchild's 96LS488.

## GPIB HANDSHAKE SIGNALS

The "TALKER", or device desiring to send 8 bits of data to the CY512 over the data bus, uses the DAV (Data AVailable) signal that corresponds to the I/O REQUEST line on the CY512. Before lowering the DAV line, the TALKER must test the NRFD (Not Ready For Data) line. This line corresponds to the CY512 BUSY/RDY line. When this line is low, the LISTENER (CY512) is Not Ready For Data. When the TALKER finds the NRFD line high, it can assert (lower) its DAV write line to the CY512. Thus far, the interface is identical to the standard CY512 handshake. The third handshaking signal is an acknowledge line from the listener named NDAC (Not Data ACcepted). This line must initially be low and is raised to indicate that the data has been accepted by the CY512. The NDAC line is tested by the TALKER to determine whether or not the LISTENER has accepted the data. The CY512 BUSY/RDY line actually acknowledges the data transfer by going low, thus by inverting the RDY line, an NDAC signal can be generated. This completes the three line handshake required for the GPIB.

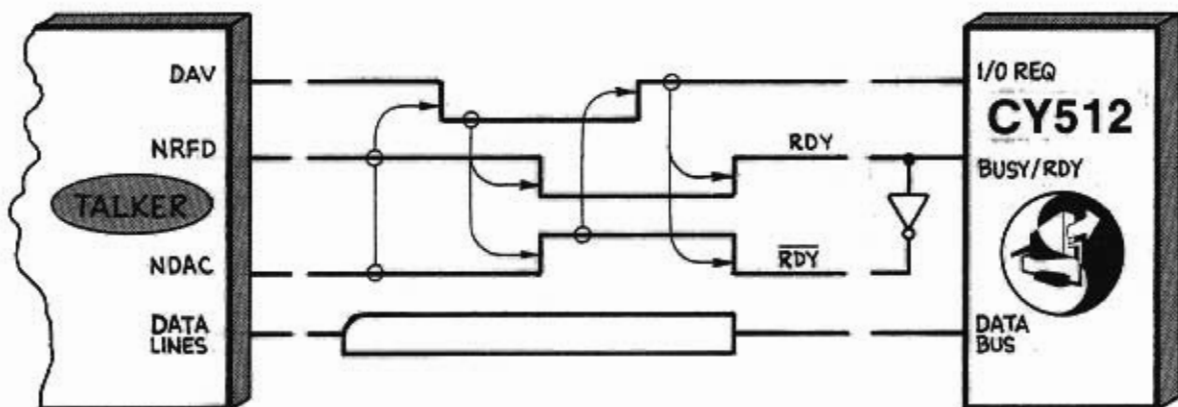


Figure 12.1 CY512/GPIB Interface.

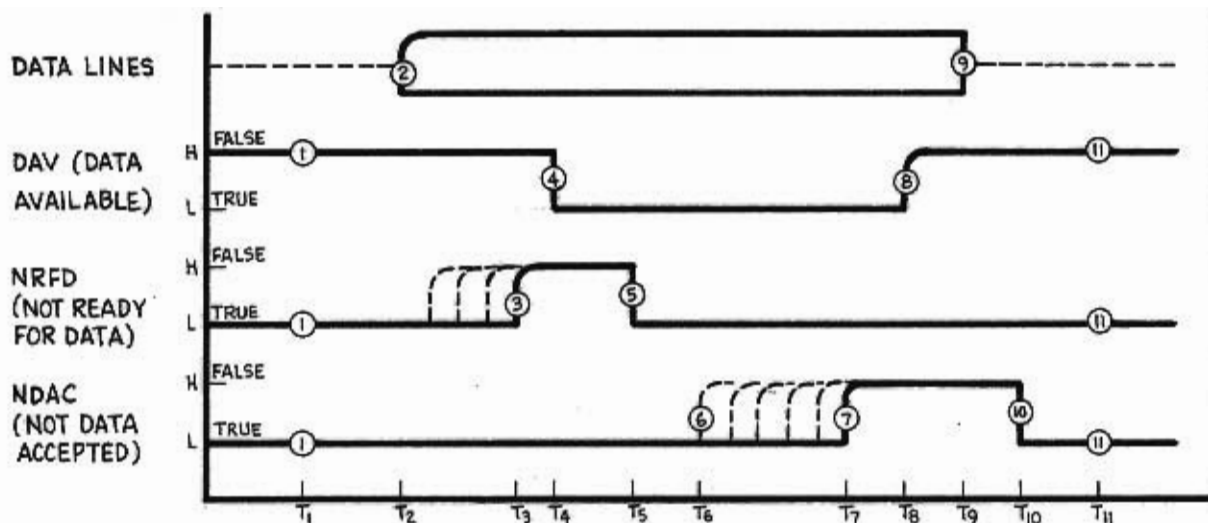


Figure 12.2 GPIB Handshake Signals.

The flowchart for the TALKER that controls the CY512 is shown in Figure 12.3. This procedure can be implemented simply using any microprocessor and describes the manner in which most GPIB interface devices function.

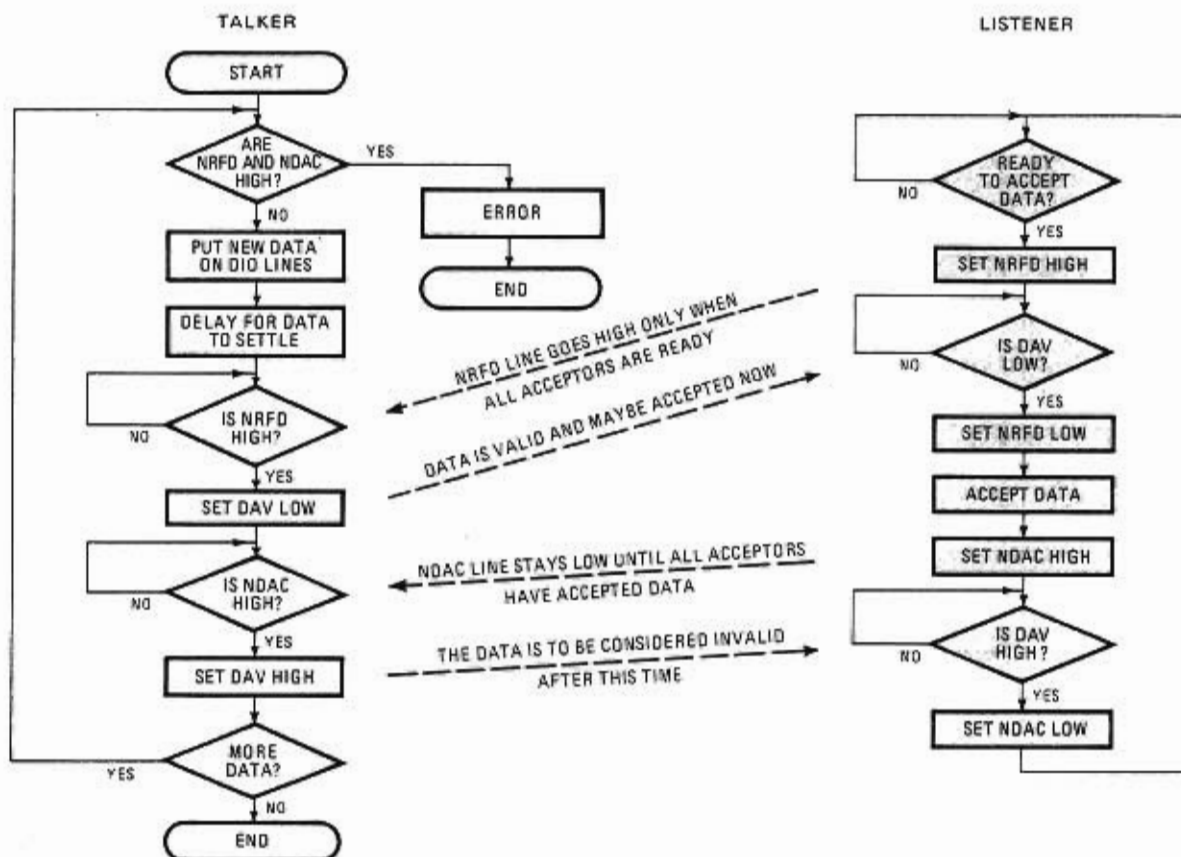


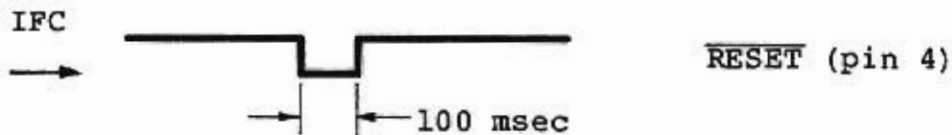
Figure 12.3 TALKER/LISTENER handshaking procedure.

## GPIB INTERFACE MANAGEMENT SIGNALS

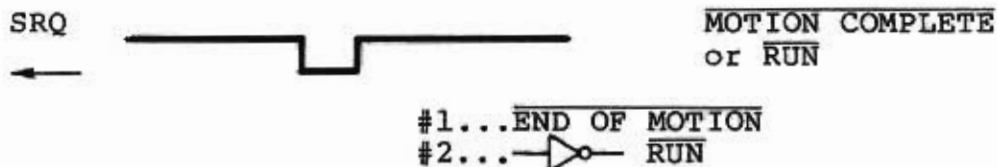
In addition to the three line handshake, there are several other control lines defined by the IEEE-488 interface specifications. These are described below and identified with appropriate CY512 signal lines.

IEEE-488

CY512



Interface Clear goes low after power on. This line is used to reset the CY512 and can replace the power on startup circuitry.



Service Request is used to inform the TALKER that the LISTENER (CY512) has completed an action and is ready for more commands.



The Attention line is used to signify that the data on the bus is a device address. For multiple CY512s this may be used for selection. The ATN line should inhibit the CY512 I/O REQUEST line. Note that ATN may also be used to prevent the CY512 from seeing line feeds (0AH) sent after linends (0DH) as is done by many BASIC language controllers. ATN may also be used to inhibit other interface commands to which the CY512 cannot respond.

IEEE-488 TALKER SENDS  
COMMANDS TO CY512

CY512 STEPPER MOTOR  
CONTROLLER RECEIVES  
COMMANDS

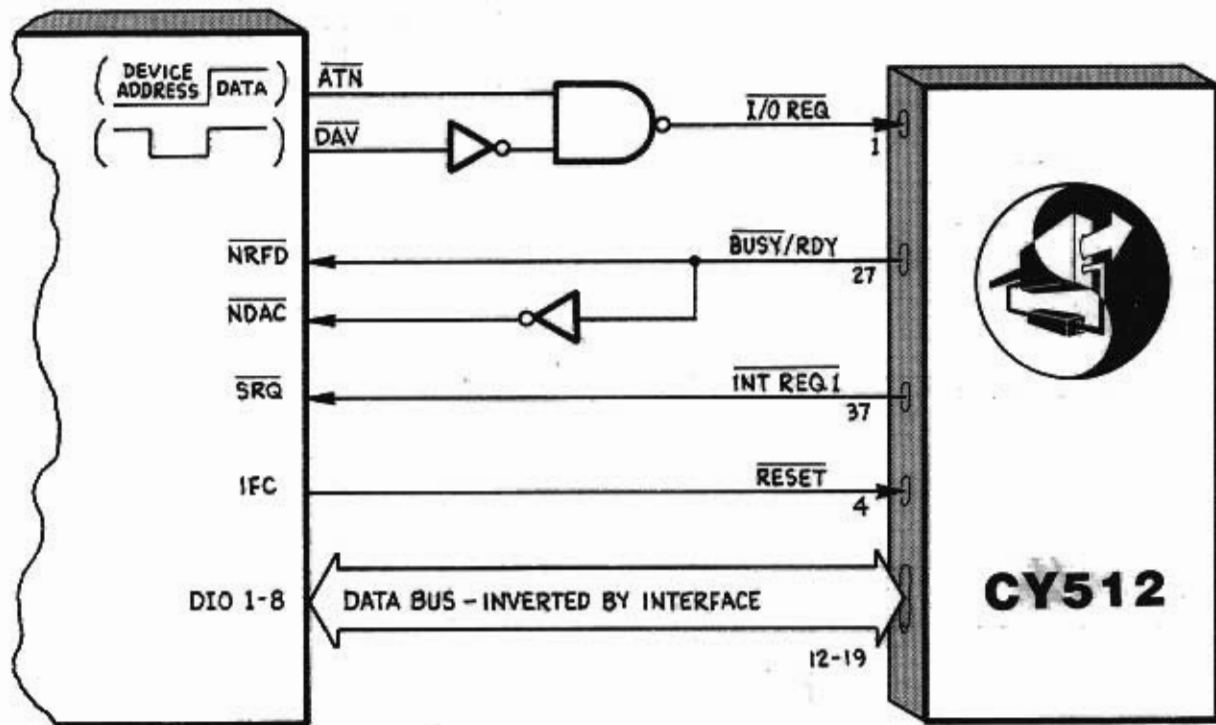


Figure 12.4 Simple IEEE-488/CY512 Interface.

In some systems the REN (Remote ENable) and EOI (End Or Identify) IEEE-488 control signals may be useful. For further information on the IEEE-488 interface the reader is referred to the following references:

IEEE STANDARD 488 - 1978  
available from  
IEEE Service Center  
445 Hoes Lane  
Piscataway NJ 08854 USA

PET and the IEEE 488 BUS  
by Fisher and Jensen, 1980  
Osborne/McGraw Hill  
630 Bancroft Way  
Berkeley CA 94710 USA

## GPIB SCHEMATIC EXAMPLE

The following pages illustrate the logic used in an actual project which connected the CY512 to the IEEE-488 bus, using the Fairchild 96LS488. The schematics indicate data flow, handshake control logic for bidirectional data transfers, and interrupt logic to control stepping and detect when a limit has been reached. Such a design supports many functions of the GPIB, and allows several CY512s or other GPIB instruments to reside on the same bus. This design is included with the permission of Christopher R. Hansen of the Mayo Foundation.

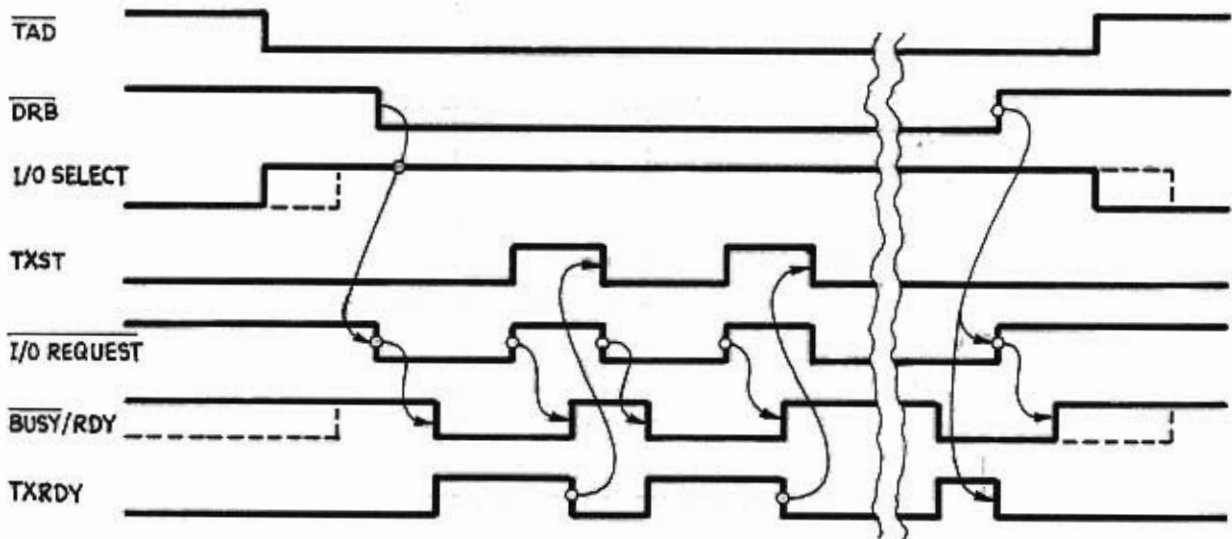


Figure 12.5 Timing for Talker addressed.

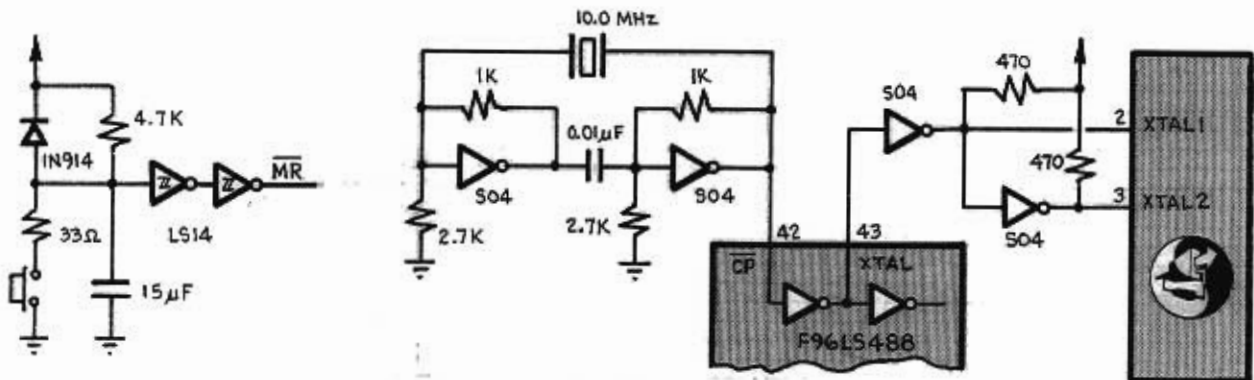


Figure 12.6 Clock and Reset Logic

The figure above shows the implementation of the clock inputs for both the CY512 and the 96LS488 from a single crystal, and a manual reset. It also shows the timing for the handshake circuits when the CY512 is asked to output its values from a Verify command.

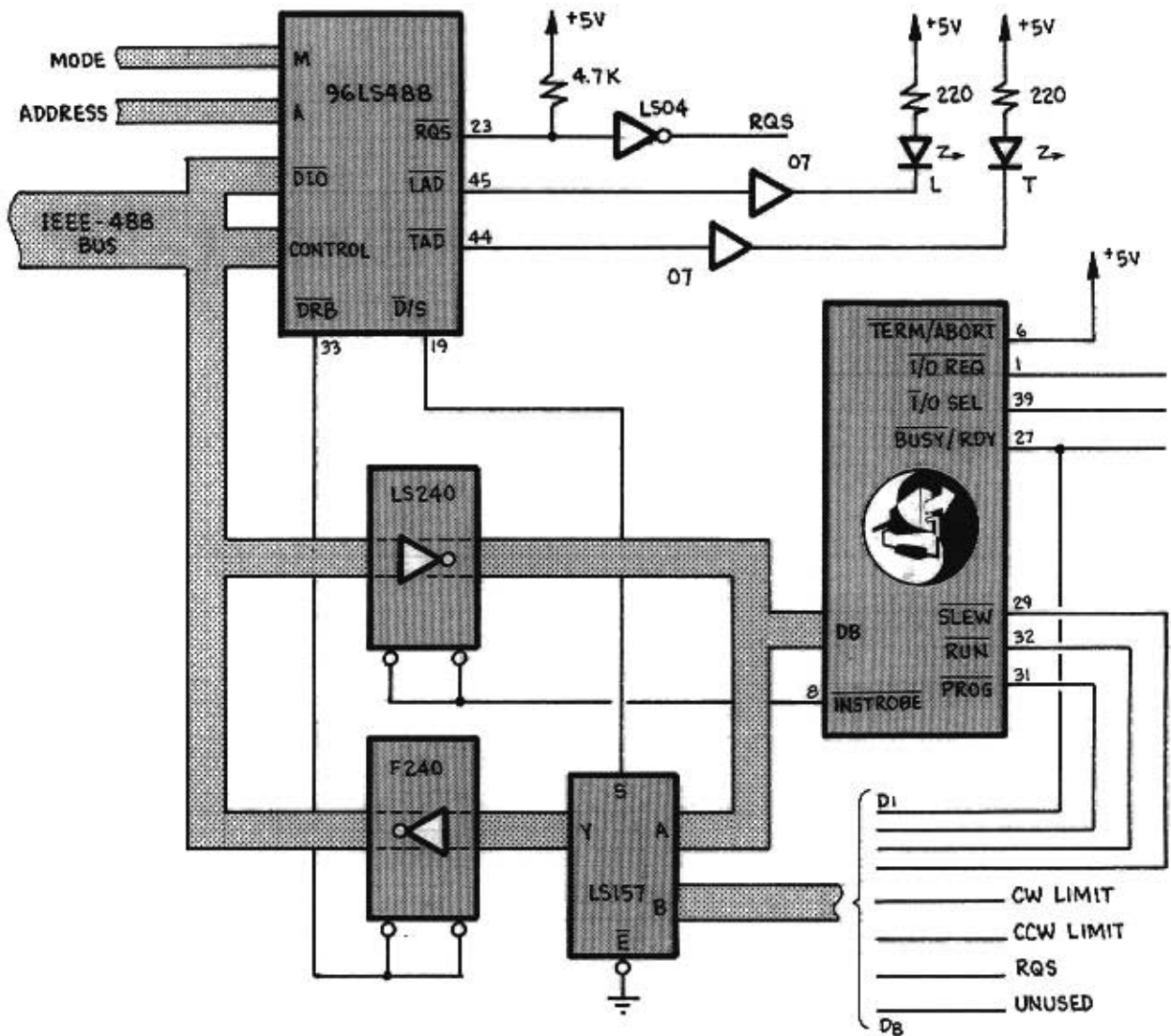


Figure 12.7 Data Paths

The data path schematic illustrates general data flow between the GPIB, the 96LS488, and the CY512. All control signals from the GPIB connect directly to the 96LS488, which interprets the bus commands and controls the handshake logic to the CY512. The eight data lines connect to the 96LS488, and through 74LS240 buffers to the CY512. The buffers invert the data signals between the CY512, which uses positive logic, and the GPIB, which uses negative logic. Two modes are used to read back information from the CY512. To read the internal parameters, using the Verify mode, a normal GPIB read operation is performed, by making the 96LS488 and CY512 the bus talker. To read back the states of various CY512 control lines, the 96LS488 is asked to perform a status read as a result of a poll command. Note that any eight of the CY512 signals may be connected to the status port. Internal CY512 data and the status information are multiplexed by the 74LS157.

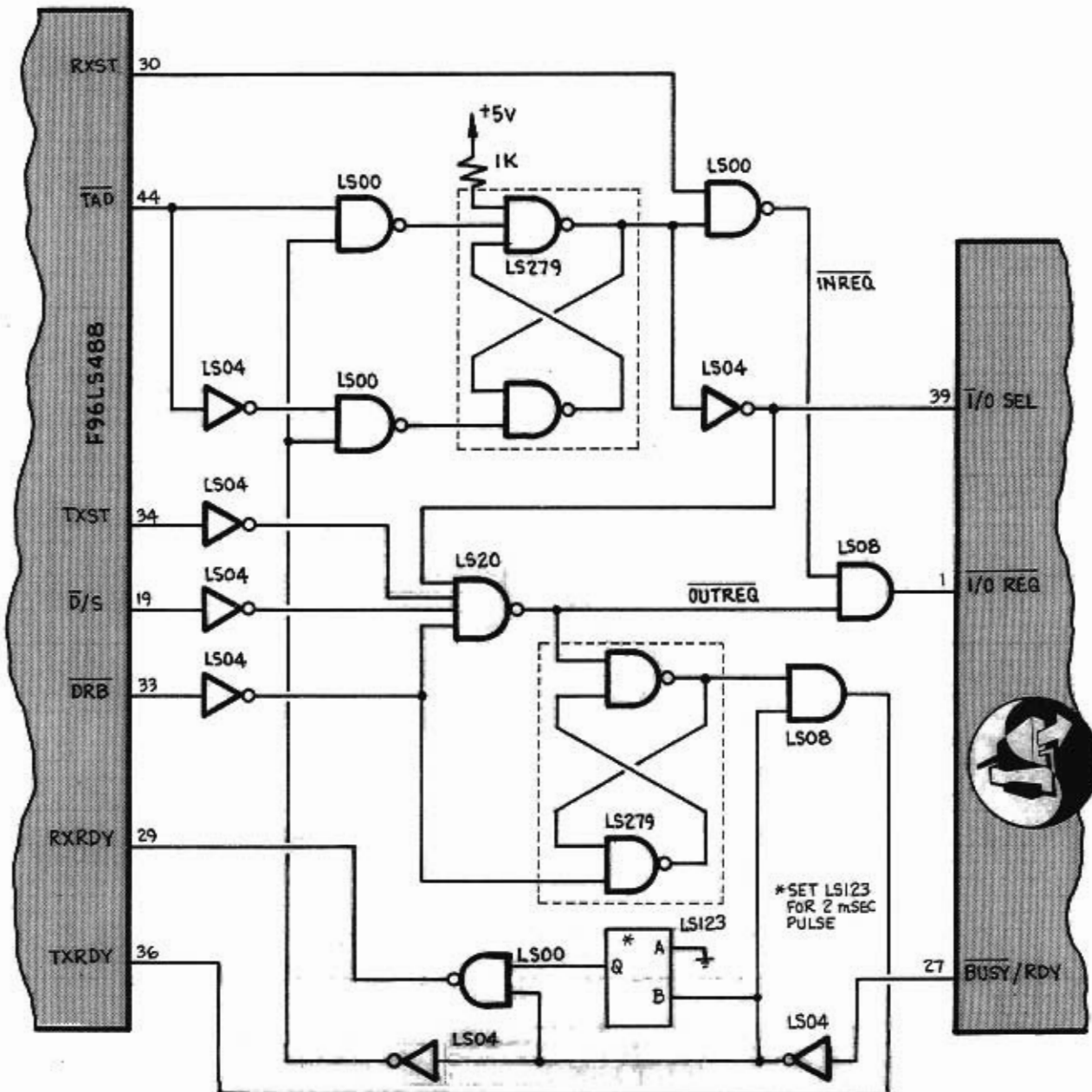


Figure 12.8 Handshake Control Logic

The handshake control logic is the key to connecting the CY512 to the 96LS488. This logic converts the appropriate signals between the simple handshake of the CY512, and the more complex handshake performed by the 96LS488. By combining the various signals from the 96LS488, the proper values for I/O Select and I/O Request are generated. Much of this logic distinguishes between the Listen mode (sending commands to the CY512) and the Talk mode (sending data from the CY512). This enables the Verify command to be used with the GPIB design. The CY512 Busy/Ready signal is used to complete the handshake between the CY512 and the 96LS488.

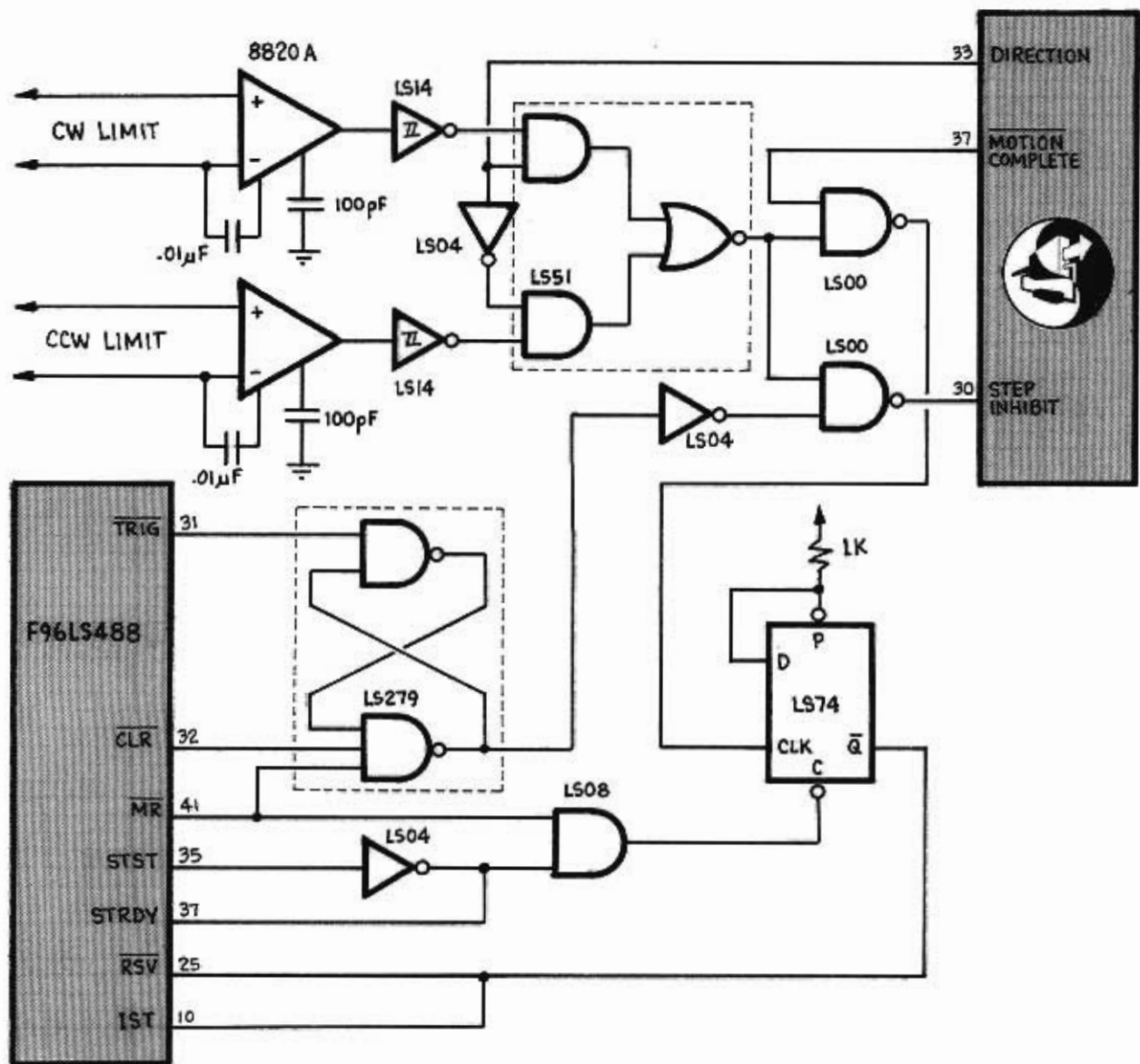



Figure 12.9 Interrupt Logic

The interrupt logic illustrates two functions. In the first, the GPIB may be used to control the start of stepping. The 96LS488 Trig, CLR, and MR signals are combined and connected to the CY512 Step Inhibit. This allows the GPIB master controller to stop any stepping operations by issuing a Device Clear command. Stepping may also be synchronized to other events by allowing the master to start the stepping using a Device Trigger command. The second function of the interrupt logic is to monitor the stepping operation, and warn the master controller when a limit has been reached. By monitoring the limits, a Service Request can be generated when a limit is reached. The Step Inhibit is also controlled by this process, keeping the CY512 from inadvertently stepping too far. Service Requests are also generated by the CY512 Motion Complete signal.



# 13 GETTING YOUR CY512 RUNNING 13

The following checklist will simplify getting your CY512 up and running.

1. Connect pins 7 & 20 to ground and pins 26 & 40 to +5 volts.
2. Be sure that pin 39 is low and pin 6 is high.
3. Set pin 36 high (ASCII mode), set pin 30 low for now.
4. Be sure RESET (pin 4) is low for at least 10 milliseconds after power stabilizes. The CY512 can be reset at any time.
5. Upon proper reset all outputs should be at logic 1 (>3V).
6. Observe the RDY line (pin 27) to be sure it is high.
7. Observe CLK/15 (pin 11  400 KHz with 6 MHz Xtal).
8. Place the "CLEARBIT" command "C" (=43H) on the data bus.

```
DB0 = 1
DB1 = 1
DB2 = 0
DB3 = 0
DB4 = 0
DB5 = 0
DB6 = 1
DB7 = 0
```

9. Lower the I/O REQUEST line (pin 1).
10. Wait for RDY (pin 27) to go low before bringing I/O REQUEST high. If using I/O REQUEST strobe circuitry that generates a low write signal when an ASCII character is placed on the bus, be sure that your software detects low RDY line (Busy) before looking for High RDY. If you are using a debounced keyboard this should not be a problem.
11. When I/O REQUEST is brought back high, RDY will return high.
12. Wait for RDY to return high before placing the RETURN code (␣=0DH) on the data bus.

```
pin 12... DB0 = 1
           DB1 = 0
           DB2 = 1
           DB3 = 1
           DB4 = 0
           DB5 = 0
           DB6 = 0
pin 19... DB7 = 0
```

13. Generate the low I/O REQUEST strobe until RDY goes low, then return I/O REQUEST high, as before.
14. Upon completion of the above sequences of steps, the Programmable Output (pin 34) will go low.
15. Repeat steps 8 through 13, replacing "C" (=43H) with "B" (=42H). This "BITSET" command will cause the Programmable Output (pin 34) to return high. All other outputs (except RDY) should have remained high during the above procedure.
16. Repeat steps 8 through 13, replacing "C" (=43H) with "-" (=2DH). This is the "CCW" command. The result of this command will be to bring the DIRECTION Line (pin 33) low.
17. Following the CCW command with a CW command ("+" =2BH) will again raise the DIRECTION line.
18. If you have reached this point successfully you should be able to enter any of the commands and obtain the correct responses.
19. Suggested sequences:
  - a. enter "E)" followed by "Q" and observe the PROG (pin 31) go low with "E)" and return high with "Q".
  - b. with STEP INHIBIT (pin 30) low, enter "A)" followed by "P 1)". The STEPPER MOTOR DRIVE SIGNALS, pins 21-24, will be activated, and PULSE, pin 35, will go low and return high, indicating the duration of the step. The drive signals will change from step to step as the above sequence is repeated.
  - c. raise the STEP INHIBIT line and enter the single step sequence as in "b" above. Nothing will happen on PULSE, or the stepper control lines, until the STEP INHIBIT line is lowered.
  - d. refer to Figures 10.12, 10.13, and 10.14. Enter these commands as listed and observe the outputs. Note that LEDs on the relevant outputs are very useful.
20. After initial checkout is accomplished using ASCII input, the user may place pin 36 low to select Binary. Read the manual carefully for differences in the two modes.

# INDEX

## A

Abort 13,15,29,43,44,46  
Absolute 9,10,26,31,42  
--see Position command  
Acceleration 10,19,23,  
52-58  
Architecture 9  
ASCII 16-18,31,36,85  
Athome 10,19,20,26

## B

Binary 16,17,31,32,36,83  
Bipolar Driver 66  
Bitset 19,20,41  
Buffer 10,29,30,33-36  
Busy/Ready 11,12,16,40,68  
Bytes 34

## C

Chip Select 70  
Circuit 11,26,44,58,60,  
62, 64-66,69,70,76,78,  
91-94  
Clearbit 19,20  
Clock 16,62  
Clockwise 9,19,24,42  
Closed Loop 59  
Command Mode 19,25,31,33,  
38  
Commands 17-20,34  
Computer 39,79,82-85  
CoRoutine 80  
Counter 9,81  
Counterclockwise 9,19,25,  
42  
Crystal 16,50,61

## D

Data Bus 10,13,15  
Data Count 32  
Deceleration --see  
Acceleration  
Demonstration System 11,  
63,64  
Direction 10,12,15,42  
Doitnow 19,20,26  
Dowhile 16,27,28  
Driver 11,12,65

## E

Electrical 61  
Encoder 60  
Enter 19,20,79  
Equate Table 82  
Execute --see Doitnow  
and Go commands  
Expend 19,24  
External Control 26-28,  
58,60,62,69

## F

Factor 19,21,51-53,58  
Five-phase motor 65  
Full Step 31,47

## G

Go 10,19,21  
GPIB 87-96

## H

Halfstep 19,21,31,47  
Handshake 11,12,40,68,  
84,87  
Home --see Athome

## I

I/O Request 11,12,15,40,  
68-90  
I/O Select 12,15,40  
IEEE-488 Bus 87-96  
Initialize 19,21,26,62  
Input 9,13,16  
Instrobe 16,40  
Instructions 17-19,26,  
--see also Commands  
Interface 35,36,63,64,  
87,89

## J

Jump 19,21,26,29

## K

Keyboard 19,21,26,29  
Kit 63

## L

Language 17  
Latch 26  
Logic 61  
Loop 19,22,26,27,28,30

## M

Modes 10,31  
Motion Complete 14,15,54  
Motor Interface 12,26,  
63-65

## N

Number Command 10,19,  
22,42

## O

Offset 19,22,47,48  
Operational Modes 31  
Oscilloscope 86  
Output 9,13,16  
Outstroke 16,40

## P

Parallel 38  
Parameter 18,34  
Phase 12,15,47,48,65  
Pins 14-16  
Position 10,19,22,26,42  
Power 61  
Prog Pin 16  
Program Mode 27-31,33  
Program Storage --see  
buffer  
Programmable output 14,  
16,41 --see also Bitset  
and Clearbit  
Programs 27-30,33,59,  
71,72,74  
PROM 77,80  
Protocol 11,12  
Prototype system 11,63,64  
Pulse 12,15,26,43

## Q

Quit 19,23,79  
Query --see Verify

## R

Ramp --see Acceleration  
Rate 19,23,49,51,53  
Rate Equation 49  
Rate Table 50  
Read-Out 37 --see Verify  
Ready/Busy 11,12,16,40,68  
Registers 35  
Relative Position 9,10,  
31,42 --see also Number  
command  
Reset 15,26,62  
RS232 75  
Run 14,16,26,41

## S

Schematics --see circuits  
Serial 75  
Signals 61  
Slew 14,16,52,54,58  
Slope 19,23,52-54 --see  
also Acceleration  
Stand-alone 77,80  
Step Inhibit 13,16,31,  
42-44  
Step Rate --see Rate  
Step Timing 43,47  
Stop 45  
Storage --see Buffer  
Synchronization 12,70,71

## T

Temperature 61  
Terminate 13,15,43  
Three-phase motor 65  
Til 19,23,27  
Timing 41-44,46,54,55,68,  
70,72,73,83-86,91  
Translator Driver 65  
Trigger --see Step  
Inhibit  
Tri-State 40,61  
Troubleshooting 95

## U

Uart 76  
Unipolar Driver 66  
Until 19,24,37

## V

Verify 19,24,37

## W

Wait 13,15,19,24,28  
Waveforms, motor 12,47,  
48,73  
Write strobe 68

## X

X --see Expend  
Xtal 16,50,61

## Z

Zero 33,38 --see also  
Command Mode

# ASCII-DECIMAL TO HEX CONVERSION TABLE

DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	DEC	HEX	ASCII	HEX
0	00	51	33	102	66	153	99	204	CC	CR	0D
1	01	52	34	103	67	154	9A	205	CD	SP	20
2	02	53	35	104	68	155	9B	206	CE	+	2B
3	03	54	36	105	69	156	9C	207	CF	,	2C
4	04	55	37	106	6A	157	9D	208	D0	-	2D
5	05	56	38	107	6B	158	9E	209	D1		
6	06	57	39	108	6C	159	9F	210	D2	0	30
7	07	58	3A	109	6D	160	A0	211	D3	1	31
8	08	59	3B	110	6E	161	A1	212	D4	2	32
9	09	60	3C	111	6F	162	A2	213	D5	3	33
10	0A	61	3D	112	70	163	A3	214	D6	4	34
11	0B	62	3E	113	71	164	A4	215	D7	5	35
12	0C	63	3F	114	72	165	A5	216	D8	6	36
13	0D	64	40	115	73	166	A6	217	D9	7	37
14	0E	65	41	116	74	167	A7	218	DA	8	38
15	0F	66	42	117	75	168	A8	219	DB	9	39
16	10	67	43	118	76	169	A9	220	DC		
17	11	68	44	119	77	170	AA	221	DD	A	41
18	12	69	45	120	78	171	AB	222	DE	B	42
19	13	70	46	121	79	172	AC	223	DF	C	43
20	14	71	47	122	7A	173	AD	224	E0	D	44
21	15	72	48	123	7B	174	AE	225	E1	E	45
22	16	73	49	124	7C	175	AF	226	E2		
23	17	74	4A	125	7D	176	B0	227	E3	F	46
24	18	75	4B	126	7E	177	B1	228	E4	G	47
25	19	76	4C	127	7F	178	B2	229	E5	H	48
26	1A	77	4D	128	80	179	B3	230	E6	I	49
27	1B	78	4E	129	81	180	B4	231	E7	J	4A
28	1C	79	4F	130	82	181	B5	232	E8		
29	1D	80	50	131	83	182	B6	233	E9	K	4B
30	1E	81	51	132	84	183	B7	234	EA	L	4C
31	1F	82	52	133	85	184	B8	235	EB	M	4D
32	20	83	53	134	86	185	B9	236	EC	N	4E
33	21	84	54	135	87	186	BA	237	ED	O	4F
34	22	85	55	136	88	187	BB	238	EE		
35	23	86	56	137	89	188	BC	239	EF	P	50
36	24	87	57	138	8A	189	BD	240	F0	Q	51
37	25	88	58	139	8B	190	BE	241	F1	R	52
38	26	89	59	140	8C	191	BF	242	F2	S	53
39	27	90	5A	141	8D	192	C0	243	F3	T	54
40	28	91	5B	142	8E	193	C1	244	F4		
41	29	92	5C	143	8F	194	C2	245	F5	U	55
42	2A	93	5D	144	90	195	C3	246	F6	V	56
43	2B	94	5E	145	91	196	C4	247	F7	W	57
44	2C	95	5F	146	92	197	C5	248	F8	X	58
45	2D	96	60	147	93	198	C6	249	F9	Y	59
46	2E	97	61	148	94	199	C7	250	FA	Z	5A
47	2F	98	62	149	95	200	C8	251	FB		
48	30	99	63	150	96	201	C9	252	FC		
49	31	100	64	151	97	202	CA	253	FD		
50	32	101	65	152	98	203	CB	254	FE		
								255	FF		

## CYB-002 MULTI-PURPOSE CONTROL BOARD

A general purpose prototyping board is available which will allow the user to easily interface his computer, keyboard, or CRT to his control application. The CYB-002 board comes ready to assemble as a kit, with the capability of accepting any two Cybernetic Micro Systems control chips in any combination. Thus the board can become a dual axis stepper controller, waveform synthesizer, programmable controller, printer controller, data acquisition controller, and the like, with very little additional effort. Support software will also be available soon.

The core of the CYB-002 is Cybernetic's new Local System Controller, the CY250, which accepts ASCII commands, and addresses either of the two target chips via a pass-through mode, or accepts the data as direct commands to its own program buffer. Since the CYB-002 is wired to accept an optional EEPROM, then once programmed, it may also operate as an independent system. The board has additional circuitry for an optional LCD display and CY300 display controller, and for a network mode via the CY232. The CY232 will give the user the option of stringing boards together in a network with each having the ability to address up to 256 devices.

User definable switches and LEDs are available for various input and output signals, and an additional wire-wrapping area allows the user to customize the board to his particular application--in the case of the CY512, this could include the motor driver circuitry. While the board was designed as a prototyping aid for implementing the CYxxx family of chips, many users find that it is the ideal solution to their control problems. The CYB-002 is available with a variety of options: Display with CY300, Network with CY232, Memory with EEPROM, Keyboard, and Target with any CYxxx, as shown in the figure below:

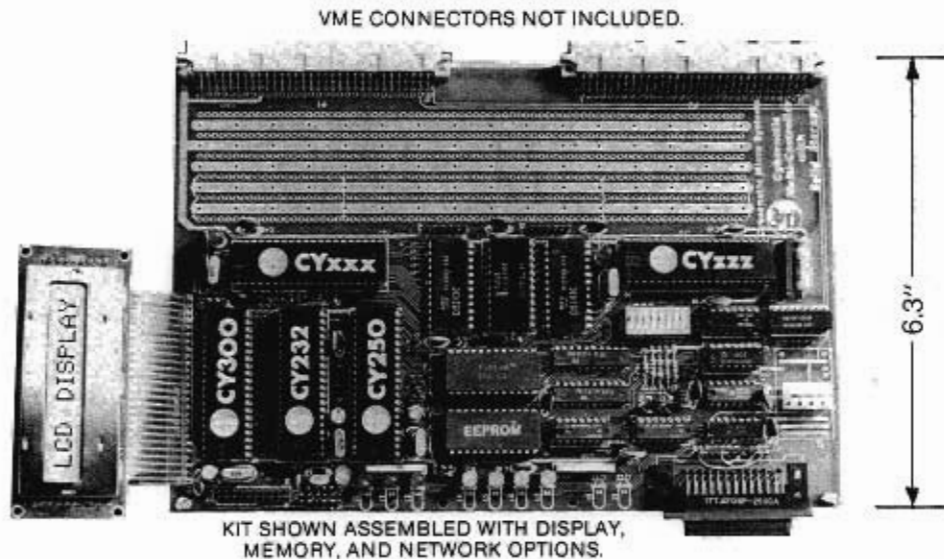


Figure 10.xx CYB-002 Multi-purpose Control Board

ADDENDA

## EEPROM STAND-ALONE INTERFACE DESIGN

The CY250 Local System Controller will allow the user to interface the CY512 to an EEPROM for easy storage of often used programs and for a stand alone system. The CY250 accepts serial or parallel commands and can address either of two CY512s via a pass-through mode, or accepts data as direct commands to its own program buffer. Alternately, the command sequences may be defined once and sent to the EEPROM, where the various command sequences are stored as named procedures, with the CY250 taking care of the EEPROM operation, space allocation, and name directory. This allows frequently used programs to be remembered by name and recalled whenever they are needed. For stand-alone operation, the CY250 has an "auto recall" feature which calls a specified routine from the EEPROM on power up or reset. This EEPROM interface has been implemented on the CYB-002 board shown in figure 10.xx. More details on the EEPROM interface may be found in the CY250 manual and the CYB-002 manual.

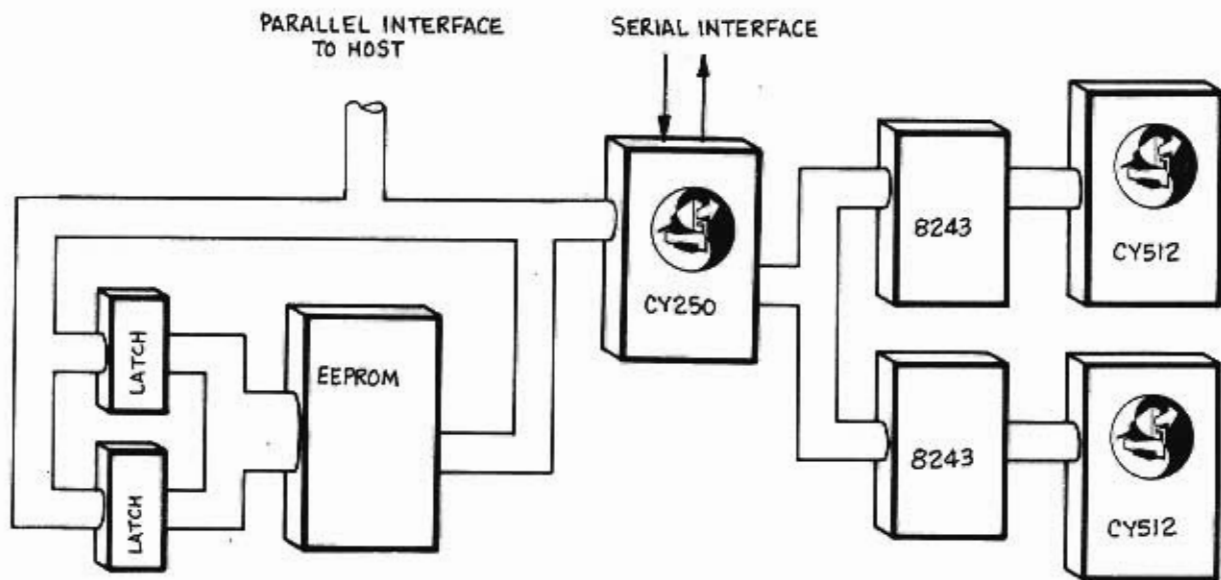


Figure 10.zz CY512 interface to EEPROM through CY250.