CYBERNETIC • MICRO SYSTEMS •

CY500

STORED PROGRAM STEPPER MOTOR CONTROLLER

This manual contains advance

product information of which certain

details are subject to change.

Cybernetic Micro Systems, Inc. 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 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 right 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

Cybernetic Micro Systems, Inc.
Box 3000 • San Gregorio CA 94074 USA
Tel: 650-726-3000 • Fax: 650-726-3003
www.ControlChips.com
info@ControlChips.com

CY500

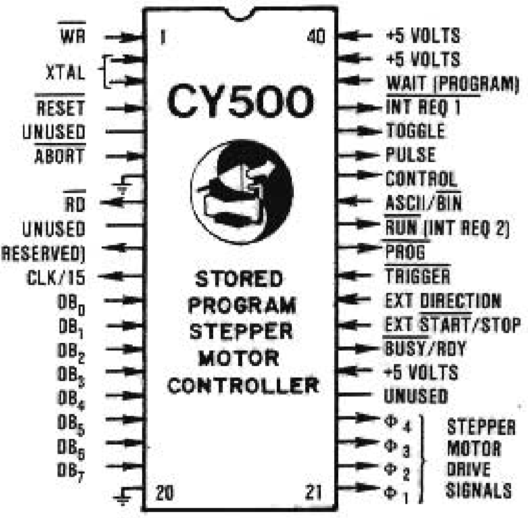
STORED PROGRAM STEPPER MOTOR CONTROLLER

The CY500 stored program stepper motor controller is a standard 5 volt, 40 pin LSI device configured to control any 4-phase stepper motor. The CY500 will interface to any computer using asynchronous parallel TTL input and provides numerous TTL inputs and outputs for auxiliary control and interfacing. The CY500 allows programming with an ASCII keyboard for prototype development and 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.

STANDARD FEATURES

- ASCII-DECIMAL OR BINARY COMMUNICATION
- SINGLE 5 VOLT POWER SUPPLY
- HI-LEVEL LANGUAGE COMMANDS
- STORED PROGRAM CAPABILITY
- HALF-STEP/FULL-STEP CAPABILITY
- ABSOLUTE/RELATIVE POSITION MODES
- PROGRAMMABLE VIA ASCII KEYBOARD
- 3300+ STEPS PER SECOND (6 MHz XTAL)
- PROGRAMMABLE OUTPUT LINE
- TWO INTERRUPT REQUEST OUTPUTS
- HARDWARE/SOFTWARE DIRECTION CONTROL
- HARDWARE/SOFTWARE START/STOP
- 'ABORT' CAPABILITY
- SINGLE/MULTIPLE STEP INSTRUCTIONS
- RAMP-UP/SLEW/RAMP-DOWN MODE
- 24 INSTRUCTIONS IN SET
- TRIGGERED OPERATION
- 'DO-WHILE' COMMAND
- 'WAIT-UNTIL' COMMAND
- SEVERAL SYNC INPUTS AND OUTPUTS

PIN CONFIGURATION



LOGIC DIAGRAM

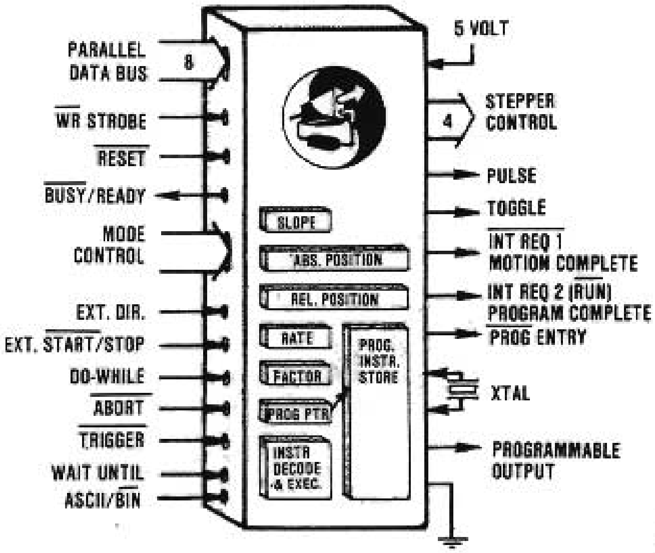


TABLE OF CONTENTS

Section 1
INTRODUCTION TO THE CY500

Section 2
OVERVIEW OF PIN FUNCTIONS

Section 3
OVERVIEW OF COMMAND LANGUAGE

Section 4
DETAILED EXAMPLES OF COMMANDS

Section 5
BINARY MODE OF OPERATION

Section 6
TIMING AND CONTROL INFORMATION

Section 7
STEP RATE INFORMATION

Section 8
ELECTRICAL SPECIFICATIONS

Section 9
CIRCUITS AND EXAMPLES

Section 10
COMPUTER CONTROL OF THE CY500

Section 11
IEEE-488 INTERFACE

Section 12
GETTING YOUR CY500 RUNNING

APPENDICES

1 INTRODUCTION TO THE CY500 1

The CY500 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.

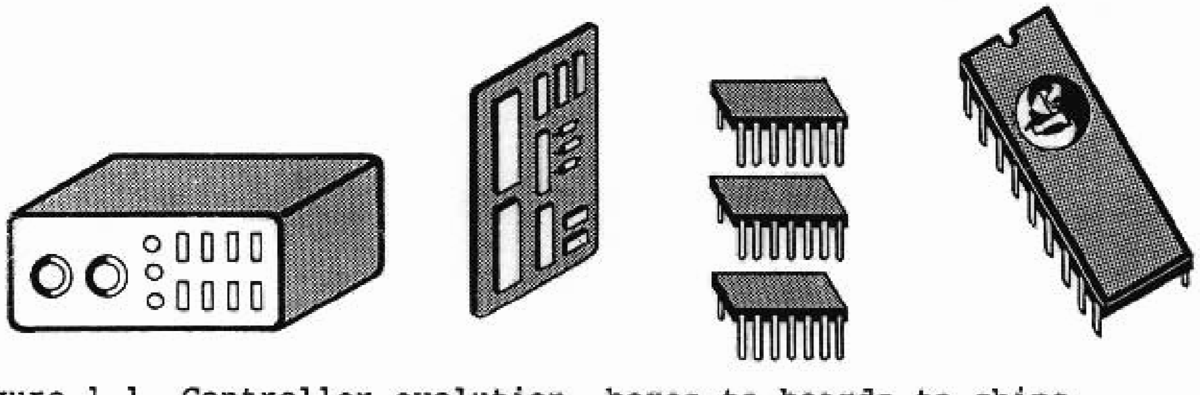


Figure 1.1 Controller evolution, boxes to boards to chips.

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 CY500 Stored Program Stepper Motor Controller not only contains the timing logic of the earlier bulky designs, it also offers parametric control over step rate, acceleration rate, numbers 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.

A STORED PROGRAM PERIPHERAL CONTROLLER

The Cybernetic Micro Systems CY500 Stored Program Stepper Motor Controller is the first peripheral controller device to offer the user stored program capability. 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:

1. Command execution
2. Program entry
3. Program execution

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.

CUSTOM DEVICE GENERATION

In many applications the user will find that the CY500 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 devices. The user can then employ these custom controllers in stand-alone applications with no host.

ARCHITECTURE OF THE CY500

The CY500 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. 18 byte program buffer
6. Instruction selection, decoding, and control, mechanisms.
7. Position register

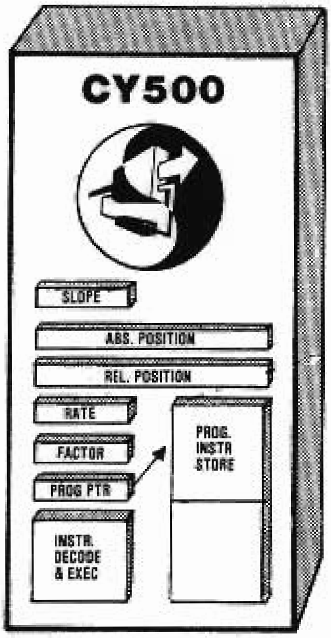


Figure 1.2 Schematic diagram of architecture of the CY500 Stored Program Stepper Motor Controller.

Input and Output Data Subsystems

The input data system accepts commands from the host (or keyboard, as described in a later section). The output data subsystem holds the output control signals to the stepper drive circuitry and includes the associated toggle and pulse timing line.

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. 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.

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.

18 Byte Program Buffer

The CY500 contains an 18 byte 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 CY500 contains a 16-bit position register that can be used to determine the current location. The CY500 will accept relative and absolute position commands; however, the position register always indicates absolute position.

KEYBOARD PROGRAMMABLE DEVICE

The CY500 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.

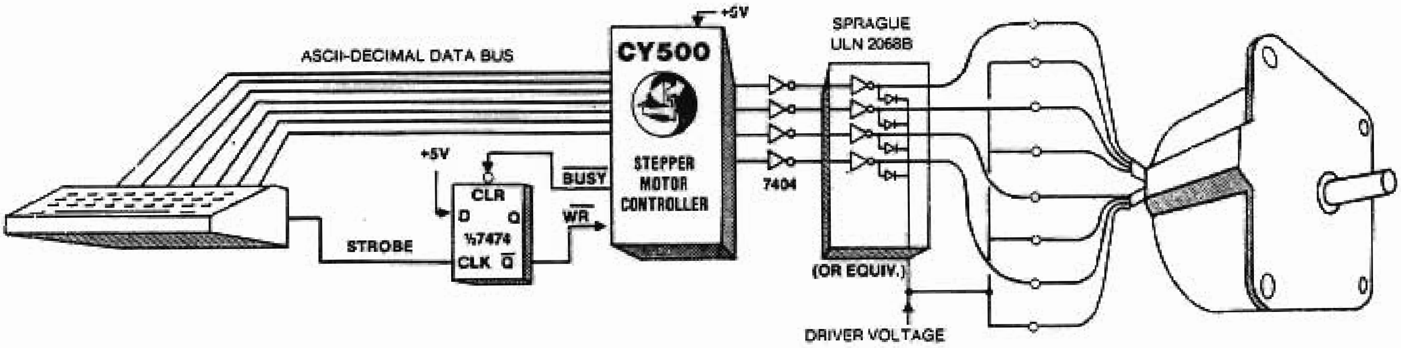


Figure 1.3 Prototype Development System.

HOST COMMUNICATION WITH CY500

The host can issue commands to the CY500 using a parallel data format. In parallel operation, complete handshaking operation occurs via the use of a BUSY/RDY line on the CY500, and the WRite strobe line, WR, from the host. The handshake protocol is shown in Figure 2.1.

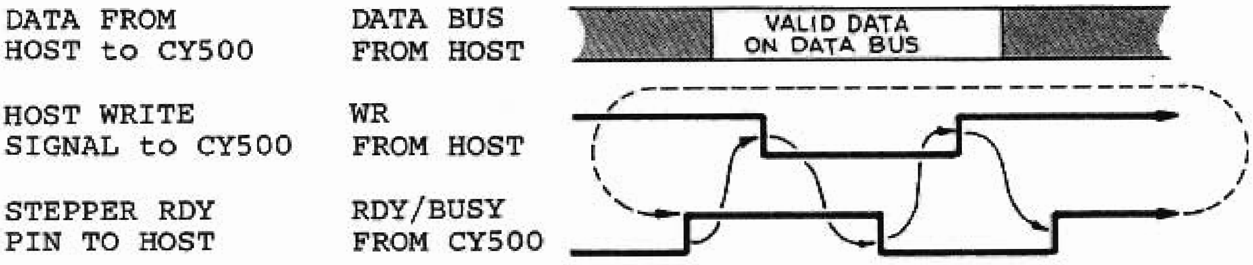


Figure 2.1 Handshaking protocol for CY500 in parallel input mode.

PROGRAM 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 CY500 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 CY500 is very simple, consisting of seven 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 CY500 Pulse outputs. The Pulse line gives one pulse at the beginning of each step.

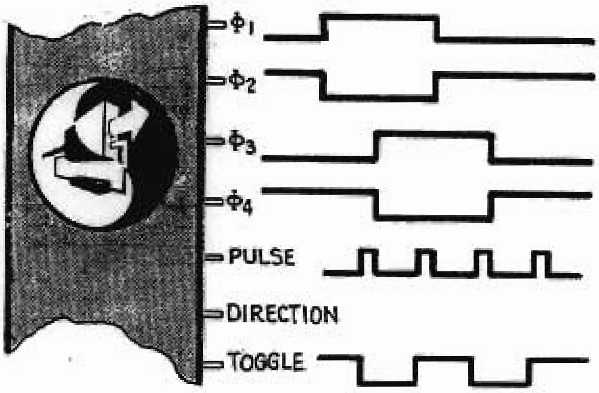


Figure 2.2 Motor Interface

The computer or data interface of the CY500 is also very simple. Commands and parameters are passed from the command source to the CY500 on an eight bit data bus. Data transfer between the command source and the CY500 is controlled by a standard two-line handshake protocol. The master processor waits for the CY500 BUSY/READY line to go high, indicating that the CY500 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 Write. Data should remain stable until the CY500 indicates data accepted by a high-to-low transition of the BUSY/READY line. During this busy time, the CY500 is processing the character just received. The master processor should then raise the Write line and wait until the CY500 is ready for the next data byte. The simplicity of the data transfer handshake, combined with the ASCII command structure of the CY500, 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.

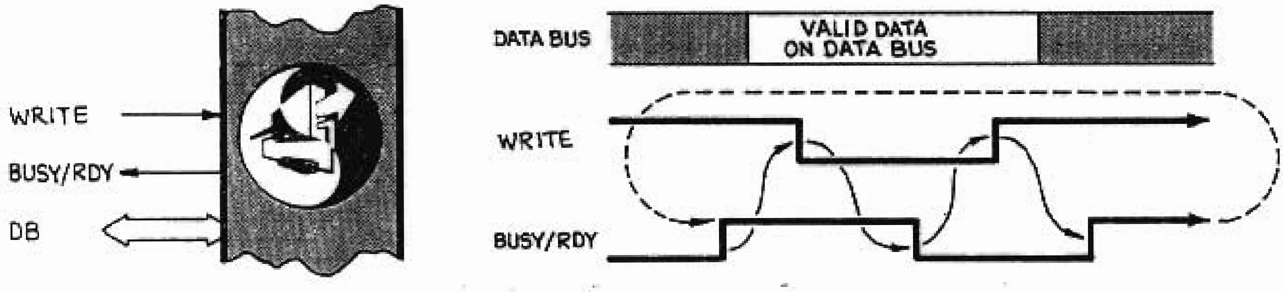


Figure 2.3 Data interface and handshake waveform.

Since most stepper motors are parts of functional systems, requiring that various parts of the system stay synchronized with each other, the CY500 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 Trigger 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. Abort signal is used to stop the motion. 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 External Start/Stop input is used with the conditional loop command. While the line is low, the CY500 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.

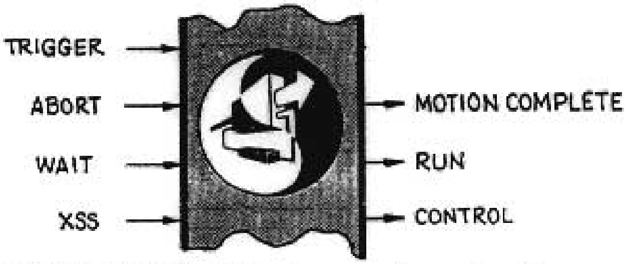


Figure 2.4 Secondary control inputs and outputs.

The CY500 also provides a number of output signals which may be used by other parts of the system. When the CY500 has stepped for the number of steps specified, the Motion Complete(INT REQ 1) signal indicates the end of the current motion. Run(INT REQ 2) is used to indicate that a program is executing. In addition, the CY500 provides an uncommitted output control, 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.

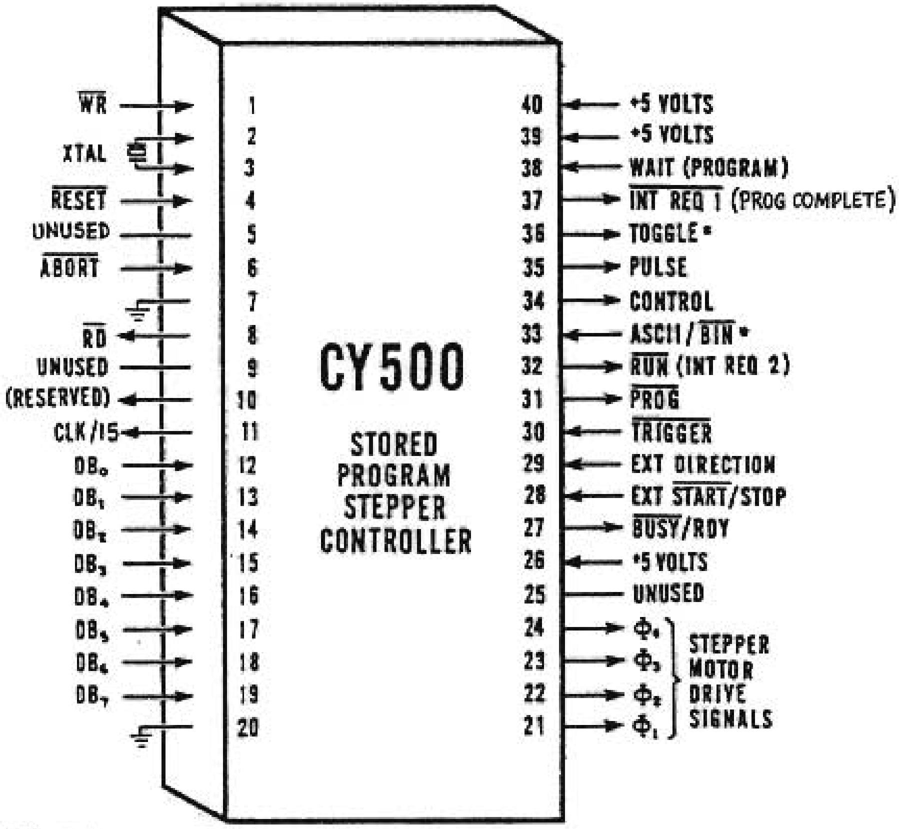


Figure 2.5 CY500 Pin Configuration.

NOTE: The CY512 Stepper Motor Controller is compatible with the CY500 with the following exceptions:

PIN#	CY500	CY512
33	(input) ASCII/BIN select	(output) EXT DIR indicator
36	(output) TOGGLE	(input) ASCII/BIN select
10	unused	(output) parallel write strobe
29	(input) EXT DIRECTION	(output) SLEW indicator
39	unused	(input) I/O SELECT

TABLE I CY500 PIN DESCRIPTION

DESIGNATION	PIN#	FUNCTION
VCC (input)	40	+5 volt power supply.
VDD (input)	26,39	+5 volts.
VSS (input)	7,20	circuit GND potential.
XTAL1,XTAL2 (input)	2,3	inputs for crystal or external clock (not TTL). See Clock Circuits section.
CLK/15 (output)	11	this output represents the clock signal divided by fifteen, independent of all operating modes. The pulse width is at least 400 nanoseconds.
$\overline{\text{RESET}}$ (input)	4	initializes controller to power-up state.
DB0-DB7 (input)	12-19	parallel data bus.
ϕ1-ϕ4 (output)	21-24	stepper drive signals.
$\overline{\text{WR}}$ (input)	1	write strobe from host to initiate command input when writing to CY500.
$\overline{\text{BUSY}}$/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 CY500.
$\overline{\text{RD}}$ (output)	8	the Read Strobe occurs during data input. The data on the bus must be valid until the trailing edge of RD occurs.
ASCII/$\overline{\text{BINARY}}$ (input)	33	selects ASCII-decimal or binary mode of operation.
$\overline{\text{PROG}}$ (output)	31	indicates program entry mode. Commands are entered but not executed while pin 31 is low.

(continued)

(continued)

TABLE I **CY500 PIN DESCRIPTION**

DESIGNATION	PIN#	FUNCTION
$\overline{\text{RUN}}$ (INT REQ 2) (PROGRAM COMPLETE) (output)	32	indicates program execution mode. Commands cannot be entered while program is executing (pin 32=low).
$\overline{\text{MOTION COMPLETE}}$ (INT REQ 1) (output)	37	signal to interrupt host at end of stepping.
EXT $\overline{\text{START}}$/STOP (input)	28	controls starting/stopping of stepper motor if in external start/stop control mode via the J command. If running a program, the pin is tested when the T command is encountered in the program. If low, program will continue looping, if high, program will exit loop and fetch next instruction.
WAIT (input)	38	program waits for this pin to go LOW when **wait UNTIL** instruction is executed, and waits for it to go HIGH when the **WAIT** command is encountered.
TOGGLE (output)	36	timing signal; toggles with each new step.
PULSE (output)	35	5 microsecond pulse at beginning of each step.
$\overline{\text{TRIGGER}}$ (input)	30	used to trigger each step when low and inhibits stepping when high.
EXTDIR (input)	29	selects direction (HI=CW, LOW=CCW) if external direction control is selected via LEFTRIGHT command L.
$\overline{\text{ABORT}}$ (input)	6	stepping motion is aborted when low. Next command in program is executed. Must be tied high if not used.
CONTROL (output)	34	user programmable output pin may be used for any purpose.
UNUSED	5,9,10,25	must remain disconnected.

BIN-ASCII™ FEATURE

The CY500 user-orientation has not been accomplished at 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 CY500 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 CY500 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 " ", and the decimal number 738 is then entered, followed by the carriage return key, ↵ = ØDH 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 for use with 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 CY500 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).

HI-LEVEL LANGUAGE DESIGN FACILITATES PROGRAMMING

The primary characteristic 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 CY500 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 blank, then ASCII decimal number parameter, then } = ØDH.

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
DOIT	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
FACTOR	F f}	DECLARE RATE DIVISION FACTOR
SLOPE	S s}	DECLARE RAMP RATE
POSITION	P p}	DECLARE TARGET POSITION (ABSOLUTE)

TABLE II **CY500 COMMAND SUMMARY**

ASCII CODE	**NAME**	**INTERPRETATION**
A	ATHOME	SET CURRENT LOCATION EQUAL ABSOLUTE ZERO
B	BITSET	TURN ON PROGRAMMABLE OUTPUT LINE
C	CLEARBIT	TURN OFF PROGRAMMABLE OUTPUT LINE
D	DOITNOW	BEGIN PROGRAM EXECUTION
E	ENTER	ENTER PROGRAM CODE
F	FACTOR	DECLARE RATE DIVISOR FACTOR
G	GOSTEP	BEGIN STEPPING OPERATION
H	HALFSTEP	SET HALFSTEP MODE OF OPERATION
I	INITIALIZE	TURN OFF STEP DRIVE LINES, RESET CONTROLLER
J	JOG	SET EXTERNAL START/STOP CONTROL MODE
L	LEFTRIGHT	SET EXTERNAL DIRECTION CONTROL MODE
N	NUMBER	SET NUMBER OF STEPS TO BE TAKEN (RELATIVE)
O	ONESTEP	TAKE ONE STEP IMMEDIATELY
P	POSITION	DECLARE TARGET POSITION (ABSOLUTE)
Q *	QUIT*	STOP SAVING PROGRAM, ENTER COMMAND MODE. ALSO QUIT STEPPING. ***NEVER FOLLOWED BY ;**
R	RATE	SET RATE PARAMETER
S	SLOPE	SET RAMP RATE FOR SLEW MODE OPERATION
T	LOOP TIL	LOOP TIL EXTERNAL START/STOP LINE HI
U	UNTIL	STOP EXECUTING UNTIL WAIT LINE IS LOW
W	WAIT	STOP EXECUTING UNTIL WAIT LINE IS HIGH
X	EXPEND	TIME DELAY FOR SPECIFIED MILLISECONDS
+	CW	SET CLOCKWISE DIRECTION
-	CCW	SET COUNTERCLOCKWISE DIRECTION
Ø	COMMAND	STOP PROGRAM EXECUTION, ENTER COMMAND MODE

DESCRIPTION OF COMMANDS

A) 0100 0001 1 byte

The ATHOME instruction (sets the position) defines the Home position. This position is absolute zero and is reference for all POSITION commands. The ATHOME command must not be immediately preceded by a change-of-direction command. Note also that the ATHOME command should not be used twice unless the CY500 is reset or initialized between commands.

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 CY500 to begin executing the stored program. If no program has been entered, the controller will return to the command mode. 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 changed) and may be used for looping or cyclic repetition of the program.

E) ENTER 0100 0101 1 byte

This instruction causes the CY500 to enter the program mode of operation. All commands following the ENTER command are entered into the program buffer in sequence.

F) FACTOR f 0100 0110 2 bytes

b7 b0

The FACTOR command causes the rate to be decreased by the factor (1/f). The factor, f, is a number from 1 to 255.

G) GO command 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.

H) HALFSTEP command 0100 1000 1 byte

The HALFSTEP command causes the CY500 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 CY500 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.

J) JOG 0100 1010 1 byte

The JOG command places the CY500 in the external start/stop control mode where the starting and stopping are controlled by external hardware and not by the software GO command. The **J** command is applied after the rate, mode, and direction parameters have been specified, although new parameters may be entered after jogging has started. Applying a low voltage to the XSS pin causes the motor to begin stepping and continue stepping as long as pin #28 is low. The Quit command ends Jog mode. Note that the **J** command and the **T** command are mutually exclusive; one or the other, not both, may be used. See JOG in Section 4 for more details.

L) LEFT/RIGHT pin enable 0100 1100 1 byte

This instruction places the CY500 in the external direction control mode. In this mode, the External Direction Pin (#29) is used to determine the direction, either clockwise (1) or counter-clockwise (Ø). Software commands + and - are ignored in this mode. Any change of control signal applied following the output pulse will be used to determine the direction of the next step.

N n) NUMBER of steps	0100 1110	3 bytes
	a7 a0	LSbytes
	b7 b0	MSbytes

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. Note that this parameter is stored as 2 bytes in the program buffer.

O) ONESTEP 0100 1111 1 byte

The ONESTEP command is used to take a single step. The steps are untimed, and the step rate is determined by the rate at which **O** commands are received. In addition, each step may be triggered externally via the TRIGGER pin.

P p↵ POSITION

0101 0000
a7 a0
b7 b0

3 bytes

The POSITION command declares the Absolute mode of operation. The argument is treated as the target position relative to position zero. The ATHOME command can be used to define position zero.

Q QUIT (Programming)

0101 0001

1 byte

NOTE: The QUIT command is self-terminating, and should NOT be followed by the Linend ↵.

The QUIT command causes the CY500 to exit the Programming mode of operation wherein instructions are stored in the program buffer in the order received, and to return to the Command mode of operation in which instructions are executed as they are received. Note that the **Q** command is not terminated with the Linend character, 0DH = ↵, and such termination may result in incorrect operation. The QUIT command is also used after program completion prior to entering new parameters.

R r↵ RATE rate

0101 0010
b7 b0

2 bytes

The RATE instruction sets the rate parameter that determines the step rate. The rate parameter, r, varies from 1 to 253 corresponding to step rates from 50 to 3350 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 III. For crystals other than 6 MHz the step rate should be multiplied by f/6MHz, where f is the crystal frequency.

S s↵ SLOPE

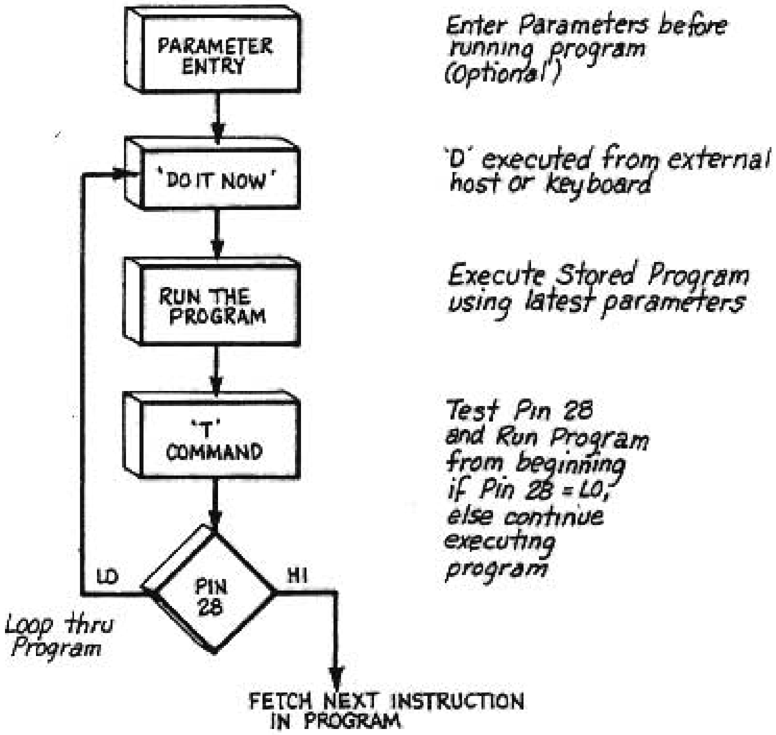
0101 0011
a7 a0

2 bytes

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**), (FACTOR is forced to 1 automatically), and the ramp rate or change in rate parameter from step to step. When executing in the SLOPE mode, the CY500, starting from rest, increases the rate 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. NOTE: The user is responsible for insuring that N > 2(maxrate). **The S command may be used only with N n↵, it will not work with P p↵.**

T) 0101 0100 1 byte

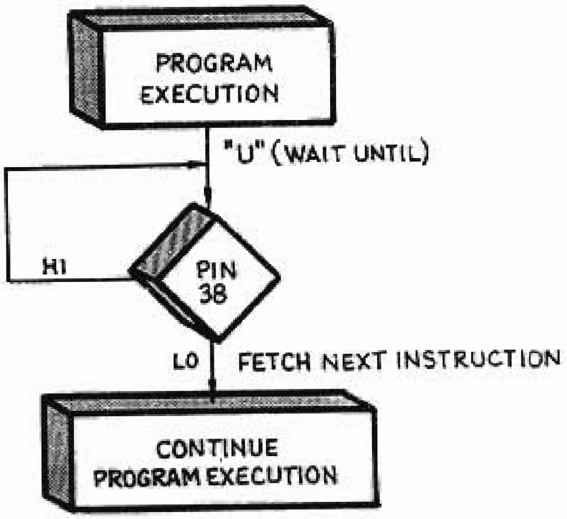
The T command provides a Do...While... capability to the CY500. This command tests pin 28 and, if low, it executes the DOITNOW command, i.e., it runs the program from the beginning (although using the latest rate, position, etc., parameter). If pin 28 is high, the next instruction is fetched and executed. Note that pin 28 is also used for external start/stop control when J (JOG) is executed. T and J are therefore mutually exclusive and can not be used together.


PARAMETER ENTRY
Enter Parameters before running program (Optional)
'DO IT NOW'
'D' executed from external host or keyboard
RUN THE PROGRAM
Execute Stored Program using latest parameters
'T' COMMAND
Test Pin 28 and Run Program from beginning if Pin 28 = LO; else continue executing program
PIN 28
LO
HI
Loop thru Program
FETCH NEXT INSTRUCTION IN PROGRAM

Example: Loop TIL command use.

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 it tests the WAIT pin. When the WAIT pin goes low, the next instruction is fetched from the program buffer and execution proceeds.



Example: wait UNTIL command use.

W) WAIT command 0101 0111 1 byte

The WAIT instruction is the opposite of the U command. WAIT test the WAIT 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 these transitions.

X x) EXPEND command

	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 A 6 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 6 MHz. This command is useful in programming a delay time between stepping motions.

+) CLOCKWISE command 0010 1011 1 byte

All steps following this command will be taken in a clockwise direction.

-) CCW command 0010 1101 1 byte

All steps following this command will be taken an a counter-clockwise direction.

Ø) COMMAND mode 0011 0000 1 byte

The CY500 is placed in the command mode and the next command is executed as it is received.

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). Note that this command de-energizes the stepper coils. 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.

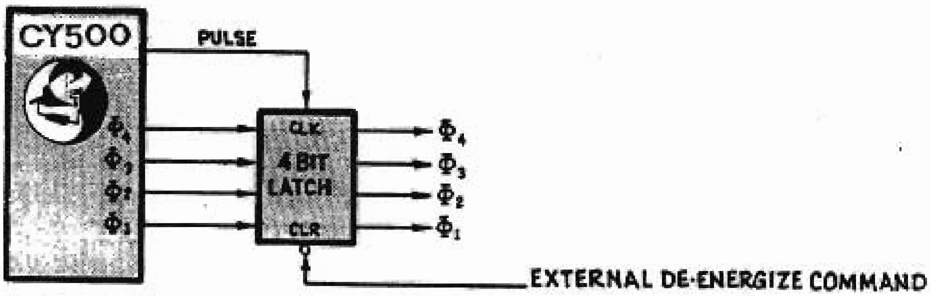


Figure 4.1 An external latch on the stepper control outputs prevents de-energizing of stepper drive coils when CY500 is reset via hardware or software and also allows an external control line to de-energize the coils independently of the CY500.

PROGRAM EXECUTION MODE

Once a program has been entered in 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 prior to program execution. It is a normal command in the sense that it is terminated with a LINEND, ↵ = ODH.

HOME POSITION

The ATHOME command, **A**, is used to declare the Home position, assigned absolute value zero. Note that the ATHOME command should not be immediately preceded by either +↵ or -↵; i.e. by either direction command. Note also that the ATHOME command should not be used twice unless the CY500 is reset or initialized between commands.

DIRECTION CONTROL

The control of the direction of motion may be obtained in either of two ways. The default mode of operation is that in which the user specifies either + (CW) or - (CCW) via software instruction. The system powers up in the clockwise direction. Note that the direction commands are separate commands; they are terminated by the carriage return character, ↲=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↲
```

The second method of controlling direction is via the L command which instructs the CY500 to test the External Direction control line (pin 29) to determine stepping direction.

ABSOLUTE vs RELATIVE POSITION

The CY500 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 $0<n\leq2^{16}-1$. 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 number of steps P is interpreted as absolute position with respect to the zero location declared by the ATHOME command. A three command sequence is required to step in the absolute position mode. First the target position is entered with the P p↲ command. Next, the stepping direction is set via the + or - command. Finally, stepping is initiated with the G command. After moving to the specified position, the system reverts to the relative mode. You may not use ramping with absolute position motions.

EVENT	POSITION MODE	DESCRIPTION
RESET	REL	hardware initialization
I↲	REL	software initialization
P p↲	ABS	absolute position command
INT REQ1	REL	rel mode set when motion complete

COMMANDING THE CY500 DURING "JOG" MODE OF OPERATION

The CY500 may be commanded while actively stepping by using the jog command. In particular, the following commands may be useful while the CY500 is in motion. (At higher step rates the command execution may introduce some jitter into the stepper control outputs.)

1. CLEARBIT will reset the control output (pin 34).
2. BITSET will set the control line (pin 34).
3. + will set CW if in internal direction mode.
4. - will set CCW if in internal direction mode.
5. **I** will initialize the system, resetting the XSS mode to off and disabling the stepper control outputs Φ1-Φ4. If this is undesirable, these outputs can be latched using the Pulse Line to clock an external latch as shown in figure 4.1.
6. **L** will set the external direction control.
7. **H** will set the halfstep mode. **E↵** followed by Q will stop the motion, clear the XSS mode (restart with **J↵**) and will strobe PROG.
8. **Q** will stop the XSS mode (restart with J↵).
9. TRIGGER stops the motion, $\overline{\text{TRIGGER}}$ releases it.
10. **R** will dynamically change the stepping rate.

The MOTION COMPLETE (INT REQ 1) goes low in XSS mode when either XSS or TRIGGER is used to stop motion.

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 CY500 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.

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 the RESET line.

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 CY500. The **T** command (loop TIL) is used as shown in the following example.

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:

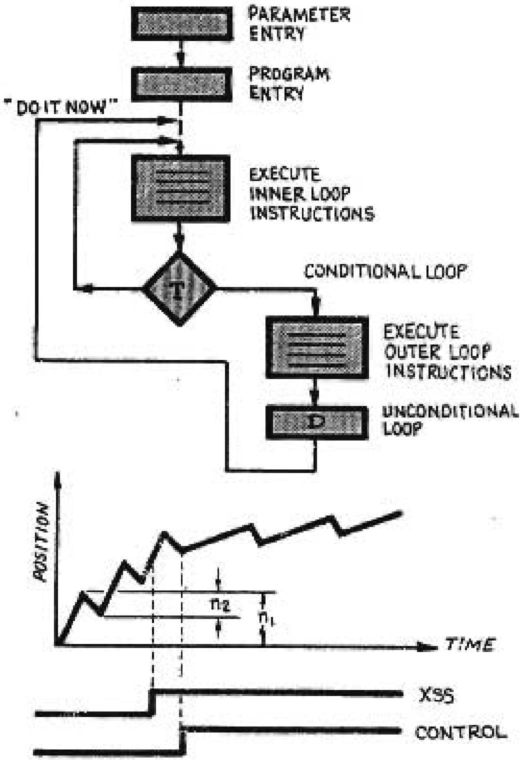


Figure 4.2 Conditional Loop

N n}	set number of steps = n
E}	enter program mode
+}	set CW direction
R r1}	set rate = r1
G}	begin stepping
-}	set CCW direction
R r2}	set rate = r2
G}	go (same n)
T}	loop TIL pin 28 goes HI
B}	set control output line
Ø}	return to command mode
Q	quit progam mode
D}	begin executing program

```
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
```

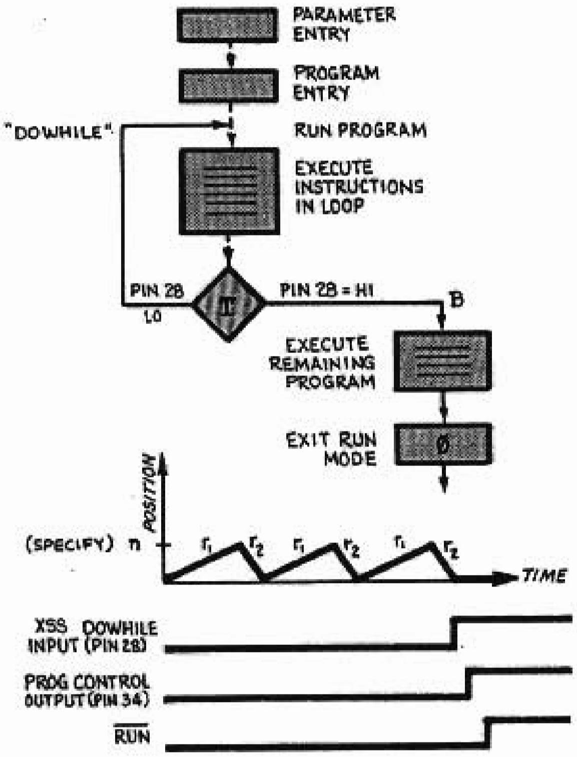


Figure 4.3
Conditional Loop Imbedded
in an Unconditional Loop.

OPERATIONAL MODE SUMMARY

MODE DESCRIPTION	MODE Ø *	MODE 1	MODE SELECTION VIA
START/STOP CONTROL	INTERNAL*	EXTERNAL	'J' COMMAND SELECTS EXTERNAL CONTROL
DIRECTION CONTROL	INTERNAL*	EXTERNAL	'L' COMMAND SELECTS EXTERNAL CONTROL
DATA TYPE	ASCII DECIMAL	BINARY	(PIN 33 = HI/LO) (ASCII/$\overline{\text{BIN}}$)
STEP COMMAND	MULTI-STEP	SINGLE-STEP	**'G'** COMMAND SELECTS MULTI, **'O'** COMMAND SELECTS ONESTEP
POSITION TYPE	RELATIVE**	ABSOLUTE**	**'N'** COMMAND SELECTS RELATIVE, **'P'** COMMAND SELECTS ABSOLUTE
STEP MODE	FULL-STEP***	HALF-STEP	**'H'** COMMAND SELECTS HALFSTEP
ACCELERATION	STEPPED***	RAMPED	**'S'** COMMAND SELECTS SLOPE FOR ACCELERATION
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, **'Ø'** SELECTS COMMAND MODE

*MODE Ø 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).

Figure 4.4 Operational Mode Summary.

BINARY DATA MODE

To facilitate control via microprocessors using binary arithmetic, the CY500 can be placed in the BINARY data mode of command execution via the application of a low voltage to pin 33. 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 inadvertantly 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 bytes of binary data to be sent to the CY500. 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 th 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 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 III 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 Position, 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. Note that 16-bit values are sent least significant byte first. All commands except QUIT are of the form shown in Table III.

TABLE III ALLOWED BINARY COMMANDS AND DATA COUNT FOR EACH

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

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	CY500 BUFFER CONTENTS
E)	45	45	
	00	00	
R 150)	52	52	52
	01	0C	
	96	96	96
N 300)	4E	4E	4E
	02		
	2C	2C	2C
	01	01	01
+)	2B	2B	2B
	00		
G)	47	47	47
	00		
N 750)	4E	4E	4E
	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 CY500 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 CY500 program buffer can contain 18 bytes of program commands and data. The description of instructions contains the length of each command and is summarized in Table IV. 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 CY500 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 IV INSTRUCTION LENGTHS AND PARAMETER CHARACTERISTICS

INSTRUCTION	BYTES	PARAMETER	RANGE
ATHOME	1		
BITSET	1		
CLEARBIT	1		
DOITNOW	1		
ENTER	1		
FACTOR	2	rate divisor	1-255
GO	1		
HALFSTEP	1		
INITIALIZE	1		
JOG	1		
LEFTRIGHT	1		
NUMBER	3	travel distance	1-65535
ONESTEP	1		
POSITION	3	target location	0-65535
QUIT	*		
RATE	2	rate parameter	1-255
SLOPE	2	acceleration	1-255
TIL	1		
UNTIL	1		
WAIT	1		
EXPEND	3	msec time delay	1-65535
+	1		
-	1		
Ø	1		

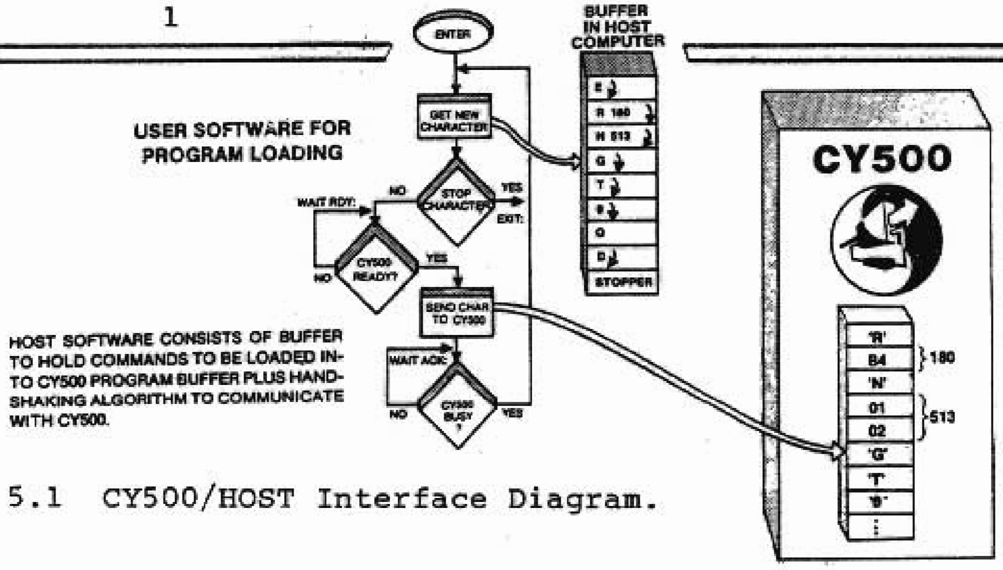


Figure 5.1 CY500/HOST Interface Diagram.

INTERFACE EXAMPLE

In the following, it will be assumed that the 8080/8085 transmits data to the CY500 via output port ODCH. The string of ASCII commands is stored at BUFFER, terminated by a stopper, or terminal symbol OFFH. The D-E register pair will be used to access the character string.

```
                ;
                ;
0081 11A800   LXI       D,BUFFER
              GETCHAR:
0084 1A       LDAX      D
0085 FEFF     CPI       OFFH
0087 CA9300   JZ        QUIT
008A 13       INX       D
008B 0E07     MOV       C,A
008D CD9400   CALL      SNDCHAR
0090 C38400   JMP       GETCHAR
                ;
              QUIT:
0093 CF       RST       EXIT
                ;
              SNDCHAR:
0094 DBB5     IN        0B5H
0096 E601     ANI       1
0098 CA9400   JZ        SNDCHAR
009B 79       MOV       A,C
009C E67F     ANI       7FH
009E D3DC     OUT       0DCH
              GOTIT:
00A0 DBB5     IN        0B5H
00A2 E601     ANI       1
00A4 C2A000   JNZ       GOTIT
00A7 C9       RET
              BUFFER:
                ;
                ;
```

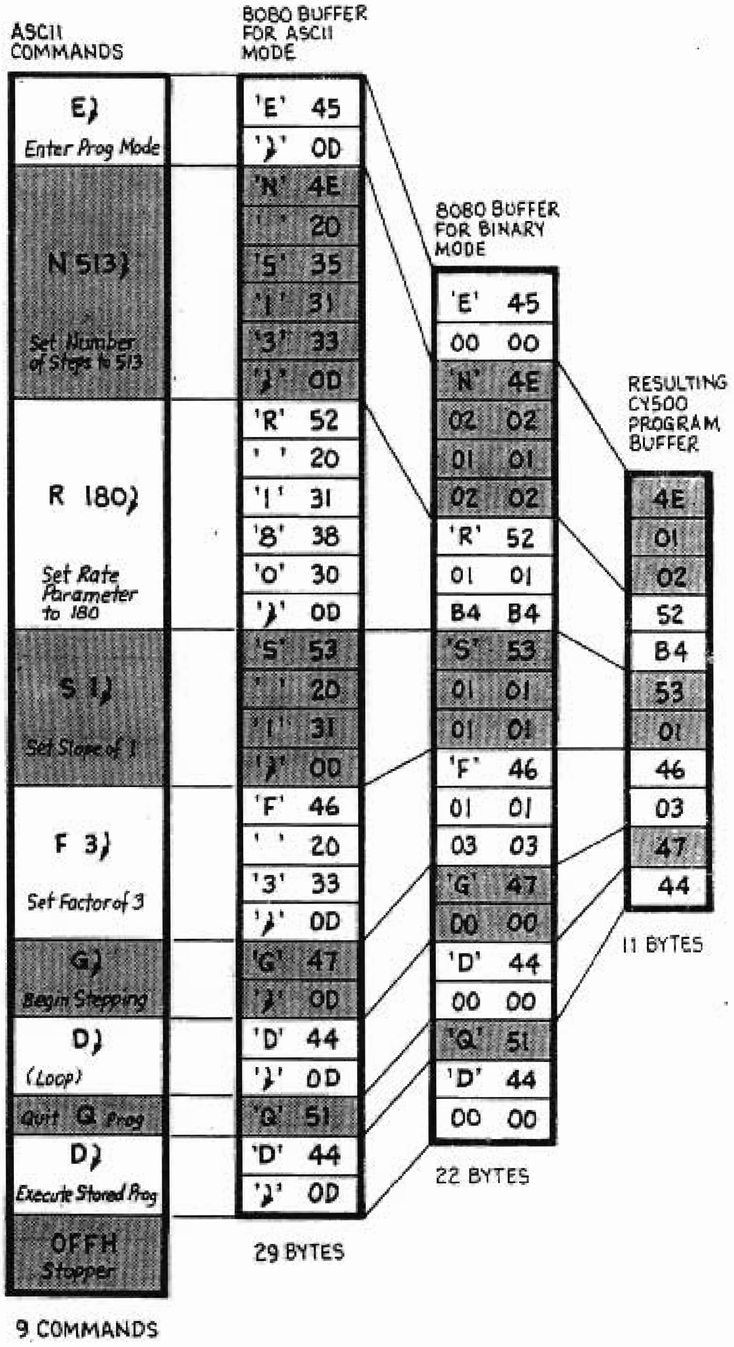


Figure 5.2 8080/8085 Interfaced to Stepper Motor through CY500 Stepper Motor Controller.

6 TIMING AND CONTROL INFORMATION 6

CY500 TIMING AND CONTROL INFORMATION

In the parallel interface mode the user must wait for the CY500 RDY line (pin #27) to be high before applying a WR strobe to pin #1. Note that no **data set-up time is required**, that is, the data may appear on the data bus at the same time that the write strobe, WR, 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 (see figure 9.4). The data is read into the CY500 from the data bus by a low going read strobe, RD, appearing on pin #8. The data should be valid at the trailing edge of RD. RD 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 RD, however **the WR line should be held low until the RDY line acknowledges the transfer by going low.** The simplest interface ignores the RD strobe and uses the RDY/BUSY line only, as shown in figure 2.1.

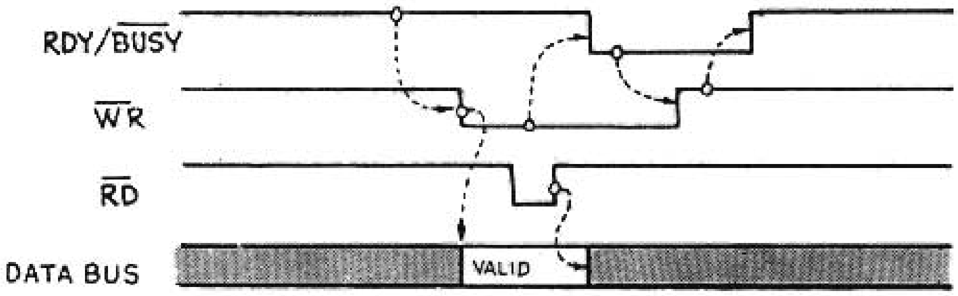


Figure 6.1 Parallel Handshake Timing Sequence.

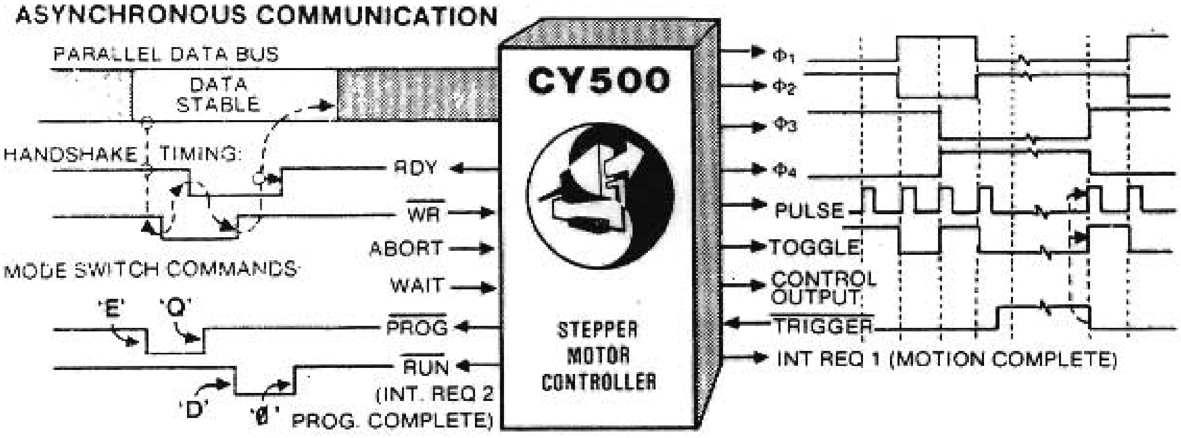


Figure 6.2 CY500 Timing and Control Signals.

DOITNOW (RUN) TIMING

After entering program code into the CY500 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 CY500 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 6.3. Using a 6MHz crystal, the CY500 will be busy approximately 150 microseconds with the D command and approximately 400 microseconds after detecting the end-of-command code (↵=ØDH). The CY500 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 the control output (pin 34) has been previously cleared) the control line will go high in less than 100 microseconds.

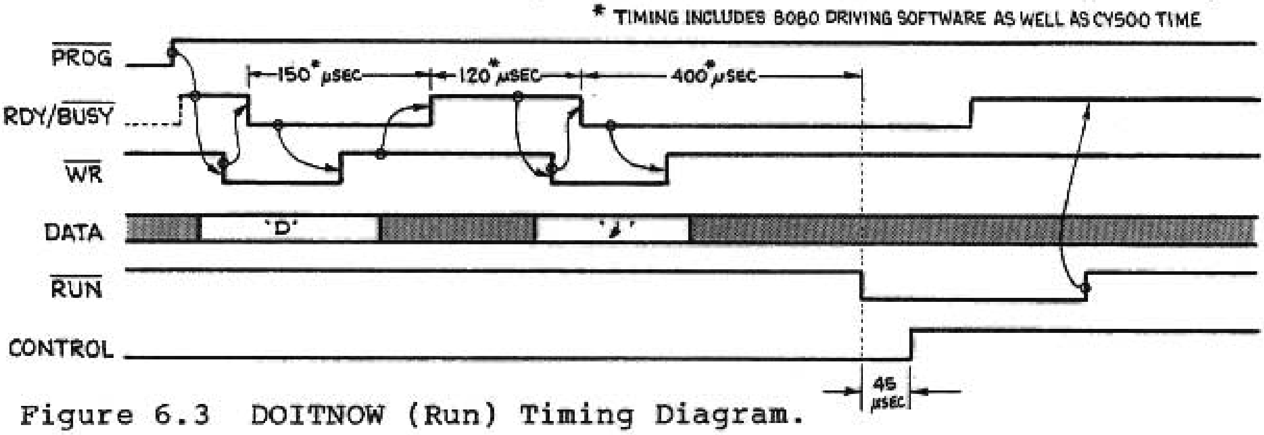


Figure 6.3 DOITNOW (Run) Timing Diagram.

PROGRAM 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 occurence, such as an operator input. The CY500 has been designed with both signal emitters and detectors to allow easy synchronization of the program to neighboring (interacting) subsystems. For example, one output pin toggles with each step taken and another produces a pulse (of duration 5-25 microseconds) with each step. One input pin is used to gate each step if the TRIGGERED mode of operation is used, while another input pin is used to provide PROGRAM WAIT until the signal on this pin goes LOW, at which time program execution continues with the next instruction. If enabled by software command or program instruction, two other input pins can be used to control direction of stepping, and to start and stop stepping under external control.

One output pin can be set or cleared by program instruction, thus allowing the program to provide a synchronization signal for synchronizing some external device, for use with an external pulse counter for counting program loops, or any other use the designer finds appropriate. An ABORT line is provided to allow detection of stops, and both MOTION COMPLETE and PROGRAM COMPLETE outputs are available for interrupting the host or for providing this information to other subsystems. These input and output control lines are shown in Figure 6.4.

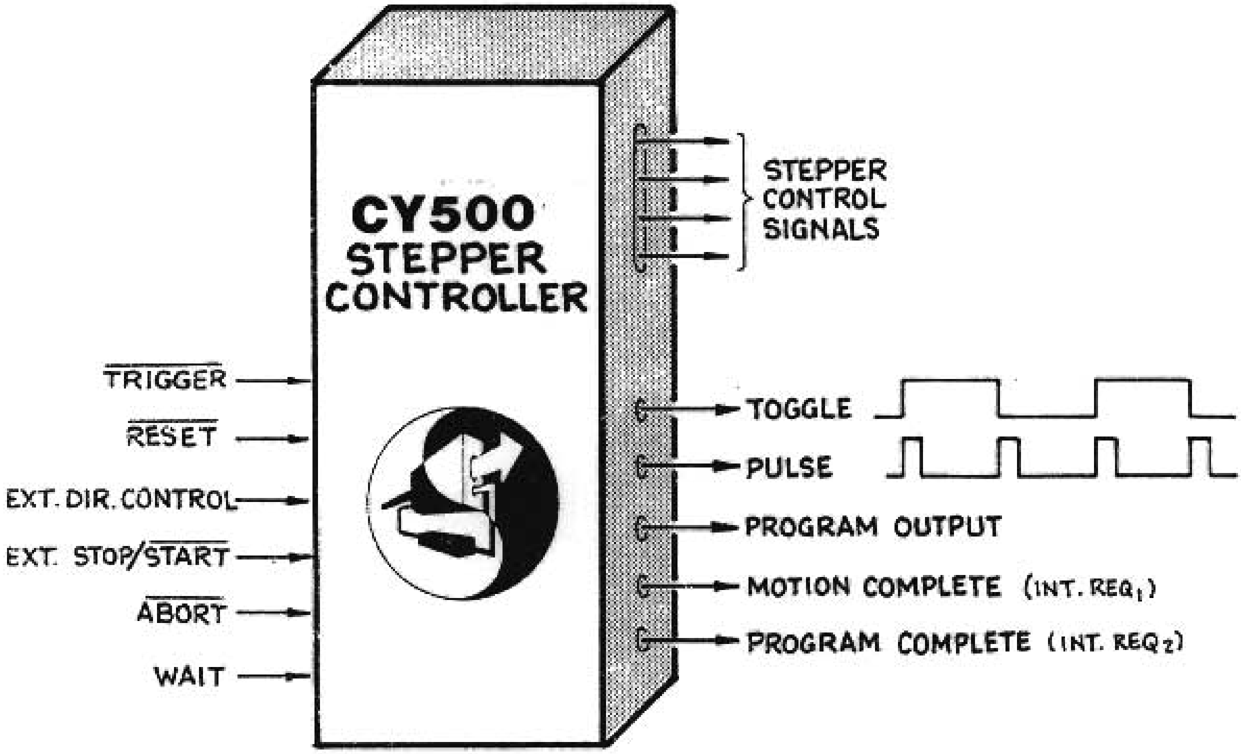


Figure 6.4 Input pins allow user control of CY500 operations or facilitate synchronization of CY500 to external events. Output lines allow CY500 to provide synchronization or control signals to external devices.

TRIGGER OPERATION

In the triggered mode of operation, the GO command initiates the stepping sequence if the trigger pin is LOW. If the trigger pin (pin #30) is HIGH, the controller simply waits for a LOW level on this pin and then takes one step. This signal may be used with SINGLE step (ONESTEP) operation or MULTI-STEP operation.

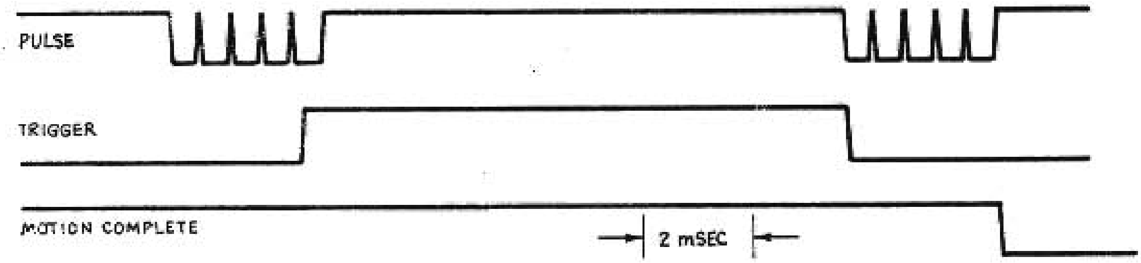


Figure 6.5 Trigger Timing.

THE ABORT SEQUENCE

If the ABORT line is brought low during a stepping sequence, the sequence is aborted and the motion complete (INT REQ 1) line goes low. If a program is executing, the next instruction is fetched else the the CY500 simply waits in the command mode for the next command. Note that **if the ABORT line is made active low in the slew (SLOPE) mode, the CY500 does not decelerate, but comes to an immediate stop.** The ABORT line may be used to find the home position by stepping CCW until a limit switch pulls the abort line low. If the next instruction in the program buffer is ATHOME, then this position will be declared position zero.

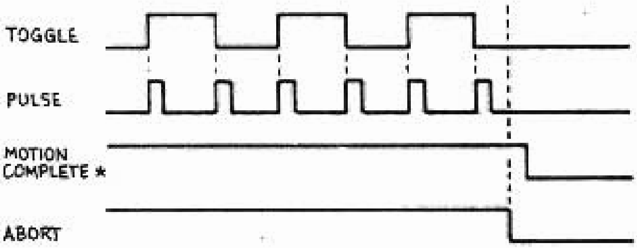


Figure 6.6 MOTION COMPLETE signal (INT REQ 1) is cleared at next step or with receipt of new byte.

EXTERNAL CONTROL

If the J command is executed, the external START/STOP (XSS) pin is used to begin stepping at the previously specified rate. The wstepping continues as long as XSS is low and stops when XSS rises. If the L command has also been executed, the direction control is affected via EXTDIR. The clockwise (CW) direction is selected when EXTDIR is HIGH and counter-clockwise (CCW) when LOW. An example of external direction and start/stop control is shown in Figure 6.7.

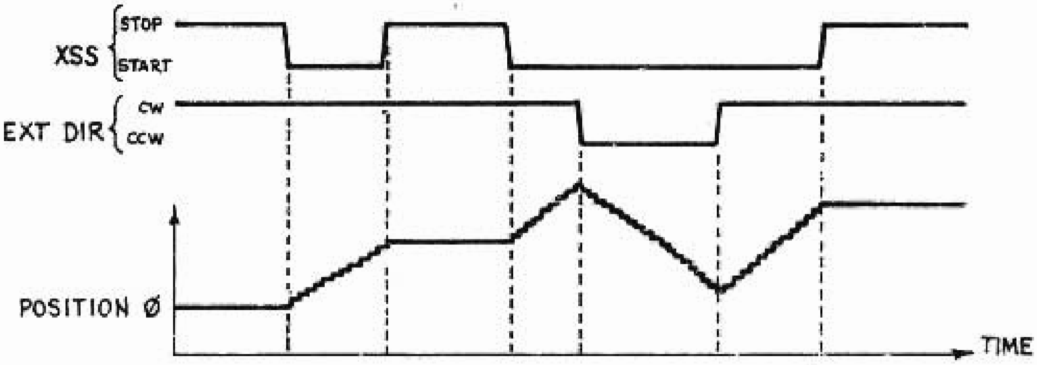


Figure 6.7 Example of operation with external (hardware) control of startstop and direction via XSS and XDIR pins.

INTERNAL/EXTERNAL DIRECTION CONTROL

In most applications the user will control direction of stepping via the software commands (+ for CLOCKWISE, and - for COUNTER-CLOCKWISE). The CY500 powers up in the software or internal direction control mode. For those applications in which an external signal is to be used for direction control, the user sends an **L** (LEFTRIGHT) command to place the CY500 in the external direction control mode. All steps occurring after this command is sent will be in the direction determined by the control pin. The internal direction control mode is restored only by a hardware or software system reset command.

The default mode is software direction control. A software command is used to select the external direction control mode. In a similar fashion, the default mode of operation of the controller requires a **G** command to execute a step or series of steps. However, the **J** command specifies external-start-stop mode and allows an external device to start and stop the step sequence by controlling the EXT START/STOP pin (pin #28).

External Direction Control

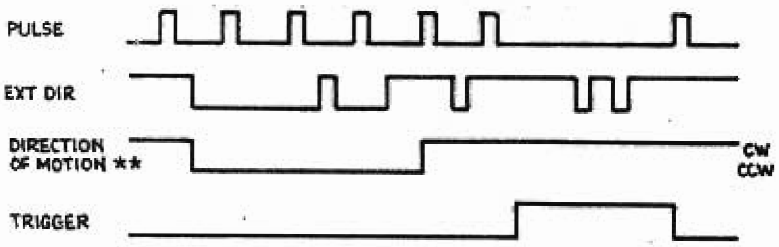


Figure 6.8 Direction of motion is not available externally except through observation of motion.

STOP/START OPERATION

If the JOG (External Start/Stop) command **J** is given, the control of stepping is effected via the voltage level on the XSS pin (pin #28). When the XSS pin is LOW, the stepper will begin stepping in the specified direction (CW/CCW) at the specified rate in the selected mode (HALF step/FULL step) and will continue until the XSS pin goes HIGH. In this mode the CY500 will receive all commands.

The MOTION COMPLETE (INT REQ 1) goes low in XSS mode when either XSS or TRIGGER is used to stop motion.

External Start/Stop Operation

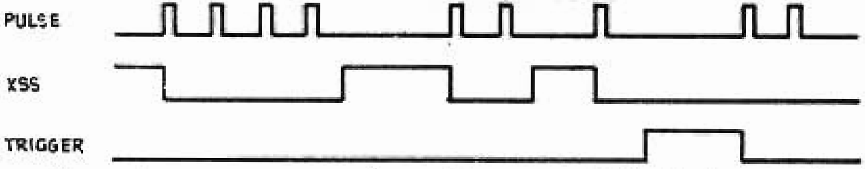


Figure 6.9 Stepping occurs at rate specified by prior parameter settings.

STEP TIMING SIGNALS

Two timing signals are provided for the convenience of the user. The PULSE signal goes HI for approximately 5 micro seconds at the beginning of each step time. The TOGGLE signal changes state at each step transition. Both of these signals are shown in Figure 6.10.

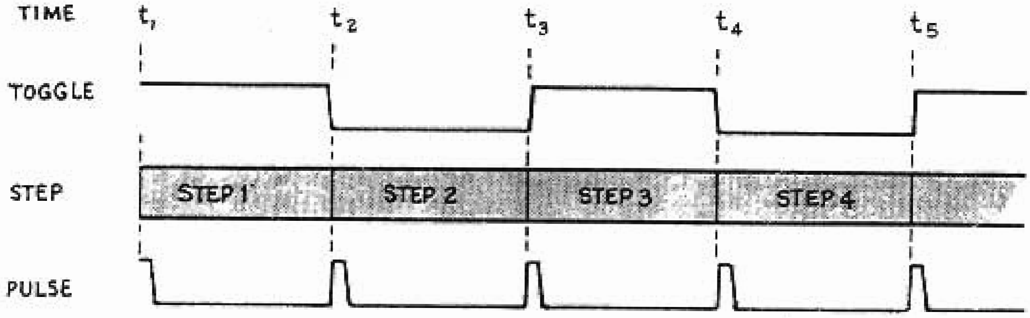


Figure 6.10 Step Timing Signals

FULL STEP DRIVER SIGNALS					
STEP	1	2	3	4	1
Φ_1	0	1	1	0	0
Φ_2	1	0	0	1	1
Φ_3	0	0	1	1	0
Φ_4	1	1	0	0	1

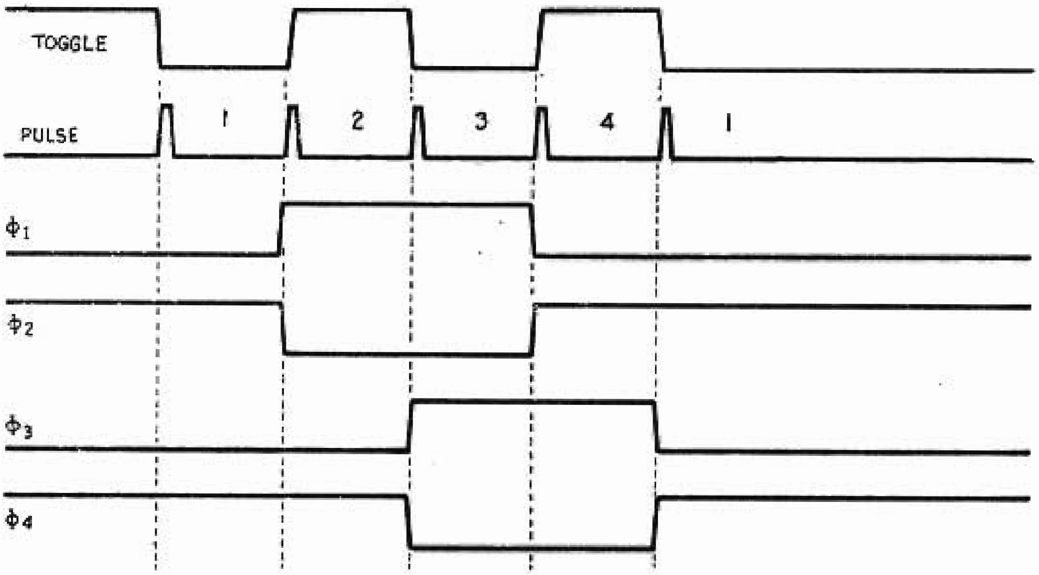


Figure 6.11 Full Step Control Outputs.

HALF STEP DRIVER SIGNALS

STEP	1	2	3	4	5	6	7	8	1
Φ_1	0	1	1	1	1	1	0	0	0
Φ_2	1	1	0	0	0	1	1	1	1
Φ_3	0	0	0	1	1	1	1	1	0
Φ_4	1	1	1	1	0	0	0	1	1

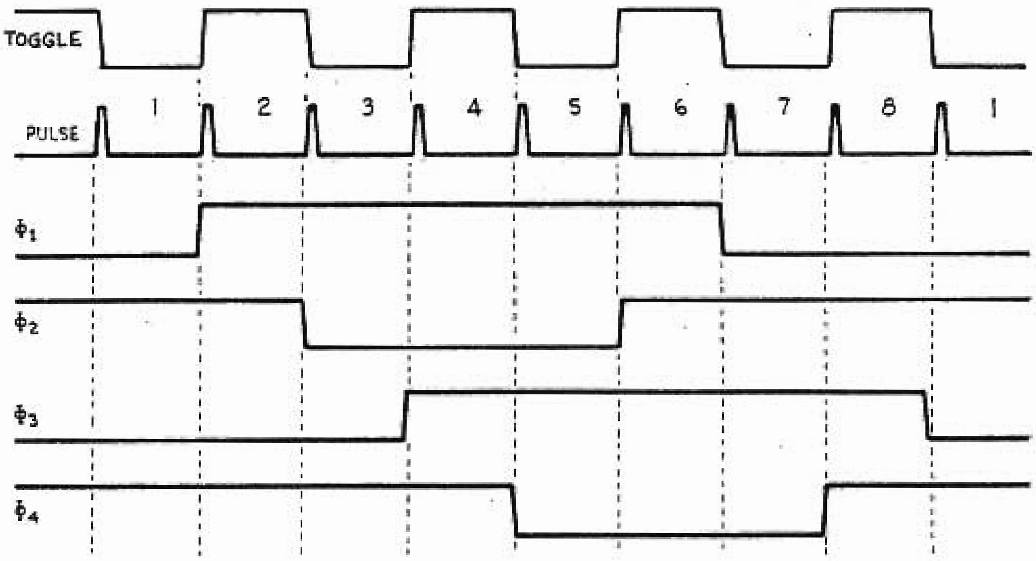


Figure 6.12 Half Step Control Outputs.

HALF STEP OR FULL STEP

Steppers operate 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 numbers 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.

7 STEP RATE INFORMATION 7

RATE CONTROL

The CY500 has been designed to provide powerful, precise position control. The step rate is also controllable over a range varying from one step every five seconds to 3350 steps/sec using a 6MHz crystal. The rate parameter is non-linear, with increased resolution at the slower speeds. At the higher rates it is possible to use an external pulse source to achieve precise rate control. The pulses are applied to the trigger input and the CY500 takes a step each time the trigger input goes low. The trigger input should be applied until the output pulse (pin #36) goes high and then removed as indicated in figure 7.1.

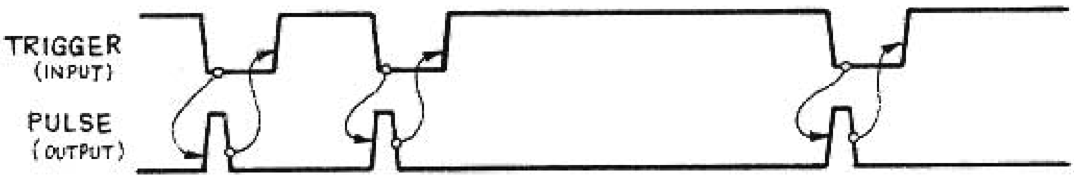


Figure 7.1 The CY500 waits for a low level on trigger (pin #30) before taking the next step.

The RATE instruction, **R r↵**, where r=rate parameter, specifies the step rate in the multiple step mode of operation. The rate parameter ranges from 1 to 253 corresponding to rates of 49 steps/sec and 3360 steps/sec respectively. (All rates discussed in this manual assume a 6 MHz crystal. For crystals with frequency fc the rates should be scaled be fc/6MHz.) The rates assume a rate FACTOR of 1, entered via the command **F 1↵**.

For rate parameter r in the range $1 \leq r \leq 255$ and a factor f in the range $1 \leq f \leq 255$, the step rate is computed via the equation:

$$\boxed{\text{steps/sec} = 1/(256-r)*80\ \mu\text{sec} + (f-1)*20.4\ \text{msec} + 107\ \mu\text{sec}}$$

the above equation could be approximated using the following:

$$\text{steps/sec} = \frac{12500}{(256-r)} \qquad \text{for } 1 \leq r \leq 200$$

$$\text{steps/sec} = \frac{12500}{(257.344-r)} \qquad \text{for } 201 \leq r \leq 255$$

TABLE V RATE TABLE

PARAMETER	FORMULA	EMPERICAL
0	48.8	48.8
25	54.1	54
50	60.7	60
75	69.0	69
100	80.1	80
125	95.4	95
150	117.9	117
175	154.3	154
200	217.9	217
205	238.8	239
210	264.0	264
215	295.2	296
220	334.7	335
225	386.5	387
230	457.1	457
235	559.4	560
240	720.7	721
245	1012.6	1013
250	1702.0	1702
251	1970.4	1970
252	2339.0	2339
253	2877.0	2901
254	3730.0	3359

Rates for a Factor of 1 (**F 1)**)

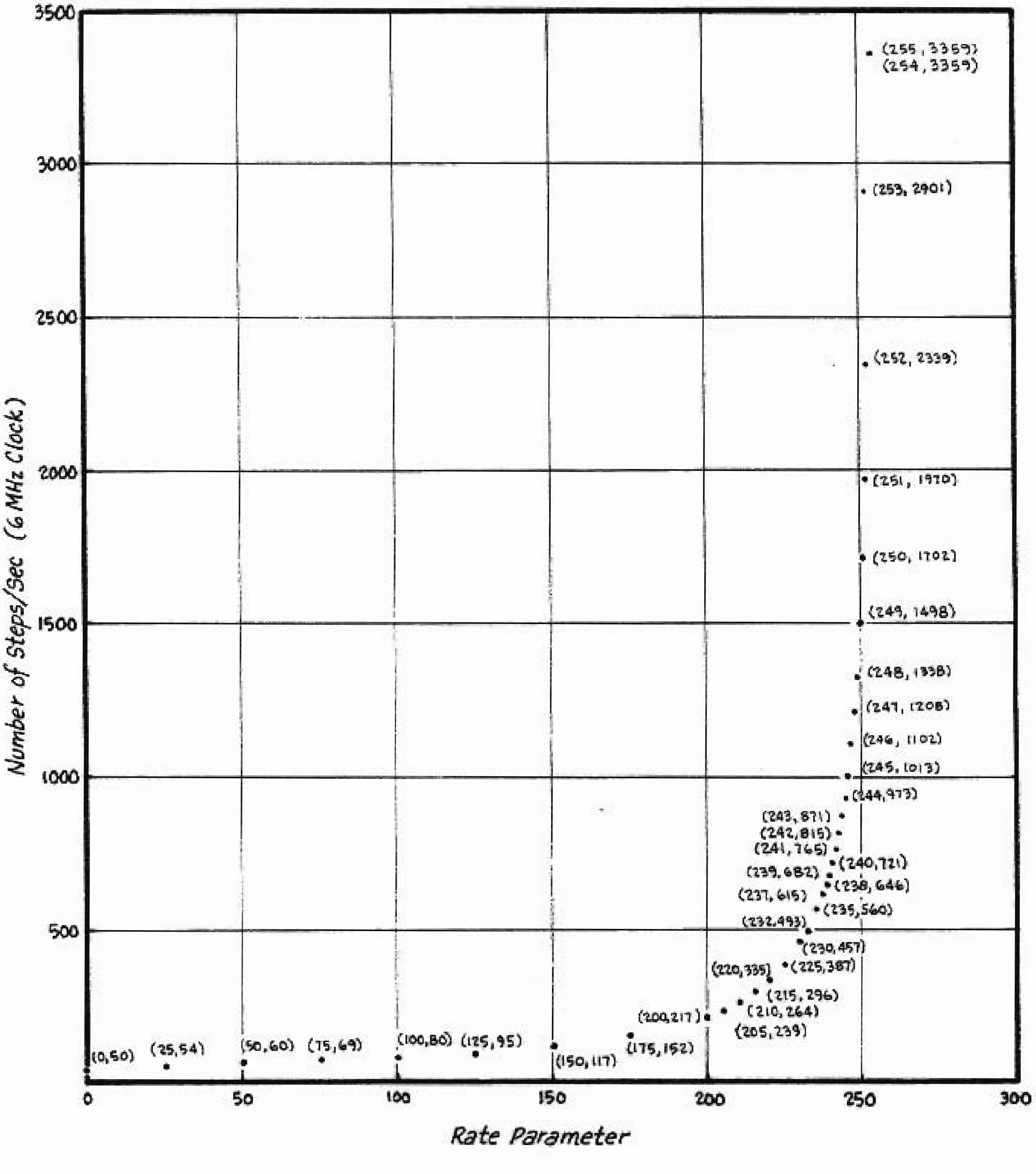


Figure 7.2 Step Rate vs. Rate Parameter.

RATE DIVISOR FACTOR

The stepper speed range discussed above assumes a rate divisor factor of 1. In this case the lowest rate of speed is 48.8 steps/sec. For many applications this may be too fast. The rate FACTOR command, F f), allows the user to divide the step rate by the factor f where 1≤f≤255. In most cases the rate parameter, r, should be set equal to 1 if a rate factor, f, unequal to 1, is to be used. The use of the rate factor is shown in Table VI.

TABLE VI STEP RATE FOR VARIOUS RATE FACTORS

(rate parameter r = 1)

RATE DIVISOR FACTOR	STEPS/SEC	SEC/STEP
1	48.8	
2	24.4	
3	16.2	
4	12.2	
5	9.7	
10	4.8	
20	2.4	
24	2.0	
48	1.0	
99	0.50	2.00
100	0.50	2.05
150	0.33	3.07
200	0.25	4.09
245	0.200	4.999
250	0.20	5.10

SPECIAL RATES

Two special rates are easily achieved via a combination of rate parameter and factor (assuming 6 MHz crystal).

RATE		PARAMETER r	FACTOR f
steps/sec	sec/step		
	1.000	48	50
60.0		49	1

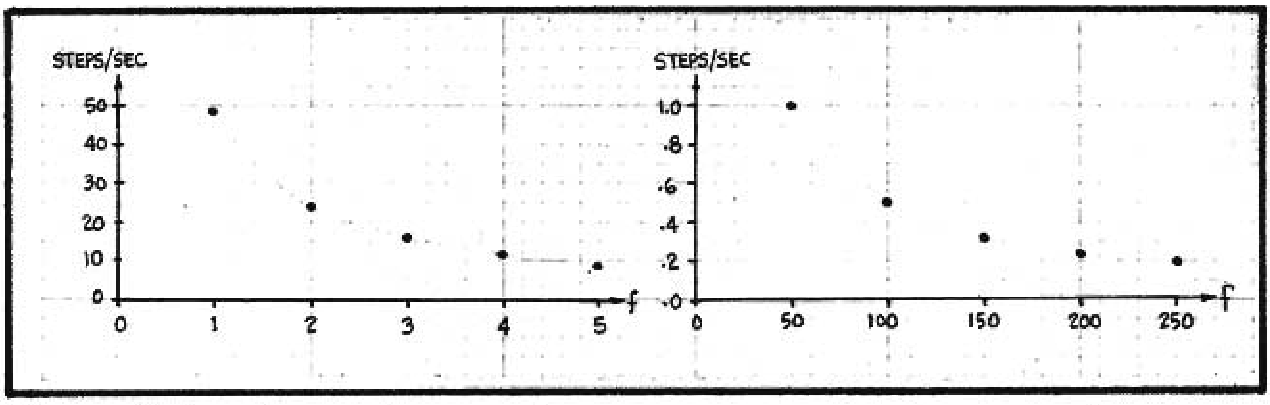


Figure 7.3 Plot of steps/sec versus rate factor with rate parameter equal to 1 and 6 MHz crystal.

VARIABLE RATE CONTROL

The flexibility of the CY500 allows the user to achieve variable rate control using external frequency sources in several ways. The simplest scheme use the JOG command, J, to place the CY500 in the External Start/Stop mode. Two possibilities exist here:

	XSS pin	Trigger
Case 1	step when low	always low
Case 2	always low	step when low

RAMPING MODE OF OPERATION

The CY500 will ramp up to a maximum specified rate and then ramp down to stop at a specified position or number of steps. The Slope command, S s), should be the last command encountered before the G command begins stepping.

During acceleration, the internal rate parameter will vary according to the following formula:

$$\boxed{r = (n-1)s + 1 \qquad \text{for step } n = 1,2,3,\ldots}$$

which means that the rate of the first step will be 1, with the value of Slope added to each succeeding step. This procedure continues until the specified maximum rate has been reached.

CONSTRAINTS ON SLEW MODE OPERATION

if R = max step rate
S = slope parameter
N = number of steps

I. $\frac{R}{S} < \frac{N}{2}$

II. $S < 255-R$ or $R \le nS \le 255$ where n = integer

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 slope command followed by the GO command. The following timing diagrams illustrate the effect of various slope parameters on the stepping behavior.

Note in figure 7.5 that when S=80 and R=240 the rate is incremented three times before R is reached while S=60 and R=240 (figure 7.6) requires four increases in the step rate to reach R=240. Note in figure 7.7 that when S=59 the rate increases for four steps, however since constraint II (above) is violated (59<255-240 is not true) the next rate is in error and the motion is erratic.

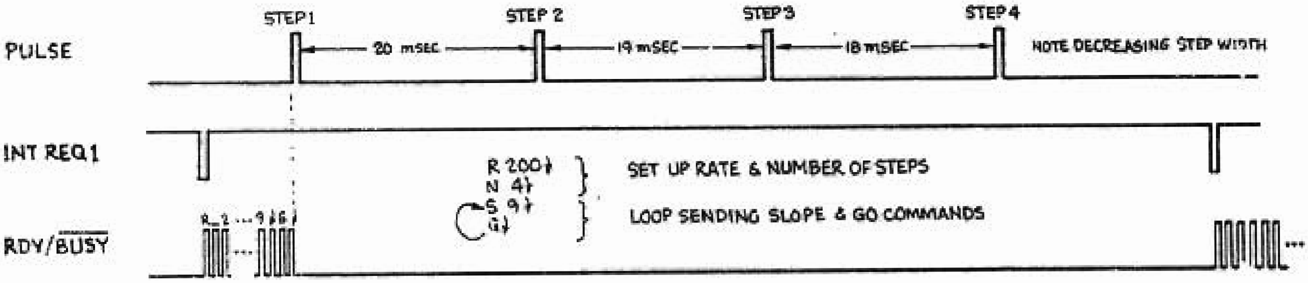

STEP 1
STEP 2
STEP 3
STEP 4
PULSE
20 mSEC
19 mSEC
18 mSEC
NOTE DECREASING STEP WIDTH
INT REQ1
R 200↵
N 4↵
S 9↵
G↵
SET UP RATE & NUMBER OF STEPS
LOOP SENDING SLOPE & GO COMMANDS
RDY/$\overline{\text{BUSY}}$
R_2 ... 9↵G↵

Figure 7.4 Ramp Rate Timing Example: R 200↵ N 4↵ S 9↵ G↵

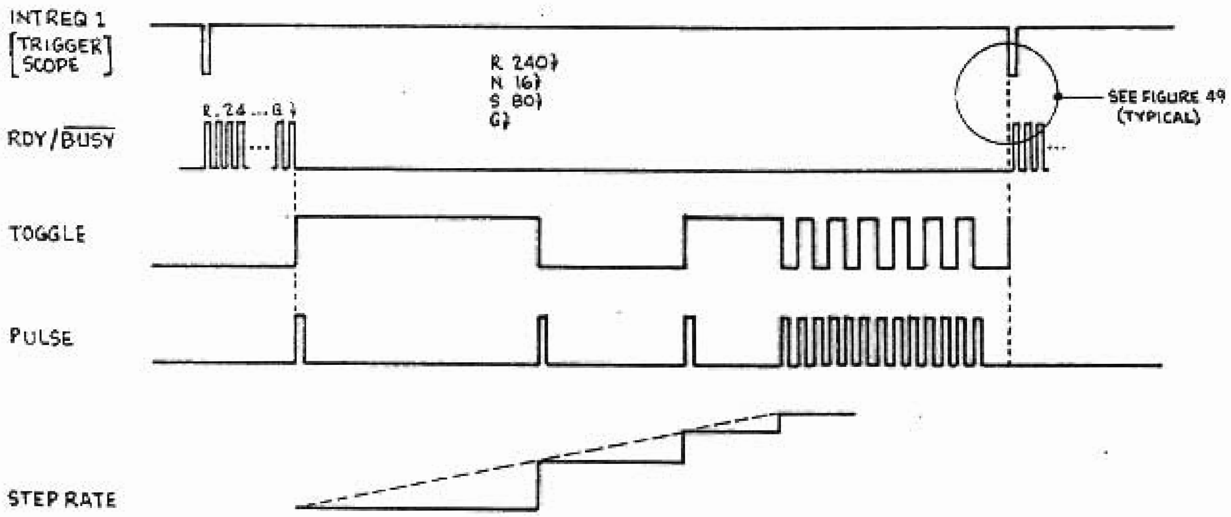


Figure 7.5 Ramp Rate Timing Example: R 240} N 16} S 80} G}

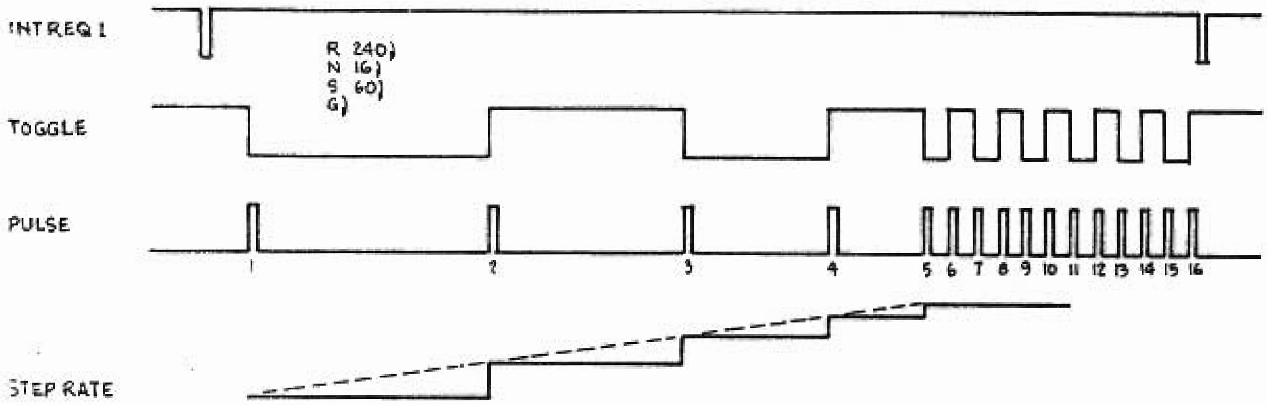


Figure 7.6 Ramp Rate Timing Example: R 240} N 16} S 60} G}

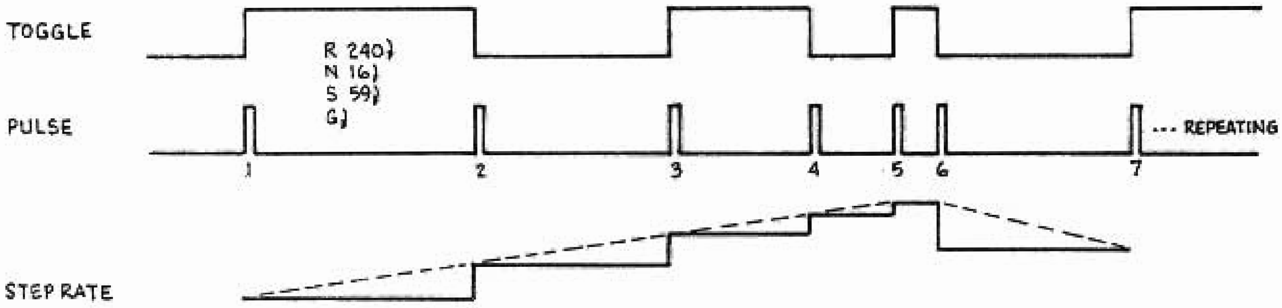


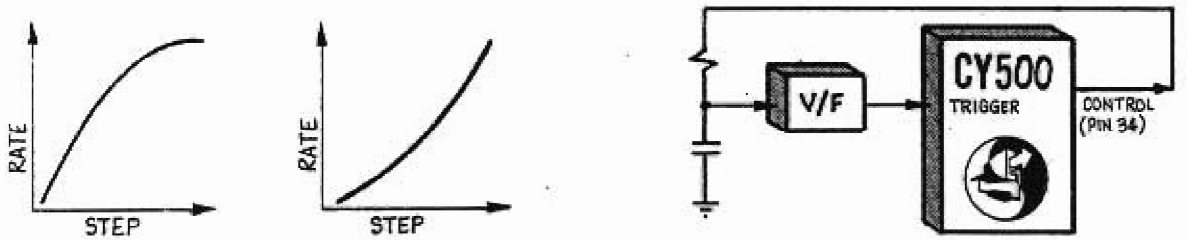
Figure 7.7 Ramp Rate Timing Example: R 240} N 16} S 59} G}
using invalid parameters (see previous text)

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 accellerating 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 position. The CY500 Stored Program Stepper Motor Controller provides for this mode of operation via the SLEW MODE command. This command sets the mode and also enters the appropriate parameter. The parameter is the RAMP RATE or SLOPE defined as the change-in-rate/step. The rate factor, f, is automatically set to 1 (for maximum velocity). The maximum rate desired is specified by the RATE command as usual, and the number of steps desired is specified by the **N** command as usual. A typical instruction sequence is shown below.

```
N 1095)    ;set number of steps = 1095

R 220)     ;set maximum rate = 335 steps/sec (see Table III)

S 12)      ;change in rate parameter per step
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 rate parameter per step. Note that at the higher step rates, the dependence of step rate on rate parameter is non-linear as shown in Table III. Thus the ramped operation is generally satisfactory at low or medium speeds but may require external timing at higher slew rates. An example of such timing is shown in figure 7.8. 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.



a.) Ideal   b.) Actual   c.) External Control of Ramp up

Figure 7.8 Actual and Ideal acceleration curves and suggested external ramp-up control circuitry.

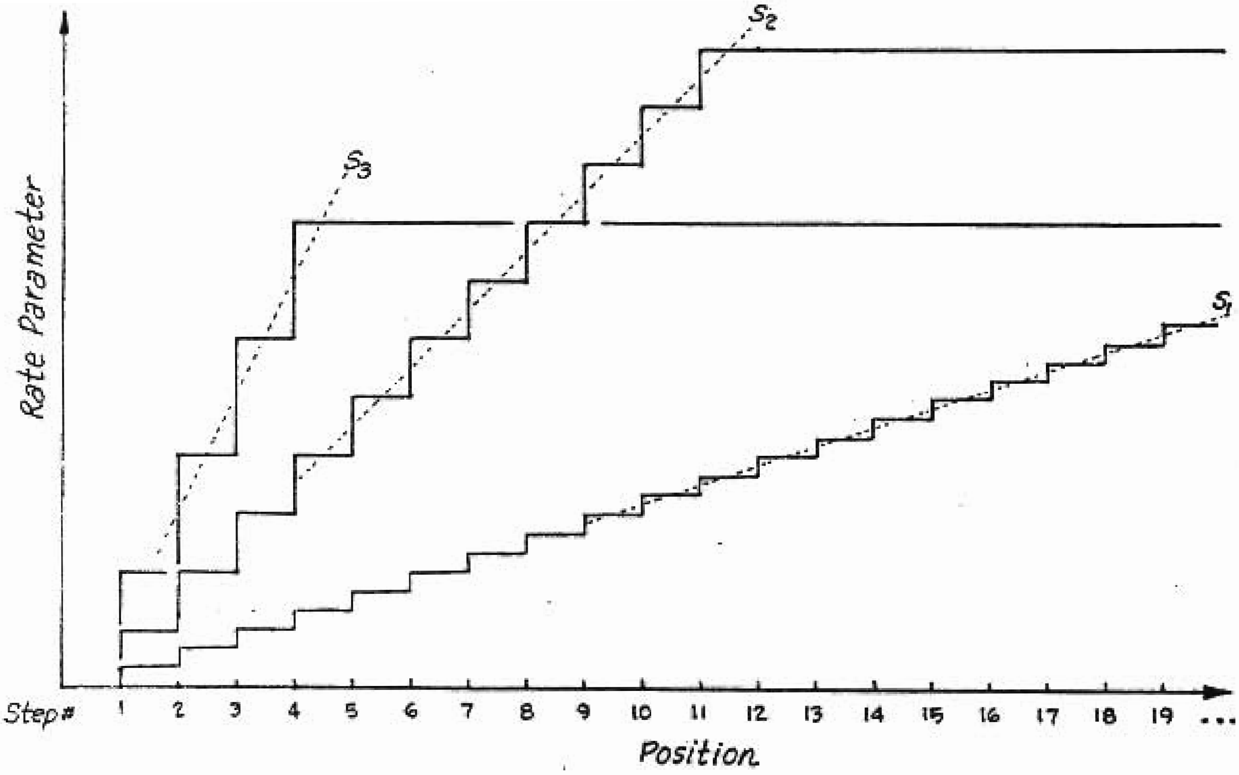


Figure 7.9 Variation of Rate Parameter with Position for three different slopes.

Slew Mode Example

Consider a command sequence to cause the CY500 controller to accelerate the motor with a slope that increases the rate parameter by 5 with each step taken until a maximum rate parameter of 180 is reached (which corresponds to about 180 steps/sec--see Table III), and then decelerate from this maximum speed to stop 513 steps from the start position. To enter this command sequence into the CY500 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 CY500.

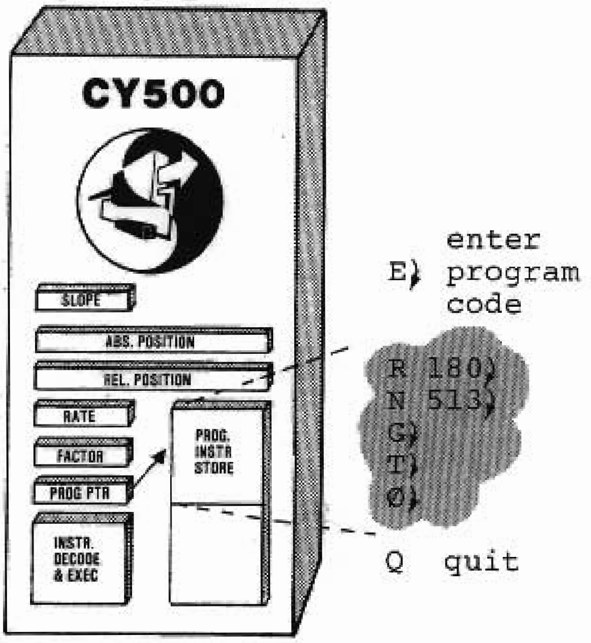


Figure 7.10
CY500 Programming Example.

CLOSED LOOP CONTROL

The constant slope acceleration implemented in the CY500 controller allows higher step rates than would be achievable with 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 CY500. Although the term "closed loop control" often indicates a rather high level of complexity and circuit analysis, the CY500 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 CY500. As shown in figure 7.11, 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.

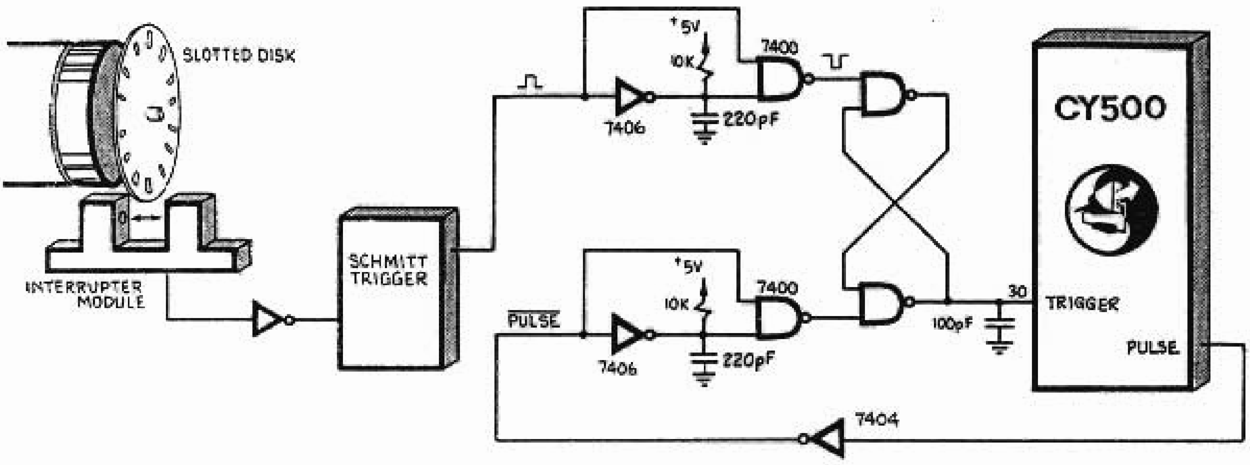


Figure 7.11 An interrupter module with slotted disk provides position feedback to assure that the maximum step rate never exceeds the motor's ability to keep up with the controlled fields, and the motor can accelerate at its maximum rate. Note that some interrupter modules include a Schmitt Trigger to produce a TTL output that can be input directly to the CY500.

8 ELECTRICAL SPECIFICATIONS 8

ABSOLUTE MAXIMUM RATINGS:

Ambient Temperature under bias...........0°C to 70°C
Storage Temperature......................-65°C to +125°C
Voltage on any pin with respect to GND...-0.5v to +7v
Power Dissipation........................1.5 watts

TABLE VII DC OPERATING CHARACTERISTICS

TA = 0°C to 70°C, Vcc = +5v±10%

SYMBOL	PARAMETER	MIN	MAX	UNIT	REMARKS
ICC	pwr supply current		80	mA	
VIH	input high level	2.0	Vcc	V	(3.8v for XTAL,RESET)
VIL	input low level	-.5	0.8	V	(0.6v for XTAL,RESET)
ILO	data bus leakage		10	µA	high impedance state
VOH	output hi voltage	2.4		V	IOH = -40 µA
VOL	output low voltage		.45	V	IOL = 1.6 mA
FCY	crystal frequency	1	6	MHz	

ELECTRICAL CONVENTIONS

All CY500 signals are based on a positive logic convention, with a high voltage representing a 1 and a low voltage representing a Ø. Signals which are active low are indicated by a bar over the pin name, i.e., $\overline{\text{RESET}}$.

All input lines except Write, Abort, and pin #39 include 50K ohm pull-up resistors. If the pins are left open, the input signals will be high.

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 CY500. This is accomplished via the use of a 1 ufd. capacitor as shown in figure 8.1. Reset must be low for 10 msec after power-up. Once the CY500 is running, Reset need only be low for about 11 usec (6 MHz crystal).

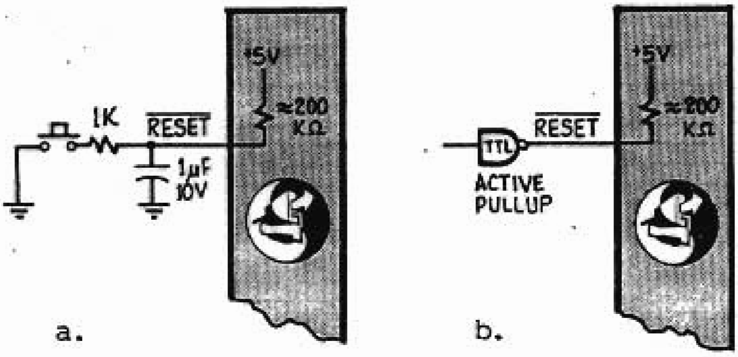


Figure 8.1 a)Reset Circuitry.
b)External Reset.

CLOCK CIRCUITS

The CY500 may be operated with crystal, LC, or external clock circuits. These three circuits are shown in figure 8.2. All timing discussed in this manual assumes a 6MHz series resonant crystal such as a CTS Knights MP060 or Crystek CY6B, or equivalent. The CY500 will operate with any crystal from 1 to 6 MHz including a standard 3.58 MHz TV color burst crystal. As noted elsewhere, stepping rates must be scaled by f/6, where f is the crystal frequency in MHz.

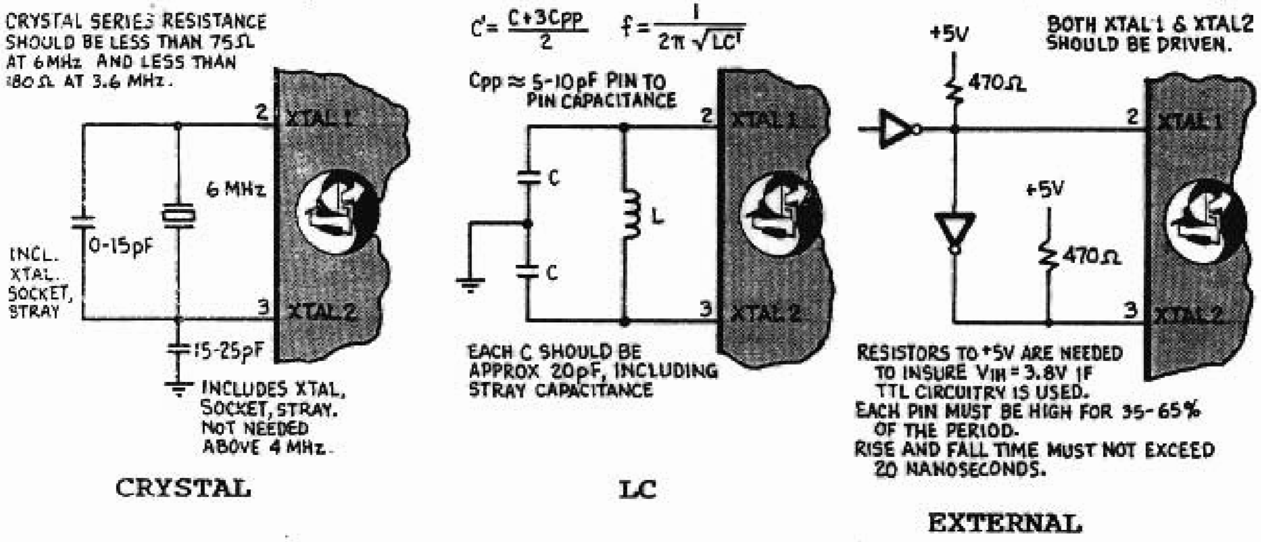

CRYSTAL SERIES RESISTANCE SHOULD BE LESS THAN 75Ω AT 6MHz AND LESS THAN 180Ω AT 3.6 MHz.
2
XTAL 1
6 MHz
INCL. XTAL. SOCKET, STRAY
0-15pF
3
XTAL 2
15-25pF
INCLUDES XTAL, SOCKET, STRAY. NOT NEEDED ABOVE 4 MHz.
CRYSTAL
C' = (C+3Cpp)/2
f = 1/(2π√LC')
Cpp ≈ 5-10pF PIN TO PIN CAPACITANCE
C
L
C
2
XTAL 1
3
XTAL 2
EACH C SHOULD BE APPROX 20pF, INCLUDING STRAY CAPACITANCE
LC
+5V
BOTH XTAL 1 & XTAL2 SHOULD BE DRIVEN.
470Ω
2
XTAL 1
+5V
470Ω
3
XTAL 2
RESISTORS TO +5V ARE NEEDED TO INSURE VIH = 3.8V IF TTL CIRCUITRY IS USED.
EACH PIN MUST BE HIGH FOR 35-65% OF THE PERIOD.
RISE AND FALL TIME MUST NOT EXCEED 20 NANOSECONDS.
EXTERNAL

Figure 8.2 Clock Circuits for CY500.

CIRCUITS AND EXAMPLES

TEST DEMONSTRATION CIRCUIT

The circuitry shown in figure 9.1 provides a simple setup that allows use of a strobeless ASCII keyboard to control the CY500. 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 very slow rates specified so that the transitions are visible to the user if LEDs are used on the outputs.

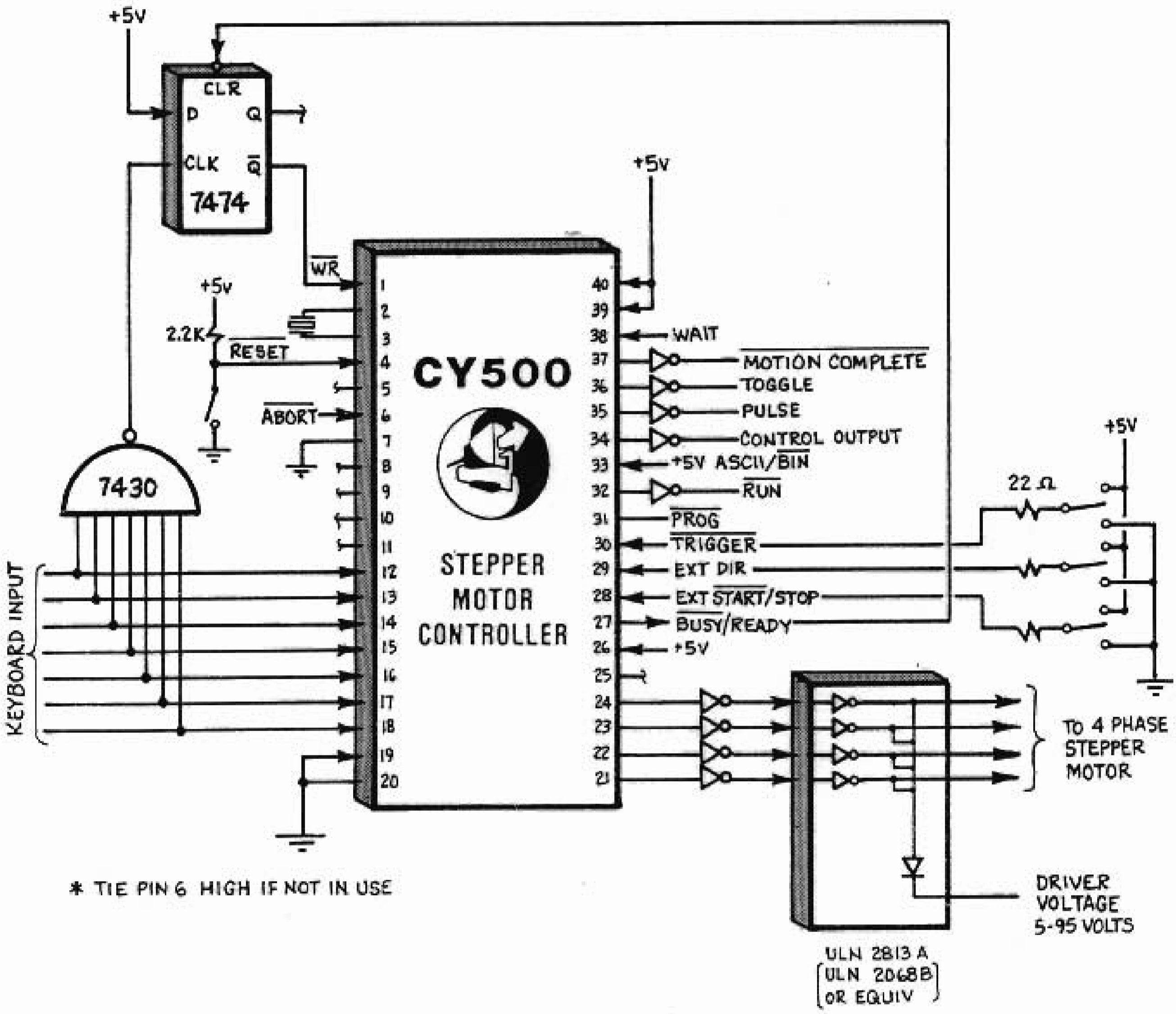


Figure 9.1 Test Demonstration Circuit

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 is also available.

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 CY500, 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:

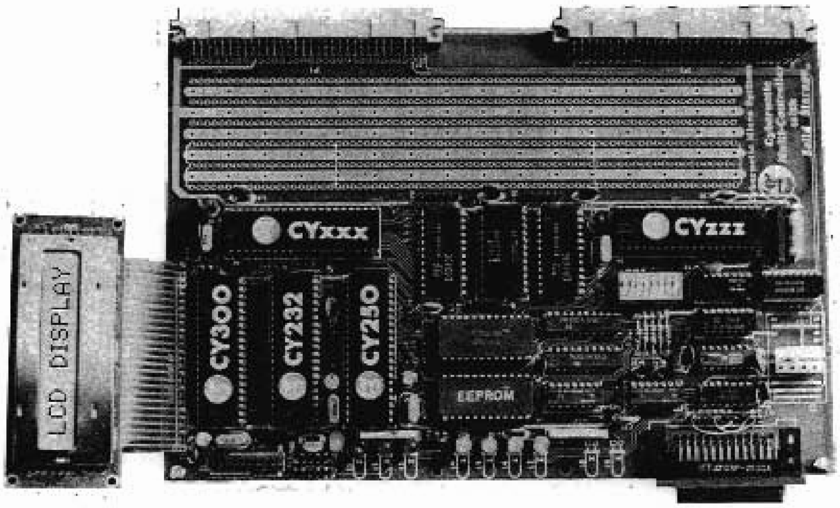


Figure 9.2 CYB-002 Multi-purpose Control Board

HANDSHAKE PROTOCOL

All commands and data transmitted from the master processor to the CY500 peripheral processor are sent asynchronously with complete handshaking performed. The master processor waits for the CY500 READY line to go HIGH before sending the active LOW WRITE signal, WR. In the parallel data transfer mode (pin #39 = HI) the data may be placed on the bus at any time prior to the HIGH-to-LOW transition of WR. The data should be stable on the bus until the CY500 RDY line goes LOW, indicating that the transfer has been acknowledged and that the CY500 is BUSY processing the command or data. The master then brings WR to the HIGH state. The next transfer can occur as soon as RDY/BUSY returns HIGH. The sequence described is shown in Figure 9.3.

```
;
;
SENDCHAR:
IN    STATUSPORT
ANI   MASK
JZ    SENDCHAR    ;WAIT TIL READY
MOV   A,C
ANI   7FH
OUT   DATAPORT    ;WITH WR LOW
;
;
BUSY?:
IN    STATUSPORT
ANI   MASK
JNZ   BUSY?       ;WAIT FOR BUSY
MVI   A,0FFH
OUT   DATAPORT    ;WR HIGH
RET
;
```

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

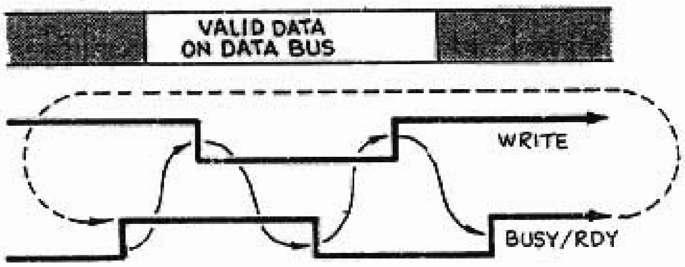


Figure 9.3 Data Transfer Handshake Sequence

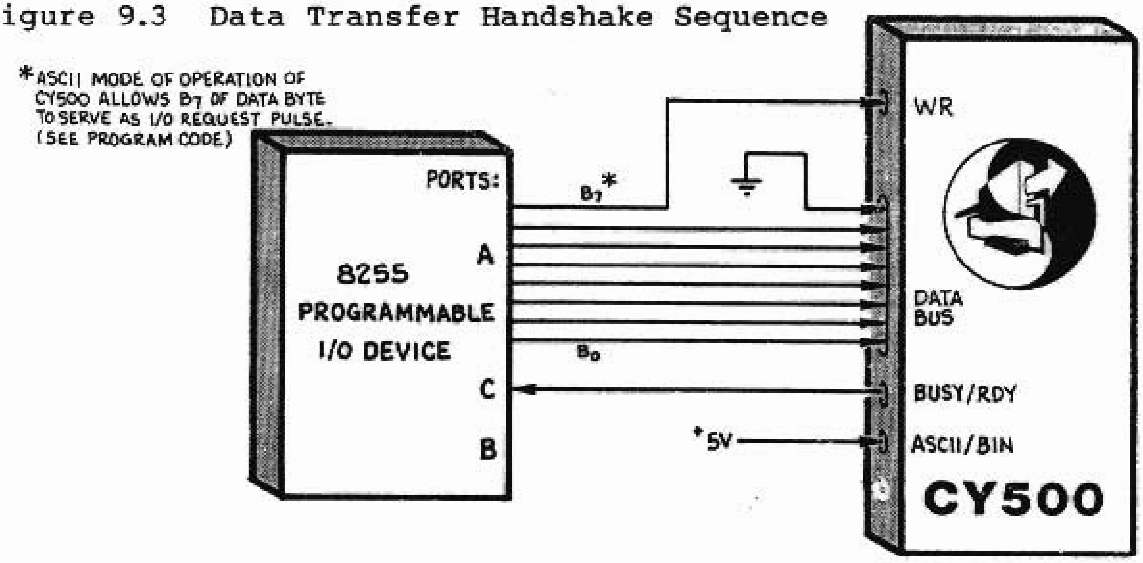

*ASCII MODE OF OPERATION OF CY500 ALLOWS B7 OF DATA BYTE TO SERVE AS I/O REQUEST PULSE. (SEE PROGRAM CODE)
PORTS:
A
8255
PROGRAMMABLE
I/O DEVICE
C
B
B7*
B0
+5V
WR
DATA
BUS
BUSY/RDY
ASCII/BIN
CY500

Figure 9.4 Example Interface to CY500 using 8255 PIO.

In the example shown in Figure 9.4 the CY500 is operating in the PARALLEL ASCII input mode. In this mode, bit 7 is always zero and b7 line of the CY500 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 WRITE strobe, thus allowing only one 8 bit output port to suffice. The SENDCHAR routine shown in Figure 9.3 demonstrates the coding used to achieve this. Of course, a separate port line may be used to generate WR if this is desired. If the CY500 is operated in the PARALLEL/BINARY mode, all 8 data bus lines are used, and a separate WR line is required. Note that in the example shown, use is made of the fact that the data and the WR signal may be applied simultaneously in parallel operation.

OPERATION OF SEVERAL CY500s USING A COMMON DATA BUS

In systems where multiple CY500s are to be controlled by a host computer it is possible to use one eight-bit output port to establish a common data bus for sending instructions to the CY500s. Each of the separate RDY lines (pin 27) of each CY500 must be monitored individually and each WR line (pin 1) must be activated separately. This technique effectively uses the WR line as a chip select (CS). A CY500 will ignore all bus information if its WR line is inactive.

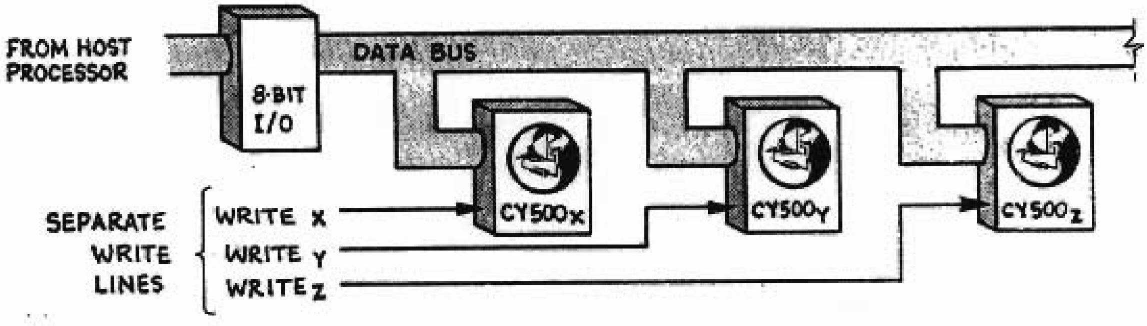
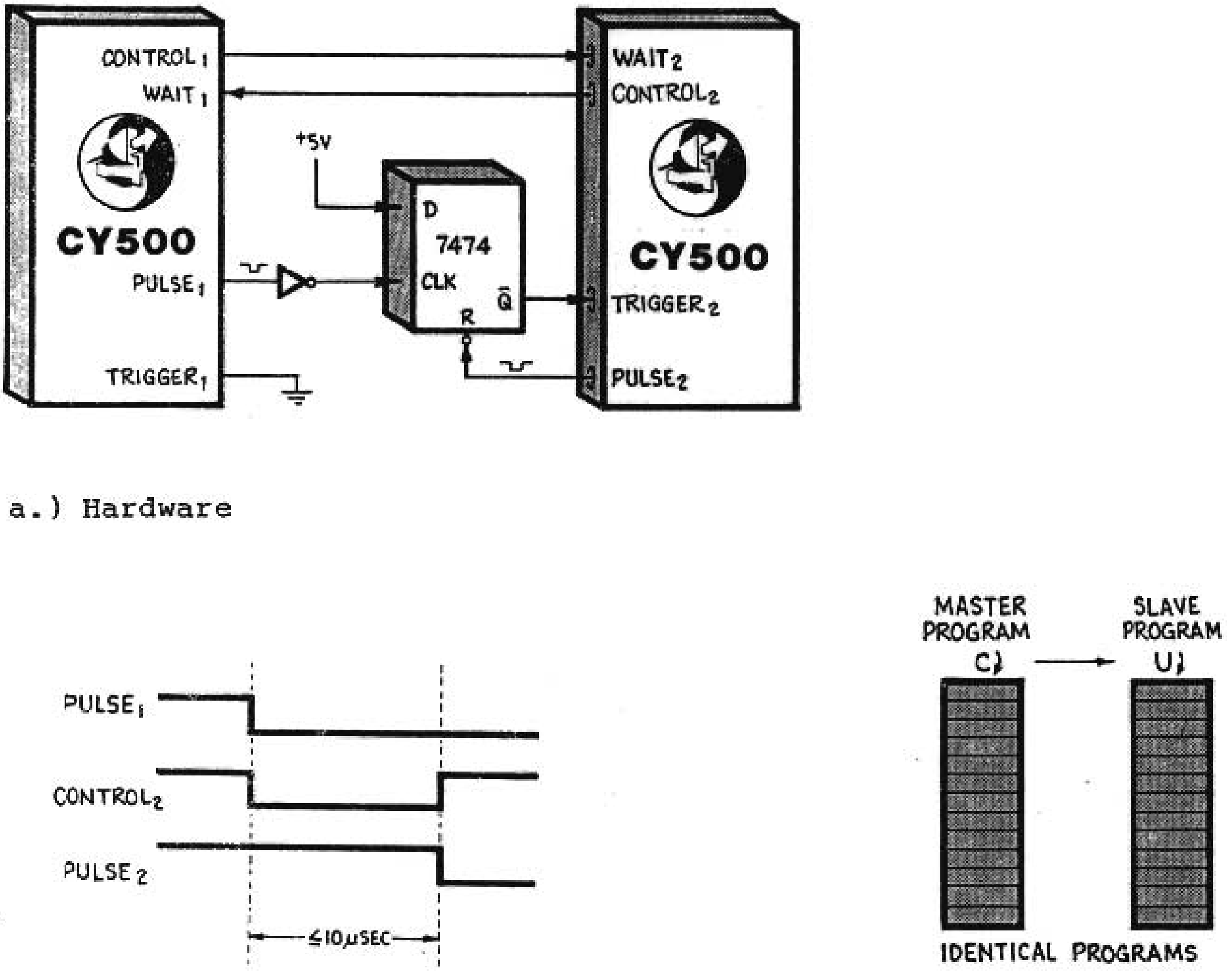


Figure 9.5 CY500s share a common data bus by using separate write lines for chip select.

SYNCHRONIZATION OF TWO CY500s

Two CY500s executing the same program may be synchronized as shown in figure 9.6. The master controller can control the WAIT line of the slave CY500 via the BITSET or CLEARBIT commands. The slave CY500 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 CY500 when it receives the (second) DOITNOW command. Both CY500s then proceed to the next instruction and are synchronized as shown in figure 9.6b to within approximately 10 microseconds. Note that when the two programs are not identical, then the master can also wait for the slave to execute its own CLEARBIT instruction and thereby achieve a more general synchronization.



a.) Hardware

b.) Timing Diagram

c.) Software

Figure 9.6 Synchronization of two CY500s.

EXAMPLE PROGRAMS AND WAVEFORMS

Figure 9.7 provides timing relations for a command sequence that inputs the parameters and executes a **G** command to begin stepping. The WR, RDY, and RD signals are related to the data bus and several outputs are shown as a function of the trigger input.

```
COMMAND MODE INPUT SEQUENCE:

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

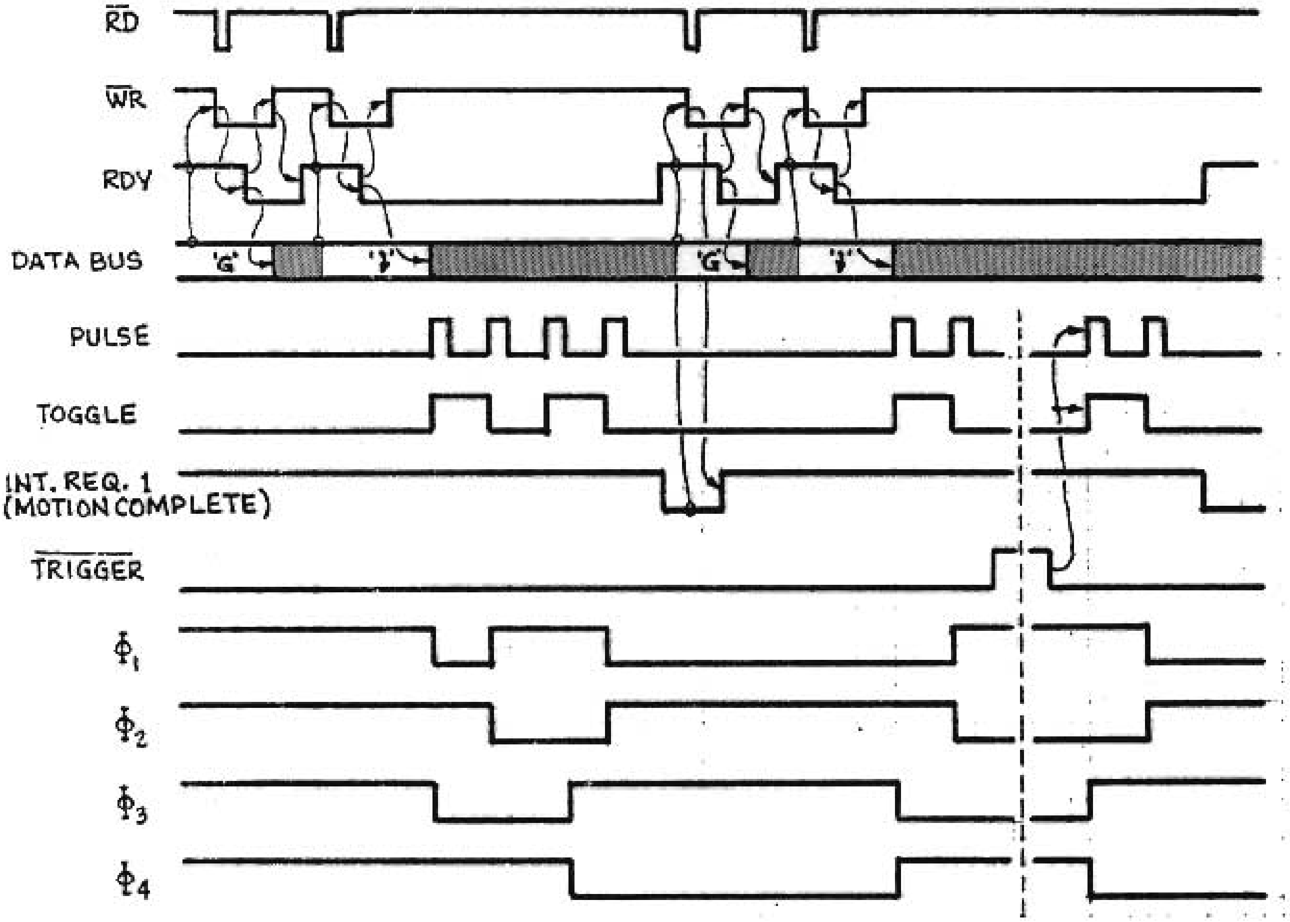
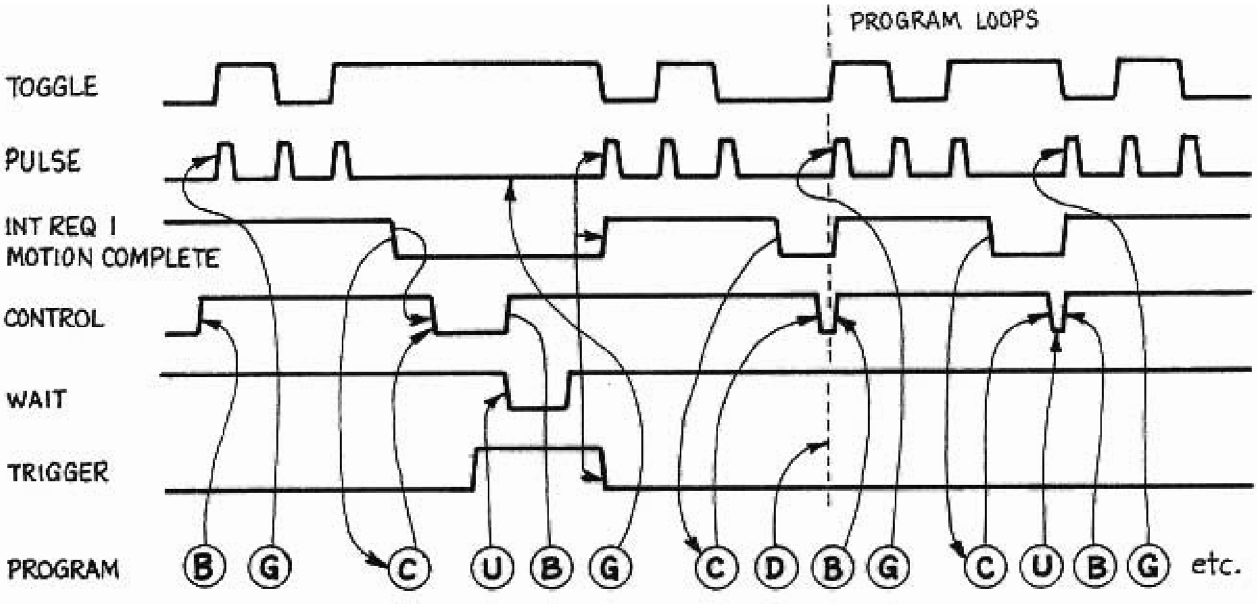


Figure 9.7 Timing Diagram for Commands

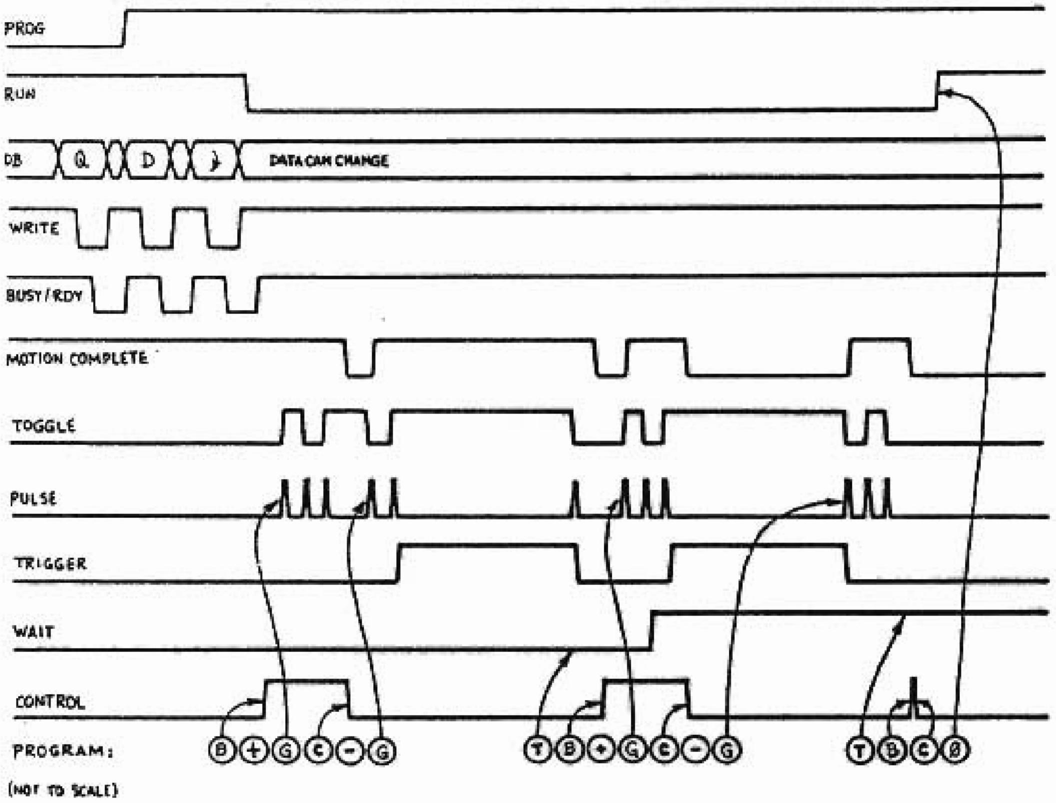


```
  preset: R 20}
          F 80}
          N 3}
          C}

   enter: E}

 program: B}<==
          G}
          C}
          U}
          B}
          G}
          C}
          D}==

    quit: Q

     run: D}
```

The timing sequence for a typical program is shown in figure 9.8. 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 9.1. The program entered sets the control output (pin #34), then takes three steps, clears the control output, and waits for the WAIT line (pin #38) to go low when the wait UNTIL instruction is executed. As shown, the trigger line has gone high, and the CY500 waits for this line to go low before stepping. Each time the three steps are completed the MOTION COMPLETE (INT REQ 1) output goes low and remains low until the next step is taken or until another command byte is received by the CY500. In the example, the output is again cleared and the program loops upon encountering the DOITNOW instruction.

Figure 9.8 Sample Program and Timing Diagram.

The use of loop TIL instruction is illustrated in figure 9.9 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 XSS/TIL line (pin 28) goes high, then fetches the next instruction. The effect of the trigger 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)
           B)   set output line          ]
PROGRAM    +)   set CW direction         |
CODE       G)   GO, begin stepping       |
           C)   clear ouput line         |
           -)   set CCW direction        |
           G)   GO, begin stepping       ]

           T)   repeat above prog Til XSS/TIL  = HI]

           B)   set output line                   ]
           C)   clear output line                 |
           Ø)   exit run mode, enter command mode]

QUIT:      Q
EXECUTE:   D)   DOITNOW
```

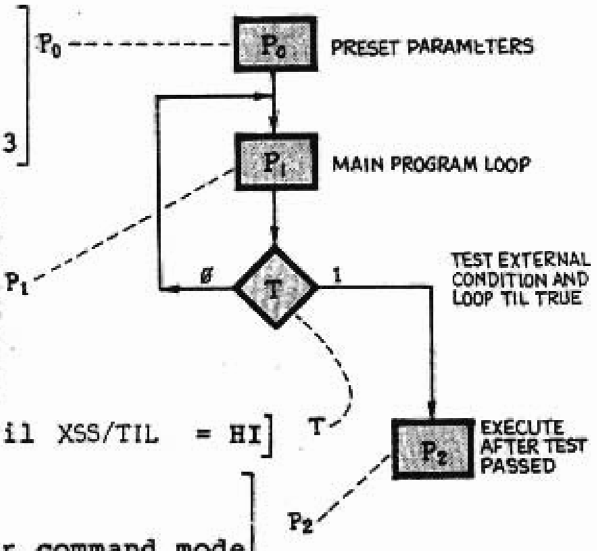

P0
P0
PRESET PARAMETERS
P1
P1
MAIN PROGRAM LOOP
T
Ø
1
TEST EXTERNAL CONDITION AND LOOP TIL TRUE
T
P2
P2
EXECUTE AFTER TEST PASSED

Figure 9.9 Timing and Control for Program Entry and Conditional Looping.

DRIVER CIRCUIT CONSIDERATIONS

The CY500 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 CY500. This circuit will take the logical signals generated by the CY500 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 CY500 Pulse and Direction signals. **This allows the CY500 to control three and five phase motors as well as the standard four phase designs.**

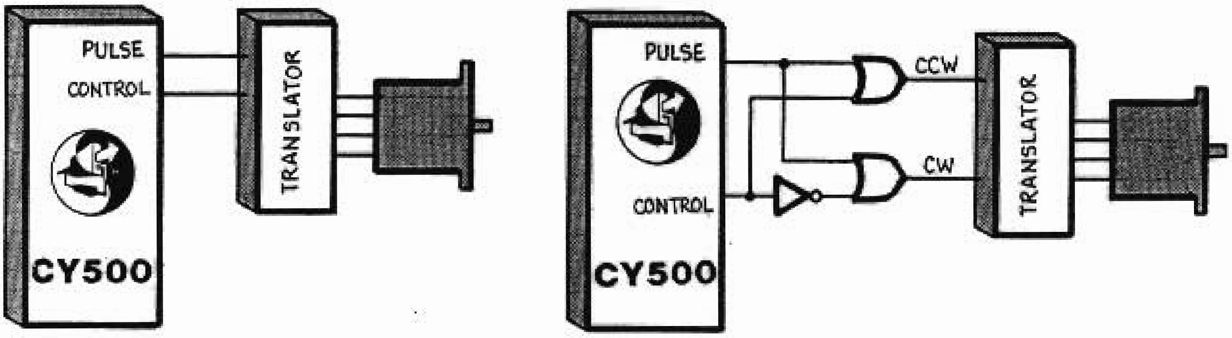


Figure 9.10 CY500 to Translator Driver connections.

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 CY500 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 CY500. 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 know 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.

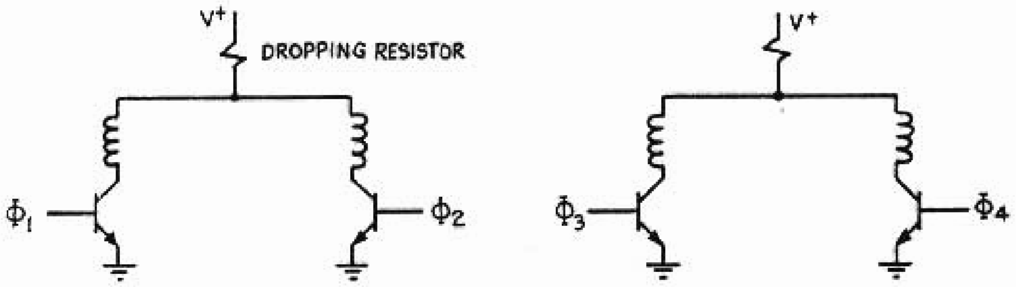


Figure 9.11 Unipolar driving circuits.

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.

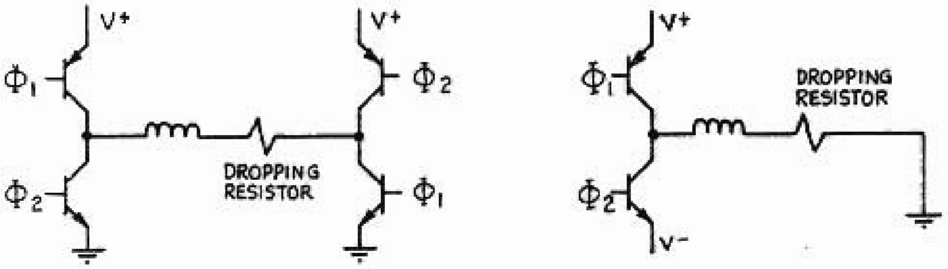


Figure 9.12 Bipolar driver designs.

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.

RS-232-C RECEIVE ONLY INTERFACE DESIGN

When the user wishes to communicate with the CY500 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 CY500.

The design shown here is a "receive only" type, meaning that the CY500 can only receive data, not transmit. 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 CY500, 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 CY500 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 CY500.

Whenever the UART receives a character, the data available line (DAV) goes high. This signal runs the WRITE line, indicating to the CY500 that a command character is ready. As the CY500 reads the character, the READ signal is used to put the character onto the CY500 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 WRITE line. Thus, the standard signals from the UART fully implement the two-line data transfer handshake used by the CY500.

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 preceeding 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.

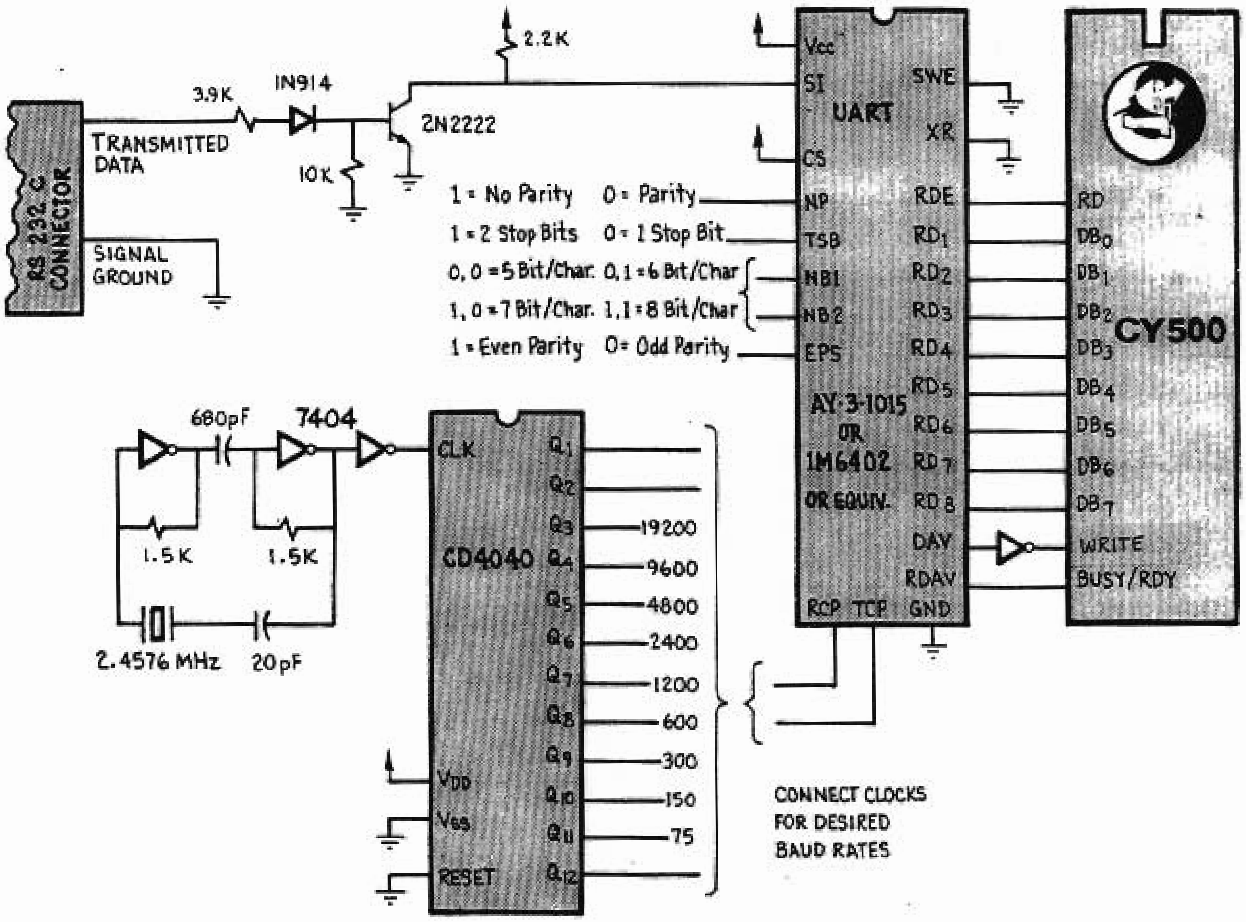


Figure 9.13 RS-232-C Interface Schematic.

The CY232 Parallel/Serial Network controller also enables the user to send data to the CY500 parallel device via a serial RS-232-C port. The actual CY232 to CY500 interface is very easy as shown in the schematic below. However, since the CY232 gives the ability to address multiple devices on a network, the CY232 address lines should be tied high or low to provide the CY500 with a specific address, and this address should be used when writing to the CY232/CY500. Also, multiple CY500s can be addressed this way by preceding each with a separate CY232 with a different address or by connecting multiple CY500s to a single CY232. In the second case, the CY232 address decoding logic should be combined with the CY232 DAV to generate a unique Write Strobe for each CY500 (see also Figure 9.5). The CY232 manual gives complete details on this interface.

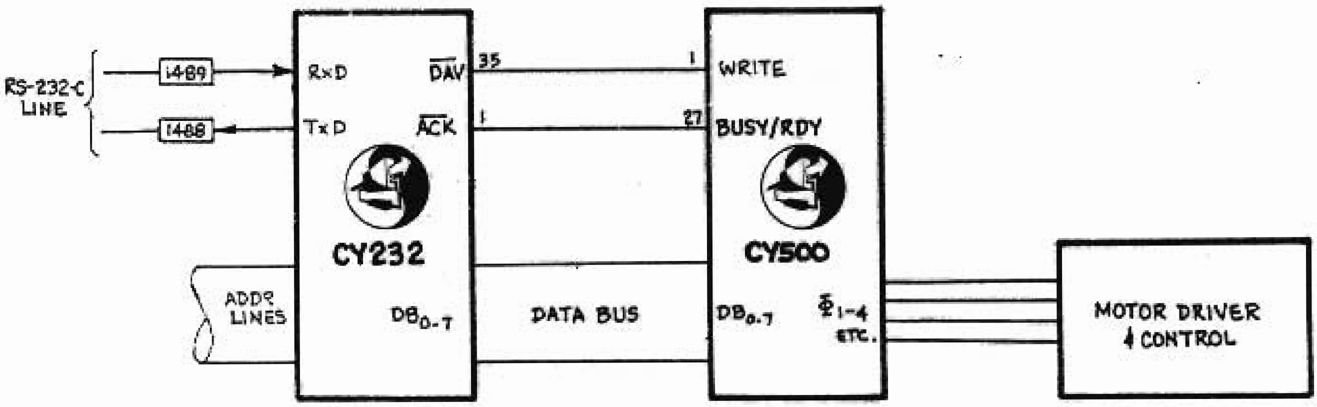


Figure 9.14 CY500 connections to CY232.

PROM STAND-ALONE INTERFACE DESIGN

When the CY500 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 CY500 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 CY500 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 CY500 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 CY500 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 CY500, so the data bus will have the byte ready when the CY500 reads it. When the CY500 becomes ready, with a high level on the BUSY/READY line, the 7400 nand gate generates a low output to the CY500 WRITE line. This will tell the CY500 that a command byte is available. The CY500 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 the WRITE line, 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 CY500 has finished processing the last command byte, it will go ready again, generating another WRITE strobe, and causing the CY500 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 CY500 READY signal from generating another WRITE strobe. The circuit stops clocking at this point, and stays frozen with WRITE 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 CY500 with a byte containing the 0FFH. Note that the 0FFH is not read into the CY500, 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 CY500 is operated in the ASCII mode. If the CY500 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 CY500. 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.

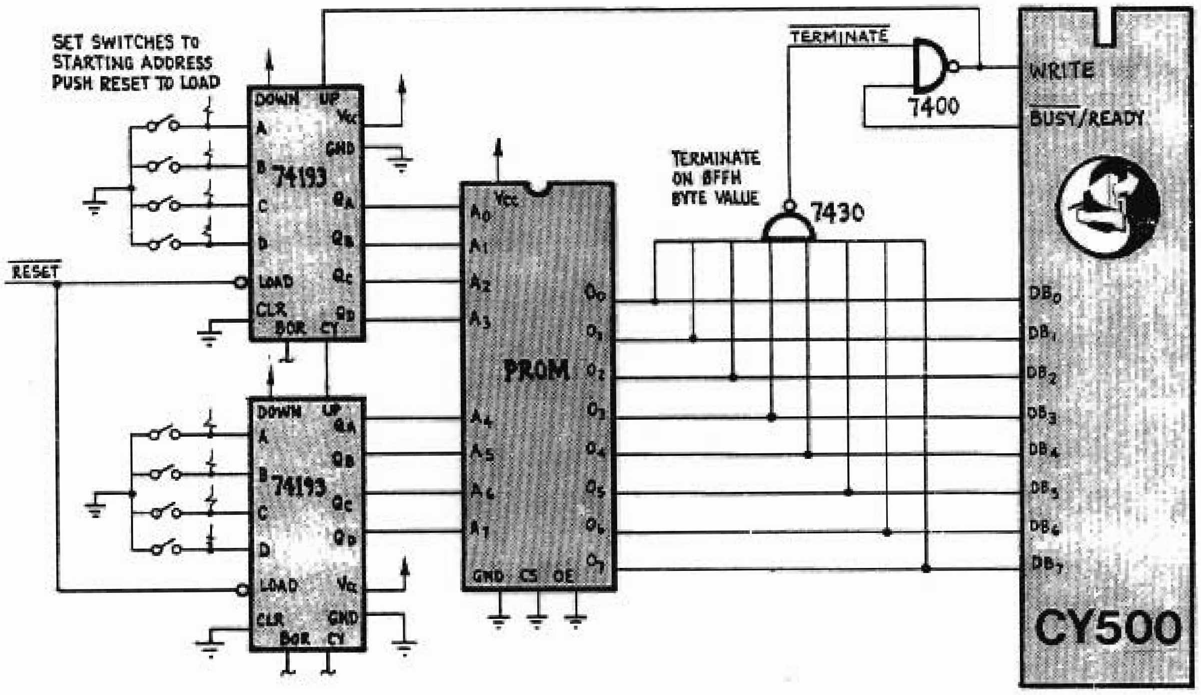


Figure 9.15 PROM Stand-alone Interface

EEPROM STAND-ALONE INTERFACE DESIGN

The CY250 Local System Controller will allow the user to interface the CY500 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 CY500s 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 9.16. More details on the EEPROM interface may be found in the CY250 manual and the CYB-002 manual.

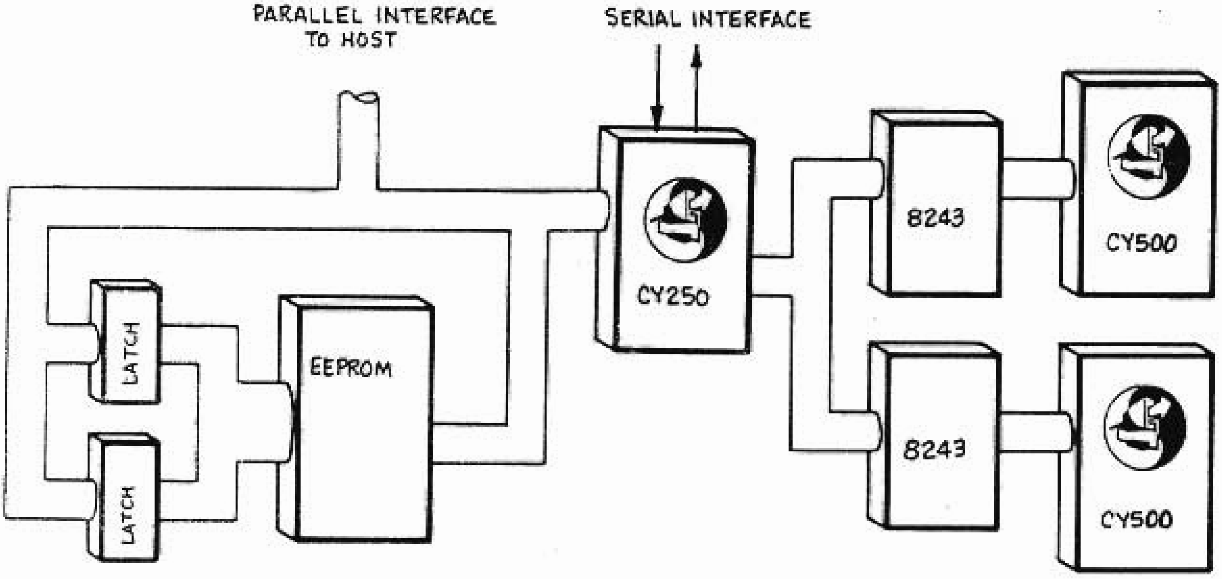


Figure 9.16 CY500 interface to EEPROM through CY250.

COMPUTER CONTROL OF CY500

The ability to control all of the CY500 control inputs and monitor all of the CY500 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 CY500 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. The setup is as shown below:

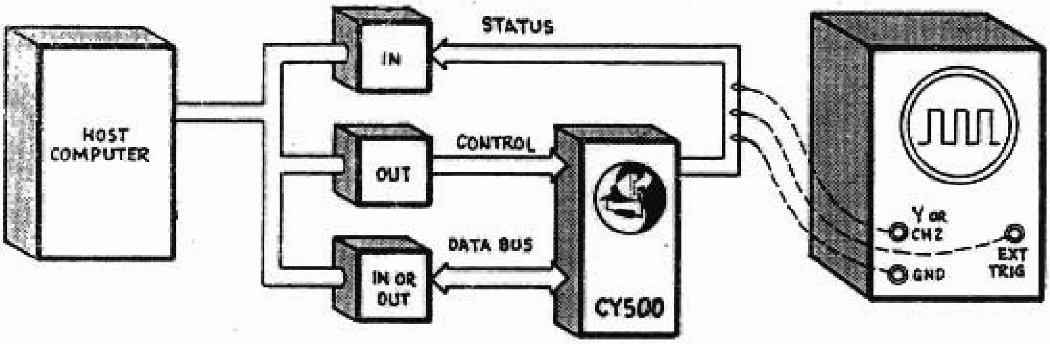


Figure 10.1 Preferred setup for Test/Display/Control of CY500 Stepper Controller.

By using a loop in the host computer (or in the CY500) the user can achieve a repetitive operation of the CY500 that allows easy display of CY500 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 (CY500 pin 37) and the CONTROL output (pin 34) serve well as external triggers.

ENTER/QUIT PROGRAM MODE

A key feature of the CY500 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 relevent (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 (ODH) 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 CY500 may be gained by presetting parameter values before entry and execution of the program. The host program may treat the CY500 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 CY500s are controlled by a single host.

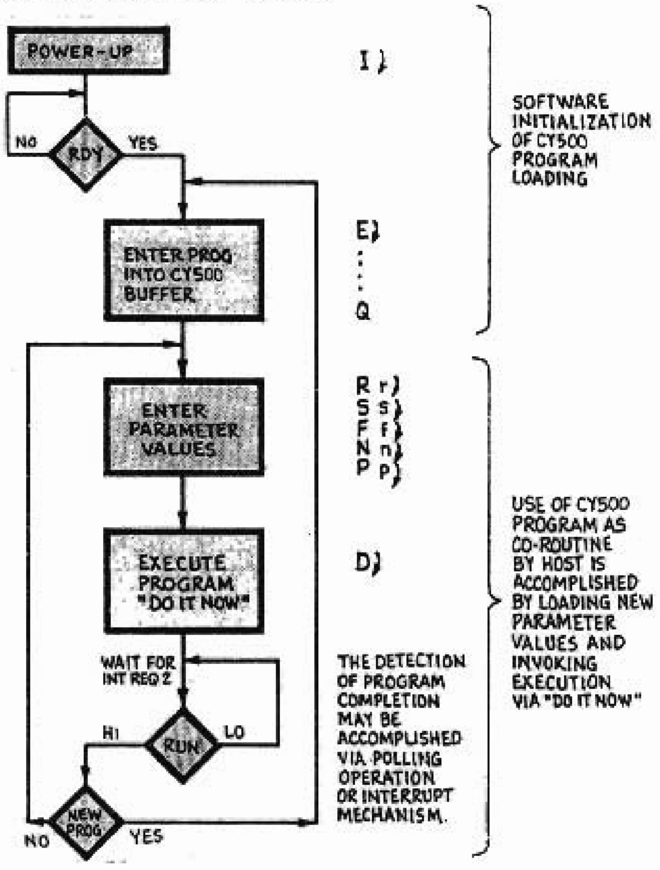

POWER-UP
NO
RDY
YES
ENTER PROG INTO CY500 BUFFER
ENTER PARAMETER VALUES
EXECUTE PROGRAM "DO IT NOW"
WAIT FOR INT REQ 2
HI
RUN
LO
NO
NEW PROG
YES
I}
E}
Q
R r}
S s}
F f}
N n}
P p}
D}
THE DETECTION OF PROGRAM COMPLETION MAY BE ACCOMPLISHED VIA POLLING OPERATION OR INTERRUPT MECHANISM.
SOFTWARE INITIALIZATION OF CY500 PROGRAM LOADING
USE OF CY500 PROGRAM AS CO-ROUTINE BY HOST IS ACCOMPLISHED BY LOADING NEW PARAMETER VALUES AND INVOKING EXECUTION VIA "DO IT NOW"

Figure 10.2 Operation of CY500 as co-routine

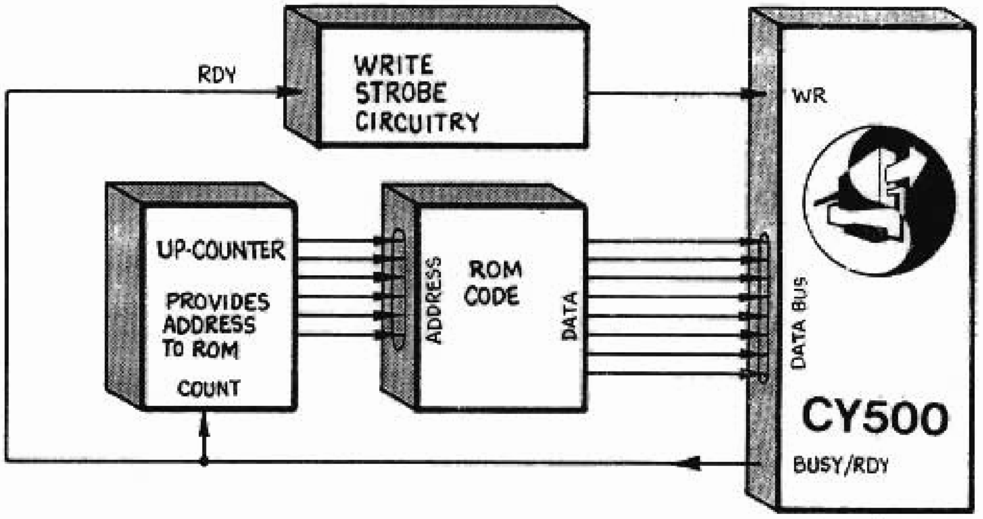


Figure 10.3 CY500 can receive commands and data from a ROM sequencer for many stand-alone applications not requiring a host microcomputer.

CY500 STAND-ALONE APPLICATIONS

The CY500 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 CY500, 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 CY500 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 10.3. See also Custom Device Generation in the first section of this manual.

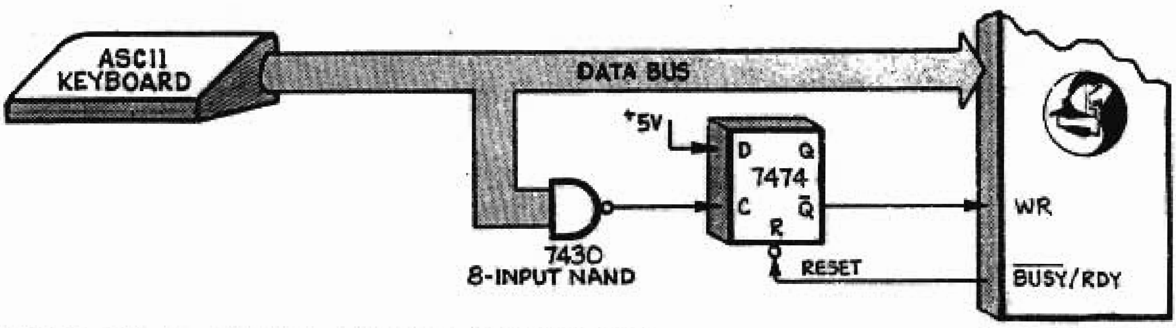


Figure 10.4 Write Strobe Generator.

PROGRAMMING EXAMPLES

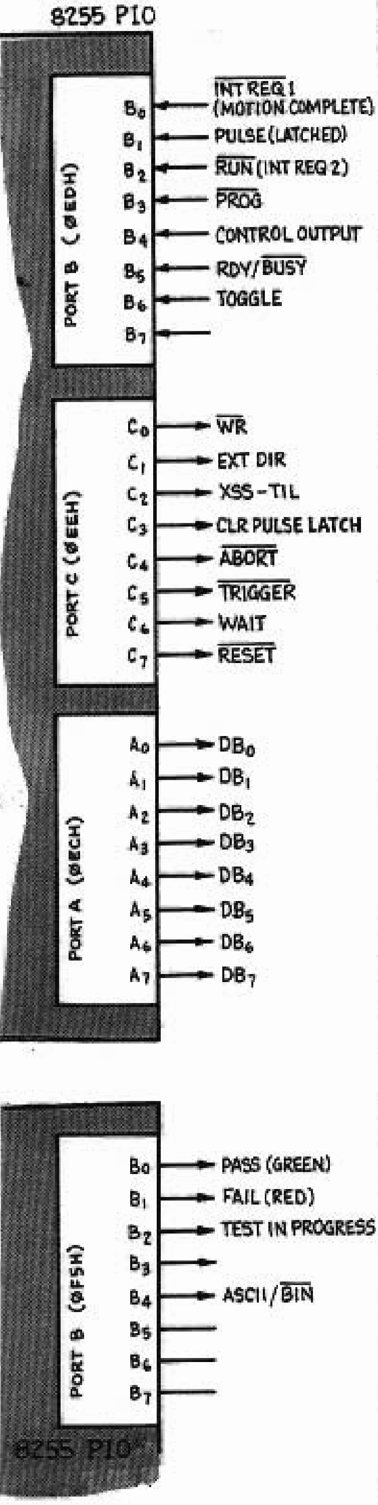
The following pages illustrate several programming examples, including waveforms and program listings. Programs are all written in 8080 Assembly Language, but the comments should allow those readers who are not familiar with the 8080 to understand what the various subroutines are doing. The programs were used on an SDK80 board, with the CY500 included in the wire wrap area.

We start with an equate table, indicationg how the CY500 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 CY500 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 CY500, 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 CY500 in ASCII mode as the Binary mode 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 VIII EQUATE TABLE AND 8255 PORT ASSIGNMENTS

The CY500 is connected to 8255-A4 ports OECH to OEEH on the Intel SDK80 board.

```
;
;    EQUATE TABLE
;
LEDS EQU OF5H       ;PASS/FAIL/TESTING
;                      LEDS ON PORT B=F5H
;
DATA EQU OECH       ;8255-A4-PORT.A
;                      CY500 DATA BUS
;
STATUS EQU OEDH     ;8255-A4-PORT.B
;                      CY500 STATUS PINS
;
MOTION EQU 1        ;B0 - MOTION COMPLETE
PULSE EQU 2         ;B1 - PULSE OUTPUT
RUNBAR EQU 4        ;B2 - RUN MODE PIN
PROGBAR EQU 8       ;B3 - PROG MODE PIN
BITOUT EQU 10H      ;B4 - CONTROL OUTPUT PIN
READY EQU 20H       ;B5 - READY/BUSY PIN
TOGGLE EQU 40H      ;B6 - TOGGLE OUTPUT
SWITCH EQU 80H      ;B7 - SWITCH TO START TEST
;
CONTROL EQU OEFH ;8255-A4-PORT.C - CY500 INPUT
WRBAR EQU 0         ;C0 - WRITE STROBE TO CY500
WRITEHI EQU 1       ;C0 = 1 - INACTIVE WRITE STATE
;
EXTDIR EQU 2        ;C1 - EXT DIRECTION CONTROL
HIEXTDIR EQU 3      ;C1 = 1
;
XSS EQU 4           ;C2 - EXTERNAL START/STOP PIN
HIXSS EQU 5         ;C2 = 1
;
CLRPULSE EQU 6      ;C3 = 0 CLEAR PULSE F/F
REMOVECLR EQU 7     ;C3 = 1 REMOVE CLEAR SIGNAL
;
ABORT EQU 8         ;C4 - ABORT LINE TO CY500
NOABORT EQU 9       ;C4 = 1
;
TRIGGER EQU OAH     ;C5 - CY500 TRIGGER INPUT
STOP EQU OBH        ;C5 = 1 - HALTS STEPPING
;
WAIT EQU OCH        ;C6 - WAIT INPUT TO CY500
HIWAIT EQU ODH      ;C6 = 1
;
RESETLO EQU OEH     ;C7 - RESET CY500 WHEN LOW
NORESET EQU OFH     ;C7 = 1 - DO NOT RESET
;
```



BINARY DATA PROGRAMMING EXAMPLE

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

```
TESTBINARY:
LXI  D,BINBUFFER
CALL SENDPARALLEL

RDYERROR?:
IN   STATUS
ANI  READY
JNZ  ERROR ;false ready

TSTINTREQ1:
IN   STATUS
ANI  MOTION
JZ   RDYERROR?

MVI  A,GREEN
OUT  LEDS
JMP  TESTBINARY
```

```
BINBUFFER:
DB   'C',Ø       ;clear pin 34
DB   'R',1,2ØØ  ;set rate para=200
DB   'F',1,1     ;set factor=1
DB   'N',2,5,Ø  ;set 5 steps
DB   'B',Ø       ;set pin 34
DB   'G',Ø       ;GO command
DB   ØFFH      ;STOPPER
```

In the command mode, the RDY/BUSY output remains low after a GO command is received until the CY500 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 CY500.

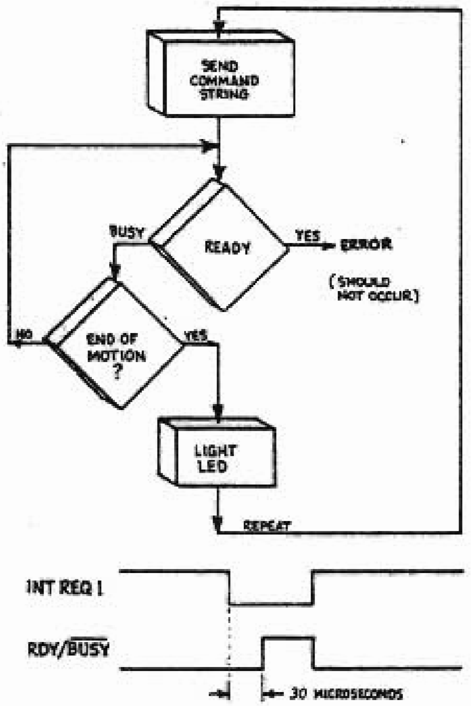
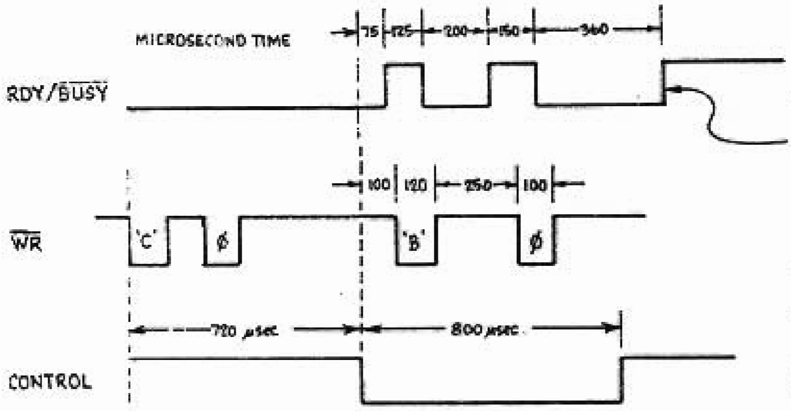


Figure 10.5 MOTION COMPLETE Timing



Trigger scope display using C output

Send C,Ø,B,Ø in BINARY command mode

Use WR to trigger scope display of CONTROL output(pin 31)

Figure 10.6 Binary Timing Example.

The parallel ASCII data is sent to the CY500 Stepper Motor Controller using the 8080 SENFPARALLEL code shown below. In this system the WRITE strobe is generated via a separate programmable control line and is removed after the data is acknowledged by the CY500.

```
                ;
                ;
                SENDPARALLEL      ; SEND PARALLEL ASCII
                ;                   BYTE
                ;                   ENTERED WITH DE-REG ->
                ;                   NEXT BYTE
                GETCHAR:
0059 1A         LDAX D
005A FEFF       CPI 0FFH          ; CHECK FOR STOPPER
005C CA6700     JZ QUIT
005F 13         INX D
0060 4F         MOV C,A
0061 CD6800     CALL SENDCHAR
0064 C35900     JMP GETCHAR
                ;
                QUIT
0067 C9         RET
                ;
                SENDCHAR:
0068 DBED       IN STATUS
006A E620       ANI READY
006C CA6800     JZ SENDCHAR
                ;
006F 79         MOV A,C
                ;---ANI 7FH       ; STRIP BIT 7
0070 D3EC       OUT DATA          ; SEND TO CY 500
                ;
0072 3E00       MVI A,WRBAR
0074 D3EF       OUT CONTROL       ; SEND WRITE STROBE
                ;
                WAITACKNOWLEDGE:  ; WAIT FOR RDY TO GO
                ;                   LOW (BUSY)
0076 DBED       IN STATUS
0078 E620       ANI READY
007A C27600     JNZ WAITACKNOWLEDGE
                ;
007D 3E01       MVI A,WRITEHI     ; REMOVE WRITE STROBE
007F D3EF       OUT CONTROL
                ;
0081 3EFF       MVI A,0FFH
0083 D3EC       OUT DATA          ; REMOVE DATA FROM BUS
0085 C9         RET
                ;
```

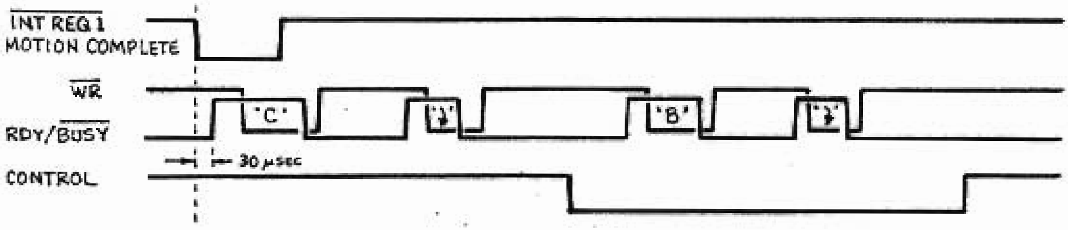
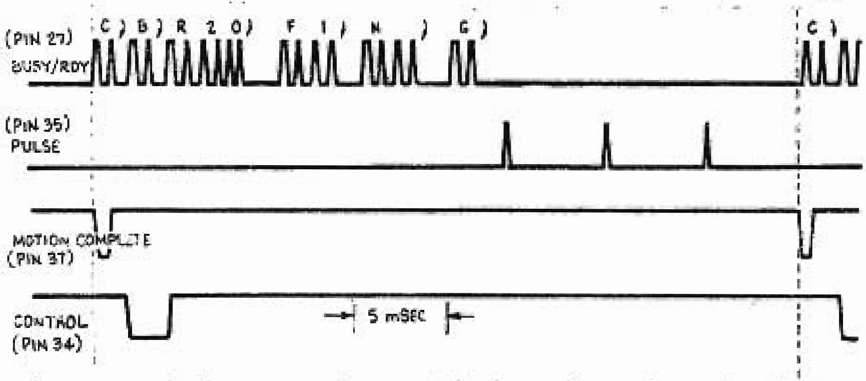


Figure 10.7 Expanded Handshake Timing Diagram.



NOTE LONGER BUSY PERIOD AFTER } ESPECIALLY WHEN DECIMAL TO BINARY CONVERSION REQUIRED.

USE MOTION COMPLETE TO TRIGGER OSCILLOSCOPE DISPLAY

Figure 10.8 Complete Timing for Sample Program.

```
                        ;
                        TESTASCII:      ; PARALLEL INPUT ASCII CODE
0029   114400           LXI D,ASCIIBUFFER
002C   CD5900           CALL SENDPARALLEL;SEND PARALLEL ASCII
                        ;
                        RDY?LOOP:
002F   DBED             IN STATUS
0031   E620             ANI READY
0033   C28600           JNZ ERROR
                        ;
0036   DBED             IN STATUS
0038   E601             ANI MOTION
003A   CA2F00           JZ RDY?LOOP
                        ;
003D   3EFE             MVI A,0FEH      ; LITE GREEN LED
003F   D3F5             OUT LEDS
                        ;
0041   C32900           JMP TESTASCII   ; LOOP FOR SCOPE DISPLAY
                        ;
                        ;      - - COMMAND STRINGS FOR CY500 - -
                        ;
                        ASCIIBUFFER
0044   430D             DB 'C',0DH        ; CLEAR OUTPUT LINE
0046   420D             DB 'B',0DH        ; RAISE CONTROL OUTPUT
0048   52203232 DB 'R 220',0DH            ; SET RATE PARAMETER=220
004C   300D
004E   4620310D DB 'F 1',0DH
0052   4E20330D DB 'N 3',0DH
0056   470D             DB 'G',0DH        ; BEGIN STEPPING
0058   FF               DB 0FFH           ; STOPPER FOR 8080 LOAD
                        ;                 ; ROUTINE
```

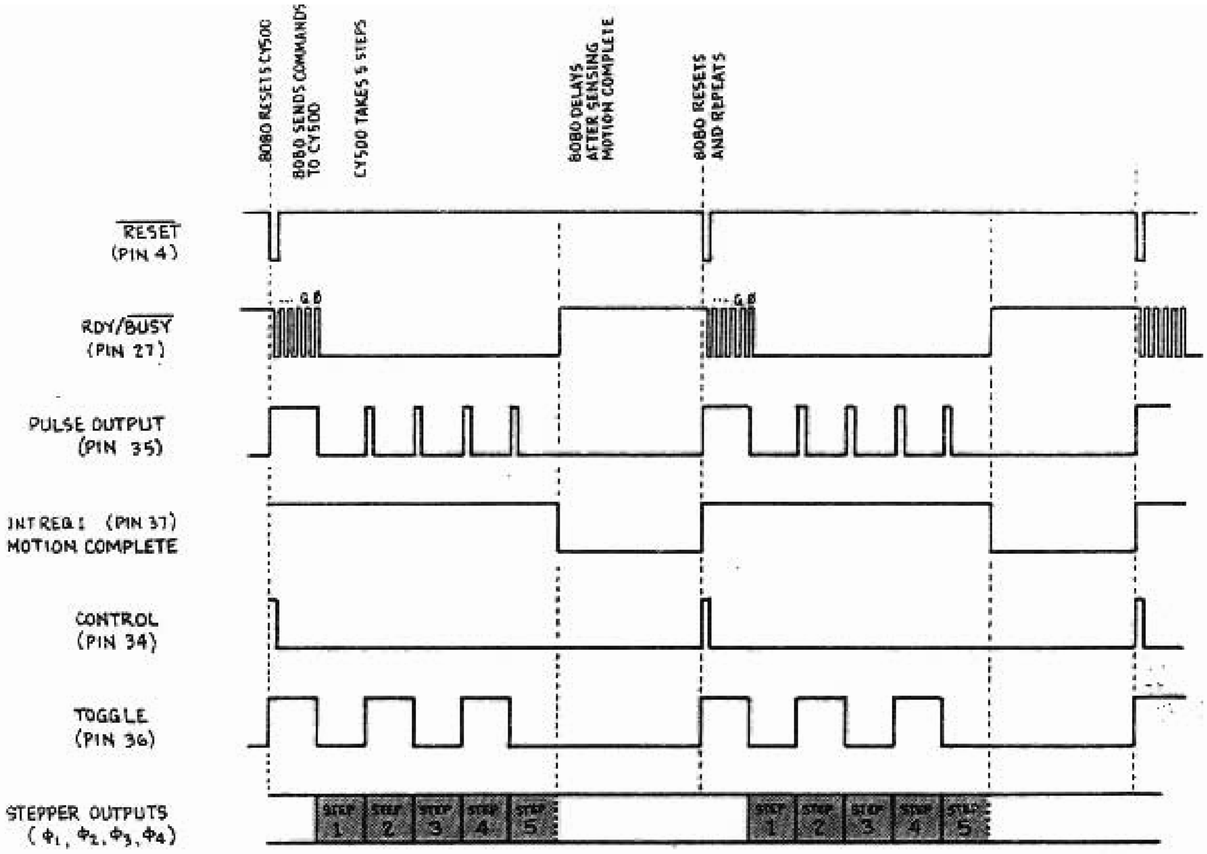


Figure 10.9 Timing for Program Shown Below.

The 8080 (or equivalent) sends the following commands and binary data to the CY500:

```
...       reset CY500 using pin 4
C Ø       clear control output (34) (trigger scope display)
R 1 FEH   set rate parameter = 0FEH
F 1 2     set rate factor = 2
N 2 5 Ø   set number of steps = 5
+ Ø       set CW direction (redundant)
G Ø       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 CY500 and repeats this process. The control output may be used to trigger the horizontal sweep circuits of a scope.

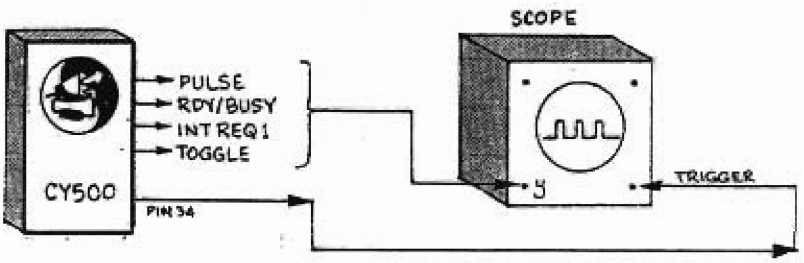


Figure 10.10 Test Setup.

IEEE-488 INTERFACE TO CY500

Using only a few SSI TTL gates, the CY500 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 CY500 signal names with the appropriate GPIB signals.

GPIB HANDSHAKE SIGNALS

The TALKER or device desiring to send 8 bits of data to the CY500 over the data bus uses the DAV (Data AVailable) signal that corresponds to the WR line on the CY500. Before lowering the DAV line the TALKER must test the NRFD (Not Ready For Data) line. This line corresponds to the CY500 RDY/BUSY line. When this line is low, the LISTENER (CY500) is Not Ready For Data. When the TALKER finds the NRFD line high then it can assert (lower) its DAV write line to the CY500. Thus far, the interface is identical to the standard CY500 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 CY500. The NDAC line is tested by the TALKER to determine whether or not the LISTENER has accepted the data. The CY500 RDY/BUSY 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.

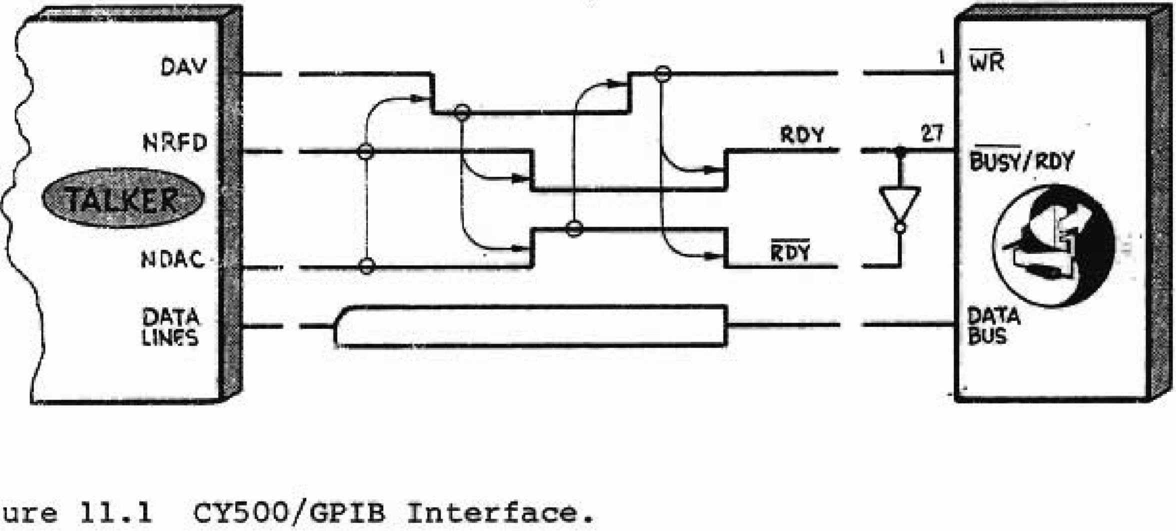


Figure 11.1 CY500/GPIB Interface.

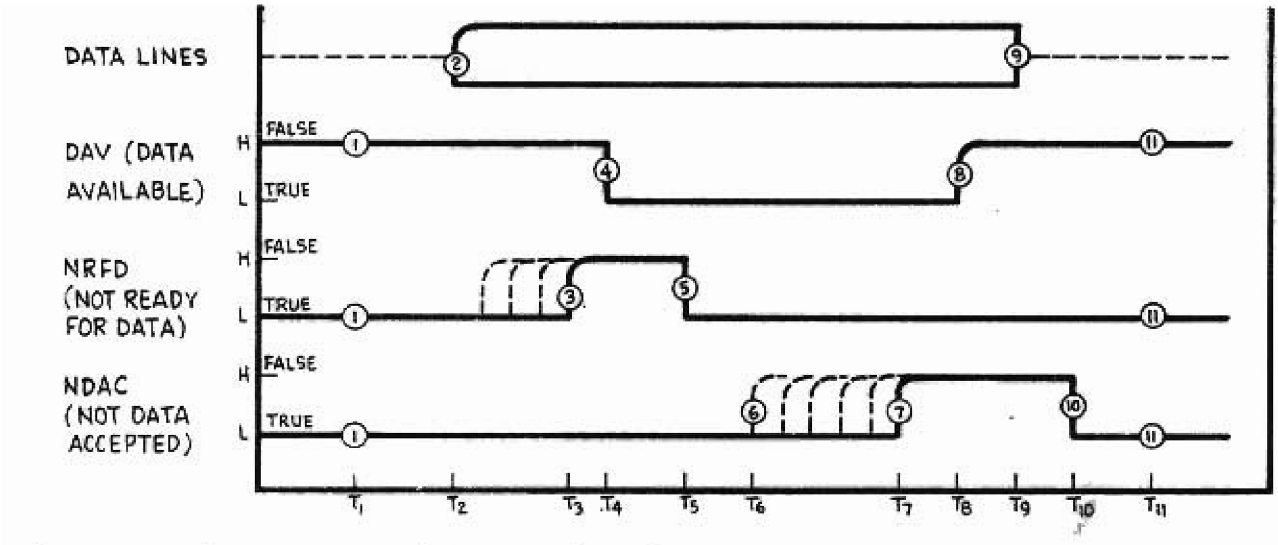


Figure 11.2 GPIB Handshake Signals.

The flowchart for the TALKER that controls the CY500 is shown in figure 11.3. This procedure can be implemented simply using any microprocessor and describes the manner in which most GPIB interface devices function.

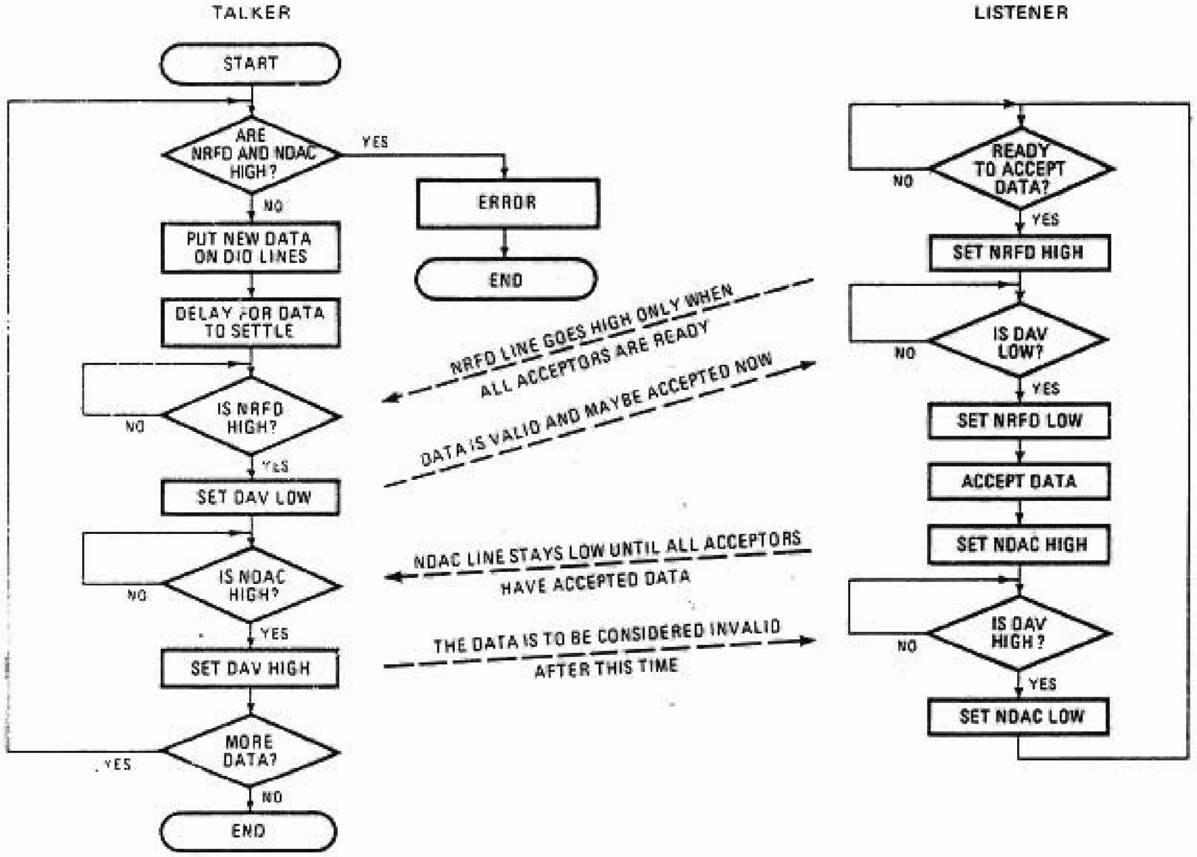


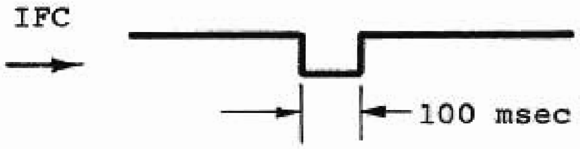
Figure 11.3 TALKER/LISTENER handshaking procedure.

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 CY500 signal lines.

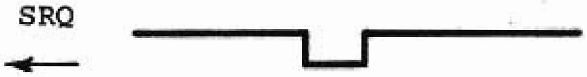
IEEE-488

CY500

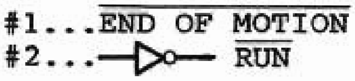


RESET (pin 4)

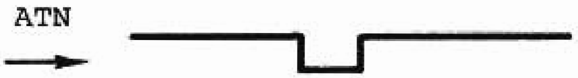
Interface Clear goes low after power on. This line is used to reset the CY500 and can replace the power on startup circuitry.

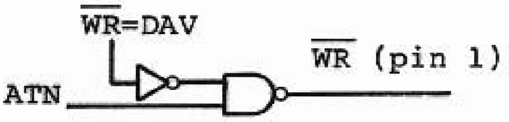


INTREQ1 (pin 37)

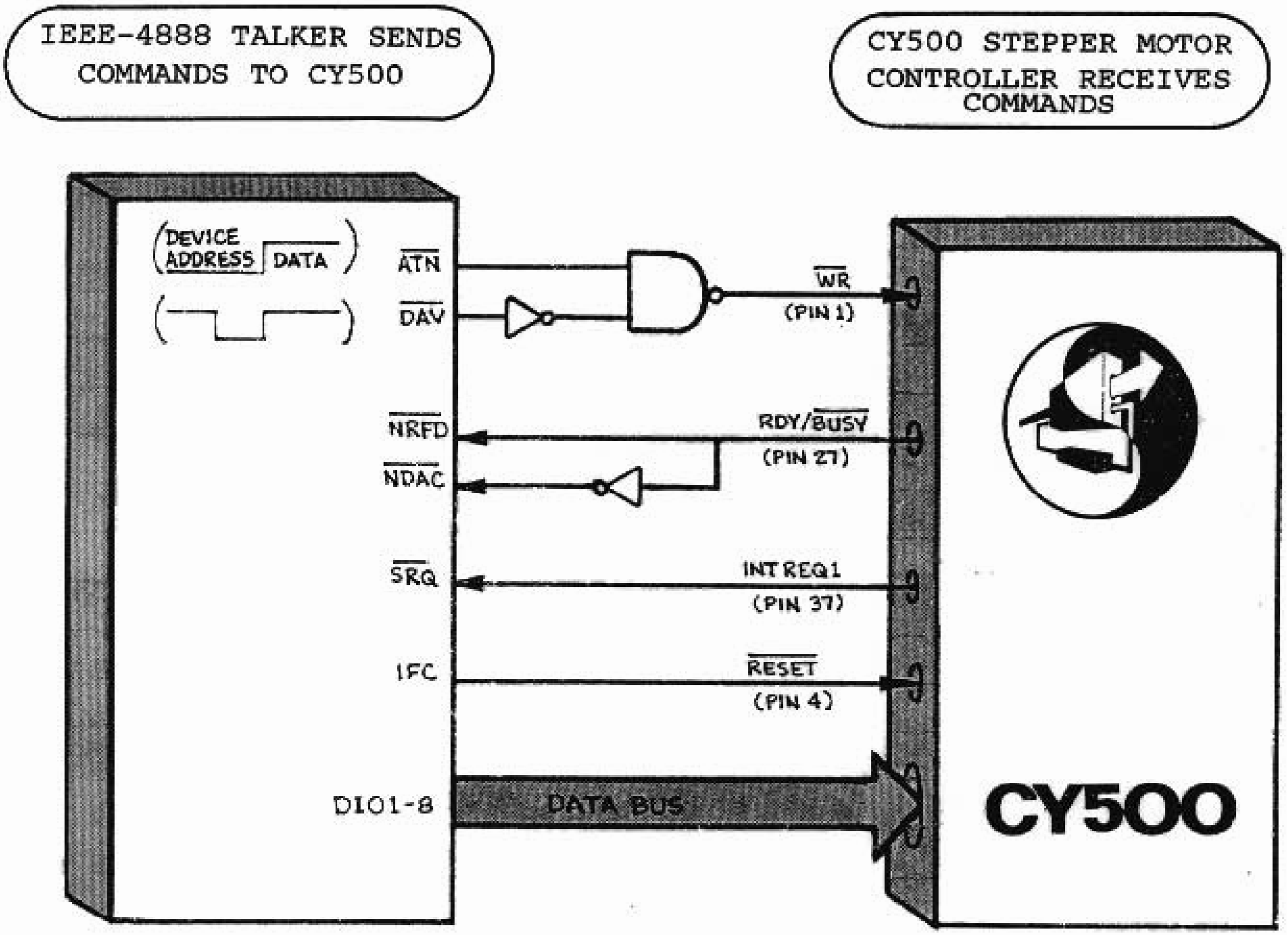


Service Request is used to inform the TALKER that the LISTENER (CY500) has completed an action and is ready for more commands.





The Attention line is used to signify that the data on the bus is a device address. For multiple CY500s this may be used for selection. The ATN line should inhibit the CY500 WR line. Note that ATN may also be used to prevent the CY500 from seeing line feeds (0AH) sent after linends (0DH) as is done by many BASIC language controllers.


IEEE-4888 TALKER SENDS COMMANDS TO CY500

CY500 STEPPER MOTOR CONTROLLER RECEIVES COMMANDS

Figure 11.4 Simple IEEE-488/CY500 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

12 GETTING YOUR CY500 RUNNING 12

Getting Your CY500 Running

The following checklist will simplify getting your CY500 up and running.

1. Be sure that pin 7 is grounded and pin 26 tied to +5 volts.

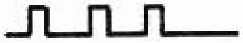
2. Be sure that pin 39 is high.

3. Set pin 33 high to select ASCII input, set TRIGGER (pin 30) low for now.

4. Be sure RESET (pin 4) is low for at least 10 milliseconds after power stabilizes. The CY500 can be reset at any time.

5. Upon proper reset all output pins should be at logic 1 (>3 volts).

6. Observe the RDY line (pin 27) to be sure it is high.

7. Observe the 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 WR line (pin 1).

10. Wait for the RDY (pin 27) to go low before bringing WR high. If using automatic WR strobe circuitry that generates low write signal when ASCII character is placed on the bus (as in figure 9.1 in manual) be sure that your software detects low RDY line before looking for High RDY. If you are using a debounced keyboard this should not be a problem.

11. When WR 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 WR strobe until RDY goes low, then return WR high as before.

14. Upon completion of the above sequences of steps, the CONTROL output (pin 34) will go low.

15. Repeat steps 8 through 14 replacing **C** (=43H) with **B** (=42H). This BITSET command will cause the CONTROL output (pin 34) to return high. All other outputs (except RDY) should have remained high during the above procedure.

16. Repeat steps 8 through 14 replacing **C** (=43H) with **O** (=4FH). This is the ONESTEP command. The result of this command will be to bring the Pulse Line (pin 35) low (and leave it low) and to apply the stepper control signals (on pins 21-24). The toggle line (pin 36) remains high.

17. If succeeding ONESTEP commands are given, the pulse line will pulse high at the beginning of each step (—⊓———) and the toggle line will change state with each step. The stepper control lines will step your motor in the clockwise (CW) direction.

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. raise the TRIGGER (pin 30) and enter the ONESTEP command (O↵). Nothing will happen on pulse, toggle, or the stepper control lines until the TRIGGER line is lowered.

    c. refer to figures 9.7, 9.8, and 9.9. 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 33 low to select binary. Read the manual carefully for differences in the two modes.

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		

CY512

INTELLIGENT POSITIONING STEPPER MOTOR CONTROLLER

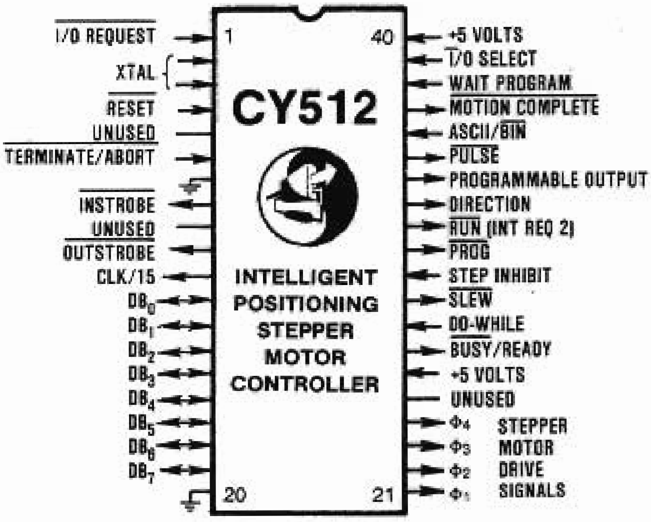
WITH ASCII/BINARY POSITION READOUT & AUTOMATIC DIRECTION FINDING

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.

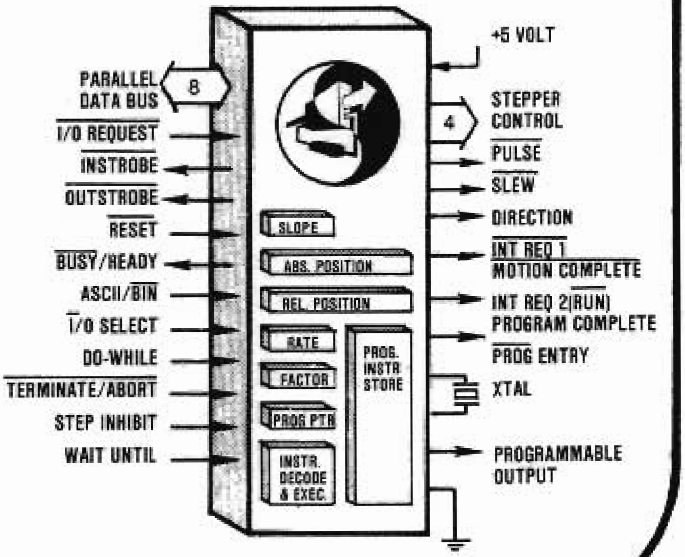
STANDARD FEATURES

- 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

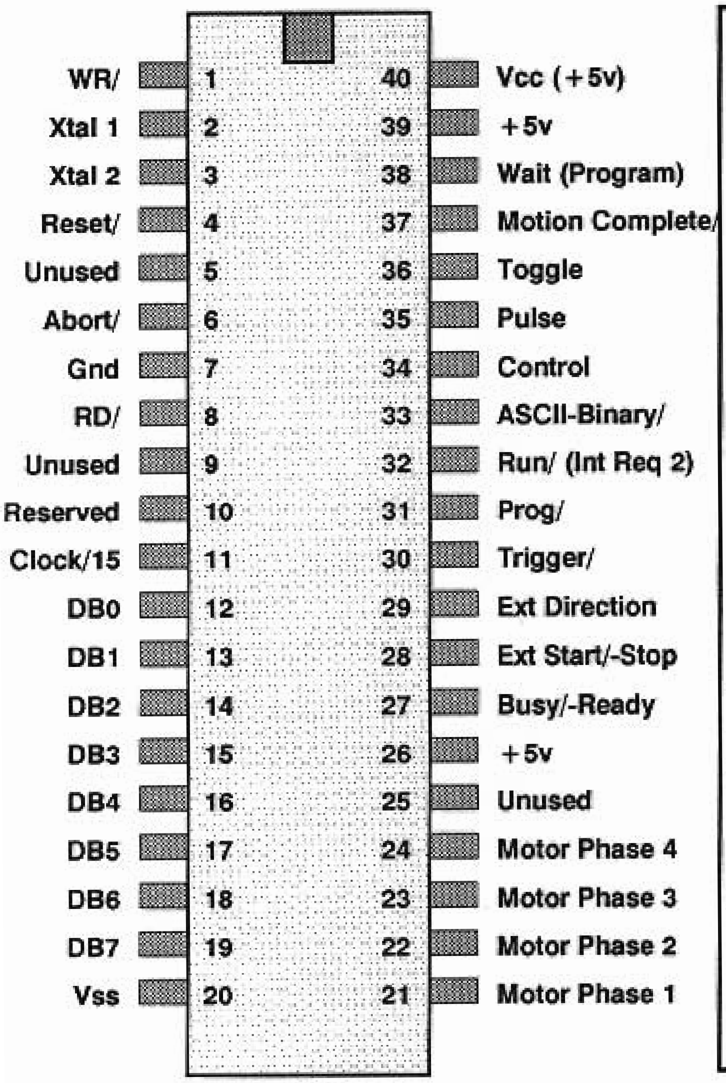


LOGIC DIAGRAM



CY500 Summary

CY500 Pins



WR/	1	40	Vcc (+5v)
Xtal 1	2	39	+5v
Xtal 2	3	38	Wait (Program)
Reset/	4	37	Motion Complete/
Unused	5	36	Toggle
Abort/	6	35	Pulse
Gnd	7	34	Control
RD/	8	33	ASCII-Binary/
Unused	9	32	Run/ (Int Req 2)
Reserved	10	31	Prog/
Clock/15	11	30	Trigger/
DB0	12	29	Ext Direction
DB1	13	28	Ext Start/-Stop
DB2	14	27	Busy/-Ready
DB3	15	26	+5v
DB4	16	25	Unused
DB5	17	24	Motor Phase 4
DB6	18	23	Motor Phase 3
DB7	19	22	Motor Phase 2
Vss	20	21	Motor Phase 1

CY500 Commands

A	Athome zero position
B	Bitset Control line high
C	Clearbit Control line low
D	Doitnow, execute stored prog
E	Enter program code
F f	Factor divides rate value
G	Go, step relative
H	Halfstep mode
I	Initialize
J	Jog using ext start/stop line
L	Left-right step from ext pins
N n	Number of Steps
O	Onestep at a time
P p	Position for stepping
Q	Quit entering program code
R r	Rate, maximum step rate
S s	Slope of accel/decel
T	Til pin 28 high, loop thru prog
U	Until pin 38 low, wait here
W	Wait for pin 38 to go high
X x	eXpend milliseconds
+	CW direction
-	CCW direction
0	Resume Command mode