SGS-THOMSON MICROELECTRONICS

GS-C200
GS-C200S

INTELLIGENT STEPPER MOTOR CONTROLLERS

FEATURES

- Absolute and incremental positioning
- Up to 999,999 step per move
- Speed range to 10,000 steps/s
- Ramp lenght to 999 steps
- Single unregulated supply voltage
- Index and velocity mode
- Automatic and Home positioning
- Loops and Delay execution
- Conditional start and stop
- Status feedback to the host
- RS232 communication port
- Point to point and Multipoint protocol
- Closed loop operation
- Counter preset (GS-C200S only)
- Jump to (GS-C200S only)
- Jump to on-condition (GS-C200S only)
- Initialization during execution (GS-C200S only)
- Auxiliary output voltages +5V, ± 12V

DESCRIPTION

The GS-C200 and GS-C200S are powerful stepper motor control modules that interface every power sequencer/driver available on the market.

A sophisticated hardware and an easy to learn programming language result in minimal development and debugging time of motion control systems. The modules are supported by dedicated software that includes both an on-screen editor and a debugger that greatly improve the module ease of use.

The instruction sets comprise respectively 25 (GS-C200) and 29 (GS-C200S) different commands which can be executed either under host control or in a stand alone environment. An on board EEPROM is used for program saving and retrieving.

The availability of three User inputs and three programmable User outputs, each of which can be tested or set under program control, assures to the designer a high level of system power and flexibility.

ABSOLUTE MAXIMUM RATINGS

Symbol	Parameter	Value	Unit
V_S	DC Supply Voltage	42	V
T_{stg}	Storage Temperature Range	– 40 to + 85	°C
T_{op}	Operating Temperature Range	0 to + 50	°C
	Humidity (non condensing)	0 to 90	%

ELECTRICAL CHARACTERISTICS (T_A = 25C and V_s=24V unless otherwise specified)

Symbol	Parameter		Min	Typ	Max	Unit
V_s	DC Supply Voltage		12		40	V
I_s	Quiescent Supply Current			80		mA
V_i	Logic Input Voltage (TTL compatible)	Low High	 2		0.8 5	V V
V_o	Logic Output Voltage (TTL compatible)	Low High	 2		0.8 5	V V
t_{cpw}	Clock Pulse Width				5	µs
t_{rpw}	Reset Pulse Width (Internal)				500	µs

MOTION CHARACTERISTICS

SPEED RANGE	10 to 10000 steps
SPEED RESOLUTION	10 steps
RAMP LENGHT	1 to 999 steps
RAMP RESOLUTION	1 step
POSITIONING RANGE (C200) (C200S)	0 to 9999999 – 8388608 to + 8388607
SINGLE MOVEMENT RANGE	1 to 999999 steps
POSITIONING RESOLUTION	1 step
POSITIONING REPEATIBILITY	+/– 0 step
PROGRAM STORAGE CAPABILITY	119 bytes

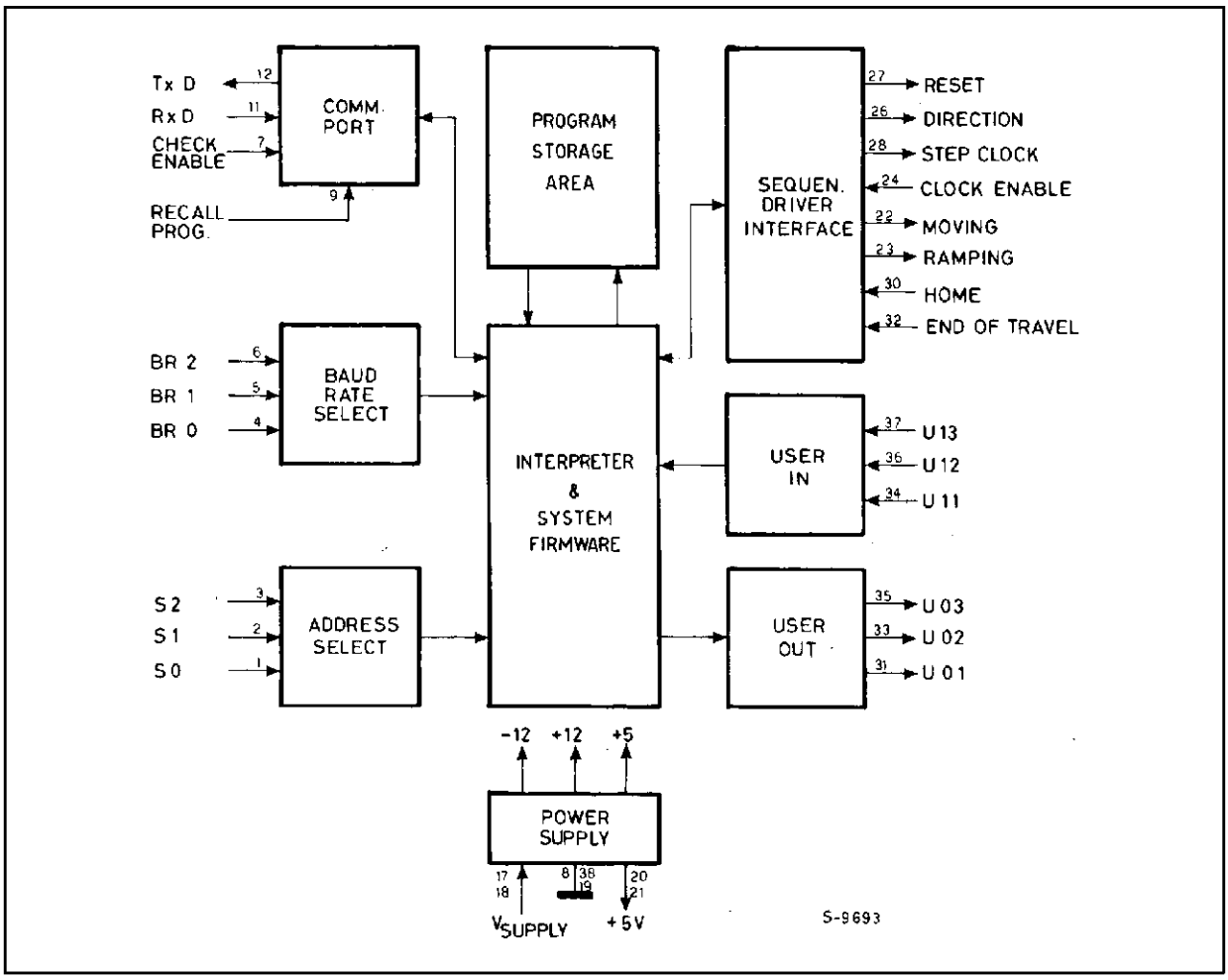
COMMUNICATION PORT CHARACTERISTICS

SIGNAL LINES	3 (TxD, RxD, GND)
BAUD RATE RANGE	110 to 9600
FORMAT	1 Start Bit 7 Data Bit 2 Stop Bit Odd parity

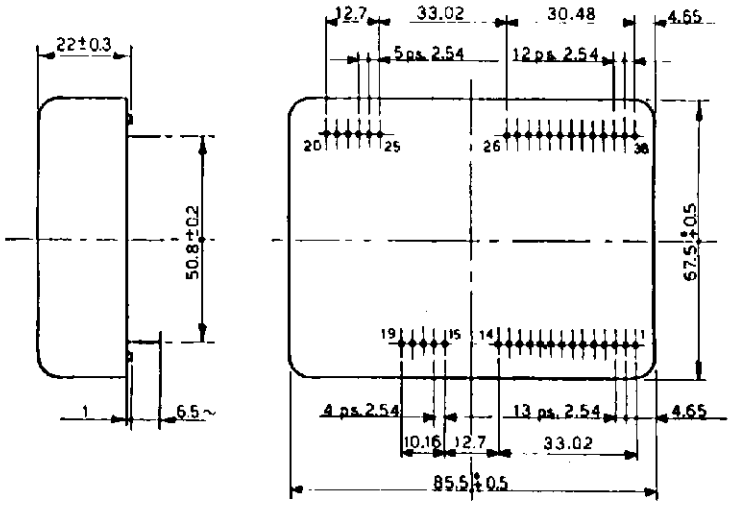
STORAGE CAPACITY

MINIMUM NUMBER OF COMMANDS	30
MAXIMUM NUMBER OF COMMANDS	45

Figure 1. Block Diagram



CONNECTION DIAGRAM AND MECHANICAL DATA



Dimensions in mm.

Bottom view

PIN DESCRIPTION

Pin	Function	Description
1	SEL0	Protocol/address LSB select input
2	SEL1	Protocol/address SSB select input
3	SEL2	Protocol/address MSB select input
4	BR0	Baud rate LSB select input
5	BR1	Baud rate SSB select input
6	BR2	Baud rate MSB select input
7	CHS	Checksum enable input
8	GND	Ground
9	REC	Program autorecall input
10		Must be connected to pin 8
11	RXD	RS232 received data input
12	TXD	RS232 transmitted data output
13	TXPD	Transmitted data pull-down resistor
14	RDY	Status logic output
15	–VSL	Unregulated –12V supply output (note 1)
16	+VSL	Unregulated +12V supply output (note 1)
17	Vs	Supply voltage input
18	Vs	Supply voltage input
19	GND	Ground
20	5V	5V Auxiliary output (note 2)
21	5V	5V Auxiliary output (note 2)
22	MOV	Motor moving logic output
23	RAMP	Motor ramping logic output
24	ENABLE	Stop enable logic input
25		Not connected
26	DIR	Direction selection logic output
27	RESET	Power driver Reset logic output
28	CLOCK	Step clock logic output
29		Not connected
30	HOME	Home position logic input
31	UO1	User 1 logic output
32	EOT	End of travel switch logic input
33	UO2	User 2 logic output
34	UI1	User 1 logic input
35	UO3	User 3 logic output
36	UI2	User 2 logic input
37	UI3	User 3 logic input
38	GND	Ground

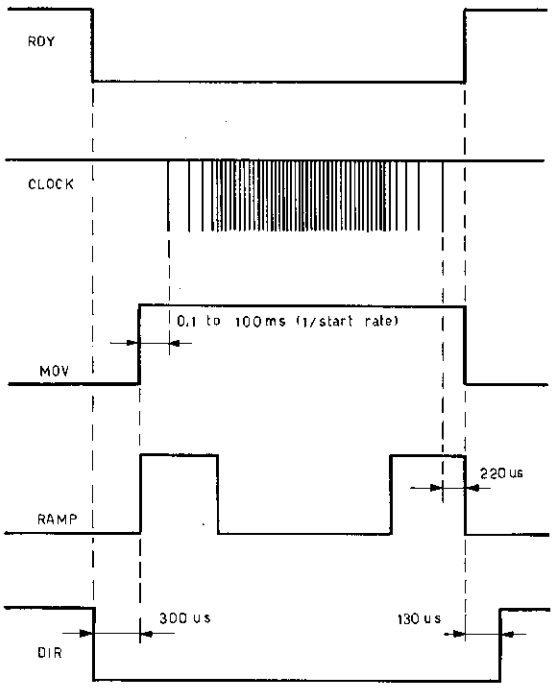
Notes: 1 – Maximum available current is 10mA
2 – Maximum available current is 100mA

The various signals that characterize the GS-C, their function and the active level are described in detail in the following:

Pin	Function
1 - 2 - 3	The SEL0 (pin1), SEL1 (pin2) and SEL2 (pin3) inputs are used to select the communication protocol and the module address. They have an internal pull-up and when unconnected they are at the 1 logic level.
4 - 5 - 6	The BR0 (pin4), BR1 (pin5) and BR2 (pin6) inputs are used to select the Baud rate of the communication port. They have an internal pull-up and when unconnected they are at the 1 logic level.
7	The CHS checksum generation conditioning input enables the user to include or exclude the checksum character from the data exchange string. A "zero" logic level applied to this input disables the control and the generation of the checksum character thus allowing the GS-C to be connected to a video terminal.
8	This pin is the common terminal for all logic signals and for the power supply return path.
9	The REC Recall Program Enable input pin, when brought to "zero", enables the automatic recall of the program stored in the EEPROM and its immediate execution.
10	This pin is for testing purpose only and it must be grounded for normal operation.
11	The RxD input of the serial communication port is used by the module to receive commands from the Host Computer. The input logic levels are compatible with the RS232 and V24 standards.
12	The TxD output of the serial communication port is used by the module to send data to the Host Computer. The logic levels of this output are compatible with the RS232 and V24 standards.
13	The TxPD Transmitted data pull-down resistor pin must always be connected to the TxD output (pin 12) when the Point-to-Point protocol is used. When the Multipoint protocol is selected, this pin must be left open on all modules except the chain terminator unit, in order to avoid the TxD output overload.
14	The RDY hardware status output (open collector) signal pin is used as the controller status flag. RDY assumes a "zero" logic level when a command or a program is in execution
15	–12V unregulated output. A maximum of 10mA can be sinked from this pin.
16	+12V unregulated output. A maximum of 10mA can be sinked from this pin.
17 - 18	Module supply input. For correct operations a supply voltage ranging from 12 to 40 Volt is required.
19	See pin 8.
20 - 21	5 Volt regulated output, available either for the Sequencer-Driver logic section or for a custom interface logic supply. The maximum current that can be sinked from this pin is 100mA.
22	The MOV Motor moving output becomes the logic level "one" when the GS-C is executing a movement. This output can be used to program the phase current level when the motion is running at a level higher than for the rest condition.
23	The RAMP Ramp in execution output is rised to the logic level "one" when the GS-C is executing an acceleration or a deceleration ramp. This output can be used to program the phase current level when the motion is ramping at a level higher than for the rest or slewing condition.
24	The ENABLE input pin allows the user to control the Step clock logic output to avoid the motor being stepped if the previous step was not correctly executed. A "zero" logic level applied to this pin stops the generation of the step pulses. This input can be used to stop the system when an emergency occurs, to execute the motion according to externally generated timing, or to implement a closed loop control system.
25, 29	Not connected.
26	The DIR Direction selection output is used to inform the Sequencer-Driver on the direction of rotation. The logic level "one" determines a clockwise rotation, but of course the rotation depends on the motor phases connection to the Sequencer-Driver.
27	The RESET Power driver Reset output is brought to the "zero" logic state for 400µs when the unit is powered-on, or when the GS-C receives the "Initialize position counter" command. This output is normally used to assure the correct start-up of the Sequencer-Driver or any other external custom logic.

Pin	Function
28	The CLOCK Step clock output is used to inform the Sequencer-Driver to perform a step. The direction (clockwise or counterdockwise) is defined by the logic status of the DIR output. In steady conditions, the CLOCK is at the "one" logic level, and the step is represented by a negative going pulse with a 1.7µs duration.
30	The HOME Home position input allows the system to find its reference point. This input can be driven by a mechanically activated contact indicating the "zero" position. It is normally used together with the EOT End-of-travel signal.
31	The UO1 User output 1 is intended for user purposes. The status of this output can be set and cleared under program control and it can be used for various functions. It is normally used for the control of external devices, the selection of the Sequencer-Driver operating mode, or the synchronization of complex movements.
32	The EOT End-of-travel input allows, in combination with the HOME input, the correct mechanical initialization of the system. For this purpose it must be brought to the "zero" logic level when the system reaches the run end position.
33	The U02 User output 2 is intended for user purposes. See pin 31 description.
34	The UI1 User input is intended for user purposes. The status of this input can be read by the Host Computer or tested during the program execution, and used to condition the start of a movement, the execution of a specific portion of a program (GS-C200S only), or any other similar operation.
35	The UO3 User output 3 is intended for user purposes. See pin 31 description.
36	The UI2 User input 2 input is intended for user purposes. See pin 34 description.
37	The UI3 User input 3 input is intended for user purposes. See pin 3 and pin 4 description.
38	See pin 8.

Figure 2. GS-C Timing Diagram



S.I.M.P.L.E. Interpreter Command and Functions
(**S**GS-**T**HOMSON **I**nteractive **S**tepper **M**otor **P**rogramming **L**anguage and **E**xecutor)

Command	Byte Length	Function
Ax	2	Activate the specified (x) User output.
Cx	2	Clear the specified (x) User output.
Dxxx	2	Delay for the specfied number (xxx) of tenth of second.
E	–	Start executing the program currently stored into RAM memory.
F	–	Feedback the GS-C status (i.e. Ready or Busy).
f+/–xxxxxxx	4	Preset the position counter to the specified absolute value (C200S).
G+/–xxxxxxx	4	Go to the specified target position (C200S).
g(+/–)	4	Move the motor indefinitely in the specified direction.
g(+/–)x	4	Move the motor in the specified direction until the specified (x) input is brought to zero.
H(+/–)	–	Find Home position moving clockwise (+), or moving counterclockwise (–).
Ix	2	Initialize the position counter (x=1), the user outputs (x=2), or both (x=3).
jx	2	Jump to memory location (x). Location (x) ranges between 0 and 118 (C200S).
jcy, x	2	Jump to memory location (x) if the binary value of the user inputs matches (y) value (C200S).
K	–	Kill the program in execution.
Lx	2	Loop for the specified (x) number of times.
M	–	Transfer the RAM memory content to EEPROM.
P	–	Enter the programming mode (C200).
Po	–	Enter the programming mode (C200S).
Px	–	Exit the programming mode (C200S).
Q	–	List to the host the program currently in RAM memory.
Rxxx	4	Set the Ramp length to the specified (xxx) value.
Sxxx	4	Set the start-stop speed to the specified (xxx) value.
Txxx	4	Set the slew rate speed to the specified (xxx) value.
Ux	2	Execute the program until the specified (x) user input is brought to a low level.
Vx	–	Read back the current position (x=1) or the user I/O status (x=2).
X	–	Transfer the program from EEPROM to RAM.
Wx	2	Wait until the specified (x) user input is raised to a logic one level.
Z	–	Stop through a deceleration ramp.
+/–xxxxxx	4	Move clockwise (+) or counter-clockwise (–) for the specified (xxxxxx) number of steps.

GS-C200 AND GS-C200S DESCRIPTION

The increasing popularity of microprocessors and their very low cost, have contributed to a fast growth of stepper motors usage in a large numbers of application previously covered by more complex, bulk and expensive DC motors servo loops. The GS-C200 and the GS-C200S modules have been conceived to help the industrial designer in designing the stepper motor applications based on microprocessor control.

These modules are programmable intelligent stepper motor controllers that coordinate highly complex movements and sequential operations. This capability is performed through the integration of sophisticated hardware and an easy to learn and very functional and powerful programming language.

Thanks to this high level programming language, the power of the instruction set and the ability to condition and control the program execution through the USER inputs and outputs, the GS-C200 and GS-C200S drastically reduce the design time and start-up manufacturing phase of very complex systems. The GS-C200S offers an advanced and powerful instruction set that includes also the conditional jump which allows for more efficient program-ming. The GS-C200, the GS-C200S and their companion modules, the GS-D200 and the GS-D200S, can be used to drive in chopped mode of bipolar stepper motor with a 2/2.5A maximum phase current rating.

The two modules (GS-C and GS-D) are available also on a single Eurocard board named respectively GS-DC200, GS-DC200S and GS-DC200SS according to the various modules combination (see the relevant data sheet). In the following the modules will be generically named GS-C. The specific module part number will be used when the feature is unique to that module.

A MOTION SYSTEM ARCHITECTURE

A complete motion system controlled by a host computer is normally configured as per fig. 3.

The GS-C logical and functional architecture is shown in fig. 1 and it includes the following basic blocks:

– Interface to the Host Computer via an RS232 communication port.

– Address and baud rate selection.

– Interface to the Sequencer-Driver (in particular but not exclusively, to the GS-D200 or GS-D200S) via 5 output and 3 input lines

– Command Interpreter and Executor.

– Program storage area

– Power Supply.

The above mentioned functions are performed by the GS-C without the addition of any external component, and the module flexibility is further enhanced by the use of only one unregulated supply voltage that can be the same used to supply the Sequencer-Driver (from 12V up to 40V).

Commands are sent to the module by a Host Computer or by a simple video terminal during the programming/debugging phase through an RS232 serial port. They are interpreted and validated by the command interpreter and executed through the Sequencer-Driver interface.

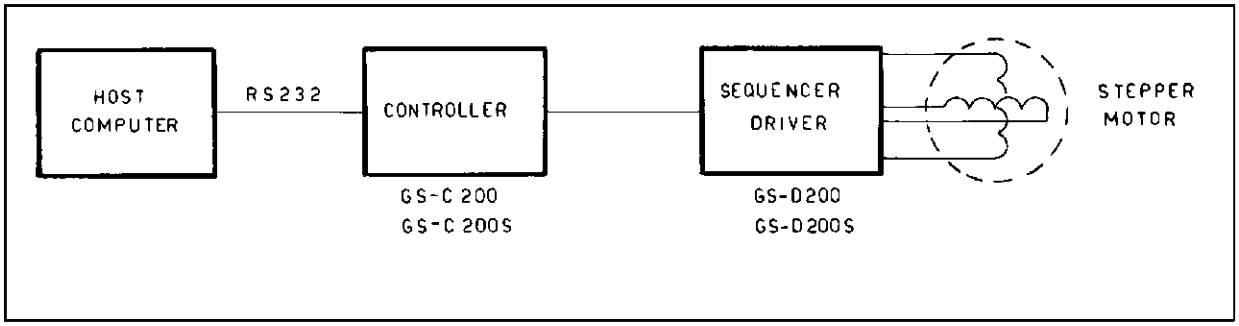
Command execution can be conditioned and controlled by the status of the USER IN-OUT interface.

A program storage area has been added to permanently store a program in an on-board EEPROM; this is particulary beneficial to obtain a low cost stand-alone controller that does not need any connection to an external computer or to store programs frequently used in complex motion sequencies thus reducing the host computer burden and speeding up the system processing.

Particular attention has been given to the simplicity of the instruction set to allow an easy design of the system to those designers that are not very familiar with microprocessor software and programming.

In the following a detailed description of the various functional blocks is given.

Figure 3. A Motion System Block Diagram



INTERFACE TO THE HOST COMPUTER AND DATA PROTOCOL

The interface to the Host Computer is through an RS232 or V24 serial communication port.

Baud Rate Programming

The Baud rate is programmed between 110 and 9600 bit/sec by using the BR0, BR1 and BR2 inputs according to the following table:

BR0 (p4)	BR1 (p5)	BR2 (p6)	Baud Rate
0	0	0	9600
1	0	0	4800
0	1	0	2400
1	1	0	1200
0	0	1	600
1	0	1	300
0	1	1	150
1	1	1	110

This setting is obtained by connecting the pins 4, 5, and 6 to ground (0 status) or by leaving them open (1 status). The communication port does not use any control line but just the transmit and receive signals. The host computer must handle the data excange in the proper way.

Module Address Programming

The communication protocol can be either Point to Point or Multipoint. In the first case a single communication line is required for each module, while in the latter more than one module (up to seven) can share the same communication line.

The Multipoint protocol as well as the peripheral device address are selected through SEL0, SEL1 and SEL2 inputs. The Point-to-Point protocol is selected by connecting all the SEL inputs to the 5V output pin (pin 20) or by leaving them open.

The following table defines the protocol and the address setting:

SEL2	SEL1	SEL0	Address	Protocol
0	0	0	7	Multipoint
1	0	0	6	Multipoint
0	1	0	5	Multipoint
1	1	0	4	Multipoint
0	0	1	3	Multipoint
1	0	1	2	Multipoint
0	1	1	1	Multipoint
1	1	1	–	Point-to-Point

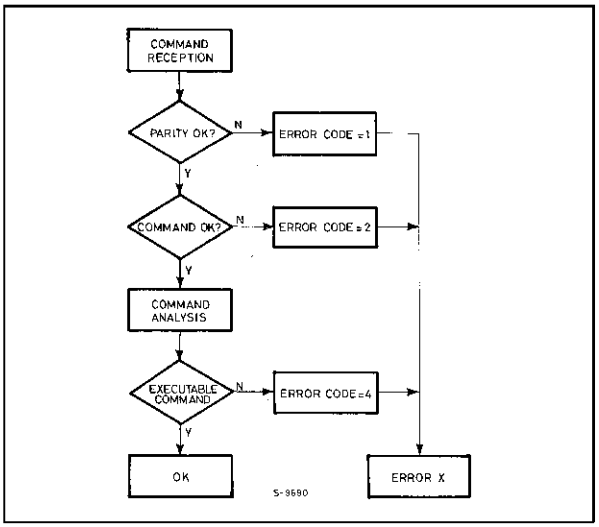
When the multipoint connection is chosen, the address of each module is obtained by connecting the various SEL pin (1, 2, 3) to ground (0 status) or by leaving them open (1 status).

The basic difference between the two protocols is represented by the sytem wiring complexity and the data throughput. The Point-to-Point offers the higher throughput data rate but it requires a connecting cable for each unit, while the Multipoint minimizes the connecting cables but at reduced throughput rate. When this latter protocol is chosen, the command must always be preceeded by the address of the unit.

Data Exchange Protocol

The dialogue is always driven by the Host Computer which sends the string containing the command or the request to be implemented. The GS-C module stores the instruction sent by the Host and then it checks if the string has been correctly received by analyzing the parity bit. It then analyzes the consistency of the received instructions by verifying the presence and correctness of the argument, and finally, it checks whether the request can be processed or not (for example, an attempt to move outside the system limits, etc.) reporting to the Host the analisys result. If no error is detected, the GS-C replies to the Host by a "Y" message. In case of error, the message will be "Error X" requesting the Host to send the message again or to modify some parameters of the previous message to fix the error detected by the GS-C. The actual value of X (see fig. 4 and error code table) gives the Host the information on the type of detected error. The procedure implemented for the dialogue with the Host is shown on the flowchart of fig. 4.

Figure 4. Controller-Host Dialogue Flowchart.



The general format of a command string is the following:

ADDRESS	COMMAND	ARGUMENT	CHECKSUM	CAR.RETURN

The **Address** must be the first transmitted character and it is present only if the Multipoint protocol is used (at least one of SEL0, SEL1, SEL2 is different from zero).

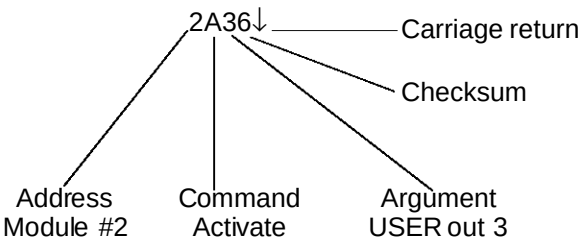
The **Command** is the second character(s) of the string, in the Multipoint protocol, but it becomes the transmission opening character when the Point-to-Point protocol is used (SEL0, SEL1 and SEL2 = 0).

The **Argument**, if required, is specified immediately after the command and its length depends on the command type.

The **Checksum** character verifies the correctness of the received string; its value is determined by the sum of the binary values of the preceding characters. The result is cut at the seventh least significant bit and ORed with exadecimal 10 (C200S/C200 from V2.2) to make the result compatible with the transmission system. The last character, the string ending character, is always a **Carriage Return** that will be identified in the following by the symbol (↓).

By connecting the pin CHS (pin 7) to ground, the checksum character is not anymore requested, and the task of guaranteeing the correctness of the message is left to the parity bit. It should be noted that by using this dialogue mode, the data integrity confidence level is reduced. Because motion systems normally operate in manufacturing premises subjected to heavy electro-magnetic noise, and because any communication problem may have catastrophic effects on the system actions, it is a good practice to use the checksum character whenever possible. The checksum character is normally not used (pin CHS connected to ground) when the GS-C is connected to a video-terminal, i.e. during the initial programming and debugging phase. In the following, three examples of command strings sent to a GS-C module are given.

Example 1 - MULTIPOINT PROTOCOL. The Host Computer wants to set the USER output 3 of the module #2. The command will have the following format:



The checksum character **6** results from the binary sum of the character **2** (ASCII value = 32) + character **A** (ASCII value = 41) + character **3** (ASCII value = 33) truncated at the seventh bit.

Example 2 - POINT-TO-POINT PROTOCOL.

The same instruction is given by the Host to a Point to Point connected module.

The command will have the following format:

A3t↓

The checksum character has an ASCII value **t** that derives from the sum of the ASCII code A+3 = 41+33 = 74 in binary weighted code or **t** in ASCII code.

Example 3 - POINT-TO-POINT PROTOCOL WITHOUT CHECKSUM.

For the same instruction, the command format will be:

A3↓

The string consists of command and argument only.

The GS-C feeds back information to the Host every time it receives a command, therefore it has not to identify itself to the Host when answering in a Multipoint connection.

The format of the string answered back by the GS-C is the following:

ANSW.CODE	ARGUMENT	CHECKSUM	CAR.RETURN

The first character, which always identifies the answer type, may assume one of the following values:

Y The command string has been correctly received.

B The controller is Busy and cannot process commands.

R The controller is Ready to process commands.

E An error has been detected. The type of error is specified by the number following the "E".

V A controller status (a position or an USER input/output status) is sent back and its value is specified by the characters following the "V".

The length of the Argument, present only for "E" and "V" answers, can range between 1 and 7 characters, and it is a function of the received command. The number following the "E" code, i.e. the error argument, specifies the detected error type according to the following table:

Error code	Type of error
1	Parity error when receiving one or more characters, checksum error, or too long a command string.
2	Command argument out of limit or not requested.
3	Storage capacity overflow.
4	Not allowed or not executable command.
5	Overflow error during program execution (GS-C200 only).
6	EEPROM programming error.

The number following the "V" code depends on the type of the received command.

When the GS-C answers to a "V1" request (feedback the actual absolute position against the Home position), the answer will be:

Vxxxxxxx↓

where the xxxxxxx represent the absolute position.

When the GS-C answers to a "V2" request (feedback the USER input/output status), the answer will be:

Vxy↓

where the x and y meaning is:

x = 1 User Input 1 = 1
x = 2 User Input 2 = 1
x = 4 User Input 3 = 1
y = 1 User Output1 = 1
y = 2 User Output 2 = 1
x = 4 User Output 3 = 1

The logic values of the inputs and outputs are added together. For example the answer:

V36↓

indicates the following USER I/O status:

UI1 =	1	UO1 =	0
UI2 =	1	UO2 =	1
UI3 =	0	UO1 =	1
	3		6

The presence of Checksum character, whose value is calculated by using the method described in the previous example, is conditioned by the CHS pin status.

When CHS is grounded (either by a logic signal or by a strap to ground) the checksum is deleted.

The string terminator is, as in the previous case, a Carriage Return.

THE SEQUENCER-DRIVER INTERFACE

The interface to the Sequencer-Driver and, through it, to the mechanical environment, consists of eight logic signals (5 outputs and 3 inputs) which enable the GS-C intelligent controller to interface the GS-D200 or the GS-D200S modules as well as any Sequencer Drivers currently available. The eight signals can be divided into two groups, named respectively:

PRIMARY SIGNALS

UTILITY SIGNALS

The primary signals are those necessary for the correct system operation:

RESET	Output to reset the Sequencer-Driver.
CLOCK	Step clock output.
DIR	Direction output.
ENABLE	Step enable input.

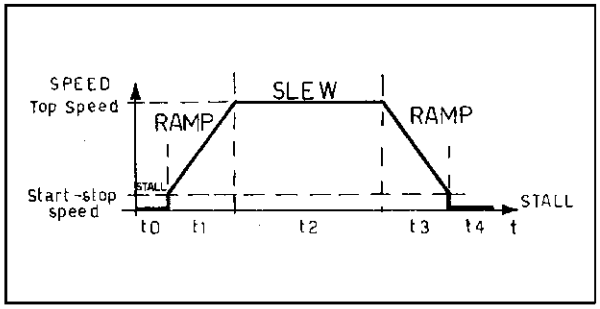
The function of each signal is described in detail in section **PIN DESCRIPTION** on page 4/31; it will be shown later that the Step Enable Input in conjunction with the position sensor of the motor, allows the implementation of closed loop systems (see paragraph **Closed Loop Operation** on pag. 27). The Utility signals allow the optimization of the driving system and the minimization of the hardware. They are:

MOV	Movement in execution output.
RAMP	Ramp in execution output.
EOT	Mechanical End of Travel input.
HOME	Electrical Home Position input.

By using these signals it is possible to correctly define the system starting point or reference position, or to change the current in the motor windings during the acceleration and deceleration phases in order to optimize the motor performance.

A typical example of the utility signals implementation is given here. Let's suppose that the required speed profile is as shown in fig. 5.

Figure 5. Speed-Time Profile.



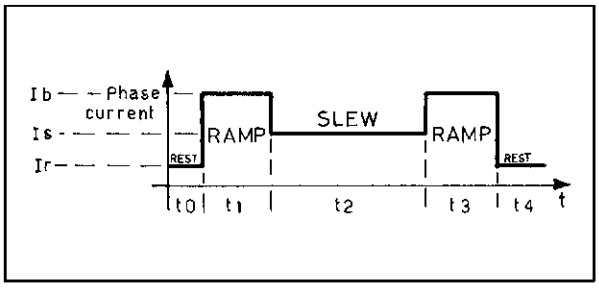
To optimize the motor torque during the acceleration and deceleration (t_1 and t_3) it is convenient to use a phase current profile as shown in fig. 6. During the SLEW phase (t_2) when the motor rotates at constant speed, the current is reduced to the minimum value necessary to compensate the system losses (friction) and the load inertia. During the STALL phase (t_0 and t_4) the current is further reduced to the bare value necessary to maintain the load in the right mechanical position. By using this current profile the power dissipation of the Sequencer-Driver and motor is optimized.

This profile can easily be implemented by using the utility signals:

MOV Movement in execution.

RAMP Ramp in execution.

Figure 6. Phase Current-Time Profile.



The status of these two outputs can be used to set the appropriate phase current value for the power driver, by a simple but effective interface circuit that is described in detail in fig. 11 of paragraph **PHASE CURRENT PROGRAMMING** on page 24.

THE USER INTERFACE

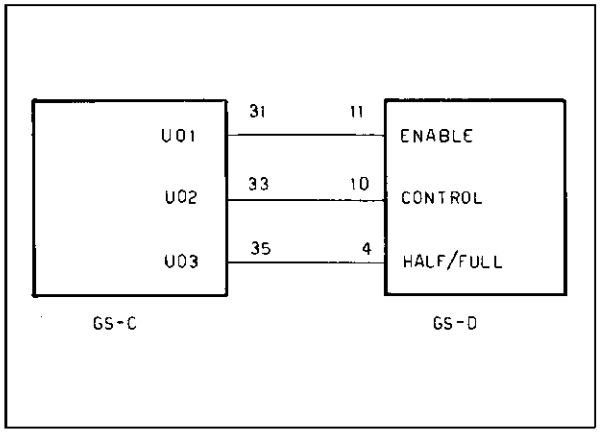
The USER interface consists of three inputs and three outputs which are TTL compatible. They can be read and/or activated during the execution of a program under the complete user control; therefore they condition a program execution.

These signals allow the implementation of complex movements, minimizing the program length and the use of external hardware. The start of a movement or of a sequence can be conditioned by a logic level applied to one or more inputs, thus performing the "mechanical tree" function.

The USER outputs logic state is set by program instructions and this information can be used by other controllers to synchronize multiple movements or to control external drivers.

By using only these signals, it is possible to build up simple systems which implement cyclic movements and create a true stand-alone system. The example reported in figure 7 shows one of the possible utilization of USER output. The example

Figure 7. USER Output Applicative Example



refers to a complete motion control system implemented by using the GS-C200 controller and the GS-D200 Sequencer-Driver. The USER output UO1 is used to enable the GS-D200 (UO1 High) or to inhibit it (UO1 Low).

The USER output UO2 is used to select the motor current decay inherent to the chop mode control of GS-D200. When UO2 is high a slow decay is imposed to the phase current during recirculation; when UO2 is low a fast decay is selected.

The USER output UO3 allows the selection between the half and full-step mode of operation of the GS-D200. Half-step occurs when UO3 is high.

The GS-C200S is capable of executing a jump command either direct or conditioned by the logic status of the USER inputs. This capability is very useful because it allows complex programs to be written by using a limited number of instructions. This feature makes also possible to have a segmented program contained in the internal memory; the selection and the subsequent execution of the needed program segment is started by a specific logic status applied to the USER inputs.

THE S.I.M.P.L.E. COMMAND INTERPRETER AND EXECUTOR AND THE PROGRAMMING LANGUAGE

The GS-C modules contain an interpreter program named S.I.M.P.L.E., acronym for **S**GS-THOMSON **I**nteractive **S**tepper **M**otor **P**rogram **L**anguage and **E**xecutor, that recognizes simple mnemonic commands, verifies the correctness of the received commands and executes the instruction sequences of each command or a complete command sequence by translation into complex executable instructions. The interpreter recognizes three different types of commands:

DIRECT EXECUTION COMMAND

DELAYED EXECUTION COMMAND

UTILITY COMMANDS

Direct execution commands are immediately actuated. They include: start and stop the program execution, set the programming mode, check position, check I/O, etc...

Delayed execution commands are run when requested by the sequence currently stored in memory. By using a combination of these commands, it is possible to perform very complex movements including also the conditioning by external stimulus, the iteration of a specific sequence for a defined number of times.

Utility commands allow the GS-C modules to perform several additional functions such as the detection of the position, phase current optimization etc... These commands, when properly used, speed up the system debugging phase and they increase the system efficiency.

Note: To easily learn how to program the GS-C and to minimize development time, a P.C. based self explaining and interactive program named **F.A.S.T.** (**F**irst **A**dvanced **S**tepper motor **T**raining program), able to communicate with the module by using the Point-to-Point protocol, has been developed and it is available to the end user. (See GS-C200PROG data sheet).

Command strings can be easily implemented also by using a high level language such as BASIC, or they can be generated by a dedicated microcontroller programmed in machine language. The dialogue speed is limited by the time required to construct the command string and to analyze the GS-C data, and it results noticeably reduced when a "machine language" program is used.

The program, after testing, can be stored in the EEPROM included in the GS-C module and then loaded and automatically executed at power-up, resulting in a low cost stand-alone system. It is also possible to save the program as a DOS file on a floppy disk for future retrieval, or to ease the field update of the program itself.

Every command is identified by one or two characters and by a variable length argument (from 0 to 7 characters). If the Multipoint communication protocol is used, the address is specified by the number that preceeds the command. All the commands sent by the Host, as well as the data generated by the GS-C, are terminated by a Carriage Return (ASCII value = 0D).

In the following pages all the commands which may be executed by the GS-C200 and the GS-C200S are detailed, as well as their format. A practical example of the command usage is also given. The presence of an asterisk at the end of the command denotes that the command is executable only by the GC-C200, while two asterisk denote a command executed only by the GS-C200S.

Each command is shown in the same format used during the programming phase, i.e. the command identifier plus the argument:

Gsxxxxxxx

The argument can be single, double or missing according to the various command types.

The various argument are identified by different letters according to the particular type i.e.:

s = sign + or –

x = figure 1 to 3

y = figure 0 to 7

v = value 1 to 999 depending on command

p = position 1to 999999 incremental or the absolute position

Apart the different number of executable commands and functions, the GS-C200S and the GS-C200 look very similar each other. The only foundamental difference is the way they manage the position counter.

The position counter is the reference ruler for the microprocessor to move correctly from the actual position to the targeted one, executing the proper number of steps in the right direction.

The GS-C200 position counter allows a maximum of ten million steps to be executed, and the home position corresponds to the 0 count position. When a movement is larger than the position ruler limits an Error 5 is reported to the Host.

The GS-C200S position counter allows a maximum total count of 2^{24} step ranging from –8388608 to +8388607 steps. When the maximum count is exceeded the counter wraps-around. For example if the position counter is +8388606 and a +5 steps movement is executed, the final position will be:

+8388606 Initial position

+8388607 After 1 step execution

–8388608 After 2 steps execution

–8388607 After 3 steps execution

–8388606 After 4 steps execution

–8388605 Final position

Of course no error is reported.

Command	Description
Ax	The **Activate** command sets a User output to the active logic level "one". The command is always followed by an argument whose value ranges between 1 and 3, and that specifies the User output to be activated. The command string: A2↓ causes the UO2 output to be set to the logic level "one". The **Activate** command is of the delayed execution type and it occupies 2 memory locations.
Cx	The **Clear** command clears a User output, i.e it forces the logic level to "zero". The command is always followed by an argument whose value ranges between 1 and 3, and that specifies the User output to be cleared. The command string: C3↓ cleares the UO3 output by forcing it to the logic level "zero". The three USER outputs are automatically cleared at power-up. The **Clear** command is of the delayed execution type and it occupies 2 memory locations.
Dvvv	The **Delay** command allows the execution of a delay. The instruction is always followed by an argument whose value ranges between 1 and 255, and that specifies the duration in tenth of sec. of the delay to be executed. The command string: D15↓ causes a 1.5 seconds delay to be executed before the next instruction is considered. The **Delay** command is of the delayed execution type and it occupies 2 memory locations.
E	The **Execute** command starts the execution of the program stored in memory. It is also used to terminate the GS-C200 programming session and no argument is required. The **Execute** command is of the immediate execution type.
F	The **Feedback** command allows the host computer to know whether the controller is ready to receive a command or not. To comply with this request, the GS-C replies by: B↓ (Busy) in case it is executing a program, or: R↓ (Ready) if it is ready to receive a command, or: E5↓ (Error) This latter answer, used only by the GS-C200, indicates that during the program execution the position counter has reached the overflow condition (i.e. > 9999999). The feedback command is of the immediate execution type.
fsxxxxxxx**	The **force** command, executable only by the GS-C200S, allows the user to preset the position counter to the desired value. This command is always followed by the sign and the value of the position that spans from – 8388608 to + 8388607. The **force** command is of the delayed/immediate execution type and it occupies 4 memory locations.
Gxxxxxxx*	The **Goto** command forces the motor to reach the specified target position. This command, executed exclusively by the GS-C200, is always followed by an argument whose value ranges between 0 and 9999999, and it defines the position to be reached. The 0 position coincides with the Home position or with the position where an Initialize command has been sent. The **Goto** command is of the delayed execution type and it occupies 4 memory locations.

Command	Description
Gs *	The "velocity mode" **Goto** command allows to move the motor continuously, i.e. the motor is accelerated to the programmed speed and then it slews indefinitely in the selected direction until a "stop" command is received. The command is always followed by the direction information. The command string : G+↓ move the motor in the clockwise direction while: G–↓ move the motor in the counterclockwise direction. To stop the motor either the Z or the K command can be used. The "velocity mode" **Goto** command is of the delayed execution type and it occupies 4 memory locations.
Gsx *	A further possibility offered by the Goto command, that greatly improves the GS-C200 flexibility, is the "controlled velocity mode" operation. This command is followed by the direction information and by the User input to be tested to stop the operation. This occurs when a "zero" level is applied to the specified User input. For example the command string: G+2↓ causes the motor to ramp to the programmed slew speed and to move at this speed until the UI2 input is brought to "zero". The "controlled velocity mode" **Goto** command is of the delayed execution type and it occupies 4 memory locations.
Gsxxxxxxx**	The Goto command forces the motor to reach the specified target position. This command, executed exclusively by the GS-C200S, is always followed by an argument whose value ranges from – 8388608 and + 8388607, and it defines the position to be reached. The 0 position coincides with the Home position or with the position where an Initialize command has been sent. The **Goto** command is of the delayed execution type and it occupies 4 memory locations.
gs **	The "velocity mode" **Goto** command allows to move the motor continuously, i. e. the motor is accelerated to the programmed speed and then it slews indefinitely in the selected direction until a "stop" command is received. g+1↓ move the motor in the clockwise direction while: g–↓ move the motor in the counterclockwise direction. To stop the motor either the Z or the K command can be used. The "velocity mode" **Goto** command is of the delayed execution type and it occupies 4 memory locations.
gsx **	An additional possibility offered by the **Goto** command, further improving the GS-C200S flexibility, is the "controlled velocity mode" operation. This command is followed by the direction information and by the User input to be tested to stop the operation. This occurs when a "zero" level is applied to the specified User input. For example the command string: gt1↓ causes the motor to ramp to the programmed slew speed and to move at this speed until the UI1 input is brought to "zero". The "controlled velocity mode" **Goto** command is of the delayed execution type and it occupies 4 memory locations.

Command	Description
Hs	The **Home** command allows the GS-C to find the mechanical reference position. The command is followed by the argument that specifies the searching direction of the End Of Travel switch. The argument can be omitted and in such a case the GS-C will execute the command: H+↓ As soon as the GS-C receives the **Home** command, it moves the motor in the selected direction at the Start-Stop speed (defined as the first instruction at the beginning of the program) until the End Of Travel input is brought to "zero". When this condition is reached the direction is reversed and the movement continues until the Home input reaches the "zero" logical level. The position counter is then cleared as well as the program contained in the RAM memory, and the controller is ready to process a new command. In the GS-C200S, the position is also cleared, but the previous program, present in the RAM is saved. When the Home and the End Of Travel inputs are tied toghether the system reference point will correspond to the End Of Travel position. To allow the system homing also in a stand alone application, an **Home** command is automatically executed at start-up after the program recall. The Home direction is defined by the logic state of the RxD input (pin 11) that when unconnected is equivalent to a **H+** command, while when connected to the +5V pin it forces a **H–** command. In a stand-alone environment, when the **Home** command is not needed, it is mandatory to ground the End od Travel and the Home inputs (pins 32 and 30). The **Home** command is of the immediate execution type.
Ix	The **Initialize** command forces the GS-C module to be selectively inizialized. The command is followed by an argument whose value ranges between 1 and 3, and that specifies where the action is addressed according to the following table: 1 = Position counter is cleared 2 = User outputs are cleared 3 = Position counter and User outputs are cleared. The **Initialize** command is used to create a logic Home position for the GS-C200 if the 9999999 steps are not enough for the specific application. This function is better performed by the force command in the GS-C200S, for which it is also possible to insert this command into the program. The **Initialize** command is of the immediate execution type for the GS-C200, while it results of the delayed/immediate execution type for the GS-C200S and it occupies 2 memory locations.
jv **	The **jump** command, executed only by the GS-C200S, allows the user to move inside the program and to repeat indefinitely a portion of the program itself. The argument specifies the memory location to be reached and it ranges from 0 (that is the program starting point) to 118. The **jump** command is of the delayed execution type and it occupies 2 memory locations.
jcv,y **	The **conditional jump** command, executed only by the GS-C200S module, allows the user to move inside the program as a function of the logic state of the User inputs. The argument specifies both the memory location to be reached (**v**), that must range between 0 and 118, and the User input condition to be matched (**y**) in order to execute the **conditional jump.** The following example shows how powerful this command is: jc0,40↓ jc1,52↓ jc2,74↓ When the first command is encountered the module tests the status of the User input pins and if their value is 0 a jump to the memory location 40 is executed. If the condition is not met the jump is not executed and the following instruction is examined, and so on. The **conditional jump** command is of the delayed execution type and it occupies 2 memory locations.
K	The **Kill** command aborts the program execution. The program can be restarted just by issuing the Execute instruction which will start the sequence from the first program instruction and not from the interrupt point; it is therefore advisable to always send a Home instruction after a **Kill** instruction in order to allow the system to start from a known position. The **Kill** command is of the immediate execution type.

Command	Description
Lo	The **Loop start** command marks the memory location where the portion of a repeatedly executed command sequence begins. This command is normally used together with the Loop repetition number command. The **Loop start** command is of the delayed execution type and it occupies 2 memory locations.
Lxxx	The **Loop repetition number** command allows an instruction, a sequence or a whole program to be repeated for the specified number of times. The command must be followed by an argument ranging from 1 to 255, that specifies how many times the portion of the program contained between the Loop start command and the **Loop repetition number** has to be executed. The sequence: L0↓ • • • L10↓ forces the command sequence included between L0 and L10 to be repeated ten times. This command in normally used togheter with the Loop start command. If the loop starting point is not specified, the interpreter repeats the sequence starting from the beginning of the program. The **Loop repetition number** command is of the delayed execution type and it occupies 2 memory locations.
M	The **Memory save** command allows the program currently stored in the RAM memory to be permanently saved in the EEPROM. The program can then be reloaded both automatically or under command. In the first case, it is executed automatically at power on, while in the latter the X command must be issued. The **Memory save** command is of the immediate execution type.
P *	The **Program enter** command sets the GS-C200 in the programming mode and it allows a new program to be entered in the memory. The instruction doesn't require any argument and it causes the cancellation of the program contained in the RAM memory The programming session is terminated by the Execute command. The **Program enter** command is of the immediate execution type.
Po **	The **Program enter** command sets the GS-C200S in the programming mode and it allows a new program to be entered in the memory. The instruction doesn't require any argument and it causes the cancellation of the program contained in the RAM memory. The programming session is terminated either by the program exit or the Execute command. The **Program enter** command is of the immediate execution type.
Px **	The **Program exit** command sets the GS-C200S in the execution mode and it allows the unit to wait for a command. The instruction doesn't require any argument. The **Program exit** command is of the immediate execution type.
Q	The **Query** command instructs the GS-C to send to the Host computer the program currently stored in the RAM memory. Every program instruction is separated by a carriage return (ASCII 13), and the program end is evidenced by the transmission of a message "END" that is the sequence terminator and it must be recognized by the Host. The instruction does not require any argument. The **Query** command is of the immediate execution type.

Command	Description
Rvvv	The **Ramp** command allows the user to define the length of the acceleration and deceleration ramps that are always identical. The command is followed by an argument whose value ranges from 1 to 999 and it determines the number of steps necessary to pass from the Start-Stop speed to Slew speed. The instruction: R50↓ specifies an acceleration or deceleration ramp 50 steps long. When the number of steps to be executed is lower than the length of the two ramps (acceleration and deceleration), the ramping is reversed before the maximum speed is reached to guarantee the correctness of the final position. More than one ramp length can be used during the program execution just by introducing an R command in the proper sequence place. R25↓ 3000↓ • • R85↓ –800↓ • This program executes a 25 steps ramp length for the movements until the R85 command is encountered; from that moment all the movements are executed with a 85 steps ramp length. This feature allows the user to optimize the motion system to adapt for different friction and load conditions. The **Ramp** command is of the delayed execution type and it occupies 4 memory locations.
Svvv	The **Start-Stop** command allows the user to choose the step rate at which the motion is started. The command is always followed by an argument whose value ranges between 1 and 1000 and it corresponds to a **Start-Stop** step rate of 10 to 10,000 steps/second (a by 10 multiplier is used). The range normally used is from 1 to 50 corresponding to a 10 to 500 steps/second rate. The command: S30↓ indicates a 300 step/sec or 300Hz Start-Stop frequency. A **Start-Stop** command must initiate any program to be executed in stand alone environment. More than one **Start-Stop** rate can be used during the program execution just by introducing a new **Start-Stop** command when needed, as shown in the following program sequence: S20↓ T200↓ • • S35↓ • T300↓ • The **Start-Stop** command is of the delayed execution type and it occupies 4 memory locations.
Tvvv	The **Top-speed** command allows the user to choose the motion system Slew speed. The command is always followed by an argument whose value ranges between 1 and 1000 that correspond to a **Top-speed** step rate of 10 to 10000 steps/second (a by 10 multiplier is used). The range normally used is from 30 to 500, corresponding to a 300 to 5000 steps/second rate. The command: T300 indicates a 3000 steps/sec or 3kHz rate (equivalent to 900 turns/minute for a motor with 200 steps/turn). More than one **Top-speed** rate can be used during the program execution just by introducing a new **Top-speed** command in the proper sequence place as per the example reported in the Start-stop speed command description. The **Top-speed** command is of the delayed execution type and it occupies 4 memory locations.

Command	Description
Ux	The **Until** command allows the program currently stored in RAM memory to be continuously executed until a specific USER input is brought to "zero". The command is always followed by an argument whose value ranges between 1 and 3, and it specifies the User input to be tested. The command: U2↓ states that the program, once started, will be continuously executed as long as the User input UI2 is at the logic level "one". Just after User Input UI2 is set to "zero", the program processes the next command after U2. The **Until** command is of the delayed execution type and it occupies 2 memory locations.
Vx	The **Verify** command allows the Host to know the current absolute position of the motor versus the Home position or the status of the USER inputs and outputs. The instruction is always followed by an argument whose value, 1 or 2, specifies the type of requested information. The request for the current absolute position is obtained by issuing the instruction: V1↓ the GC-C200 answer can be 1234567↓ while the GS-C200S answer can be: +1234↓ The request of the USER outputs status is obtained by using the instruction: V2↓ the GS-C answer can be: 25↓ that denotes the following Input/Output status: UI1 = 0 UO1 = 1 UI2 = 1 UO2 = 0 UI3 = 0 UO1 = 1 2 5 The **Verify** command is of the immediate execution type.
X	The **eXchange** command allows the user to transfer the program currently stored in the EEPROM into the RAM. This command is used either during the program debugging phase when the F.A.S.T. program is utilized, or when the fast execution of a frequently used program is needed. In this latter case the Host recalls the program from the EEPROM by simply issuing the following command string: E↓ X↓ The **eXchange** command is of the immediate execution type.
Wx	The **Wait-for** command allows the program start or a portion of program execution to be conditioned by the rising edge of an external signal applied to the a USER input. The command is always followed by an argument whose value ranges between 1 and 3, and it specifies the User input to be tested in order to conditions the next command execution. The instruction: W2↓ states that the program execution is conditioned by the presence of a "one" logic level at the User Input UI2. The **Wait-for** command is of the delayed execution type and it occupies 2 memory locations.

Command	Description
Z	The **Zero the speed** command allows a smooth stop of the motion system. When the GS-C receives this command it reduces the stepping rate to "zero" through a deceleration ramp and it stops the program execution. If there is no motion when activated, the program execution is immediately stopped. By using this command it is possible to stop the motor still maintaining trace of the system position. The program can be subsequentely restarted through an E command.
±XXXXXX	The **incremental positioning** command allows the user to perform a movement referenced to the actual position. The command can be issued either with a + or – sign that defines the direction of the motion, and it is followed by an argument ranging from 1 and 999999 that defines the number of steps to be executed. The **Incremental position** command can be mixed to the Goto absolute positioning command in a program, and it is normally used in a subroutine. The **Incremental position** command is of the delayed execution type and it occupies 4 memory locations.

During the program execution, the GS-C accepts only the **F**, **Z** and **K** commands. Any other command sent to the GS-C during the program execution has no effect, and the module will respond to the Host Computer by sending the answer B (Busy).

The GS-C200S programming requires a specific attention because, when a program includes a jump command, it is mandatory to address the proper memory position to correctly execute the sequence.

For this purpose it is mandatory to define the jump memory location by adding, for each program instruction, the proper bytes length that is specified in the command description. The program starts from memory address 0.

THE PROGRAM STORAGE AREA

The GS-C contains two storage areas reserved to the User. The first is the microprocessor Random Access Memory from where the motion program is executed, the second is an EEPROM where the programs are saved. The EEPROM contains a program or a command sequence programmed by the user that can be transferred into the RAM memory by using the X command.

The RAM contains either a program or a command sequence sent by the Host computer or transferred from the EEPROM. In any case the program that is executed when an E or Goto command is issued. is the one contained in the RAM.

If the program is sent by the Host, it is checked to verify if the logical and physical correctness has been respected and if the storage capability is not exceeded. In case an error is detected, it is notified to the Host through an appropriated error message.

The number of instructions that can be stored depends on the type of instruction, and typically it ranges between 30 and 60, for a total of 119 memory locations.

THE POWER SUPPLY

The GS-C module contains a high efficiency switch mode power supply. It generates the various regulated voltages required for the proper operation of the internal logic and the communication port, starting from an unregulated input voltage that can range from 12 to 40 Volt. The module also features a 5V output capable of delivering up to 100mA, which can be used to supply external devices or the logic port of a GS-D module. This output is protected against short circuit to ground. Two outputs at ±12V are available with a current capacity of 10mA.

PROGRAM EXAMPLES

After the description of the communication protocol, of the various commands and of the various messages, some simple programs examples are given in the following.

Example 1

The required action is to run a motor at 1000 steps/sec. rate, with a start-stop rate of 100 steps/sec., and a ramp length of 50 steps. The target position to be reached is the step 500000.

The operative sequence is the following:

1) Connect the GS-C200 to an Host Computer equipped with the advanced Basic program.
2) Power-on the GS-C200.
3) Enter the DOS operating system and then run the F.A.S.T. program (see the **GS-C200PROG** datasheet).
4) Start the programming session by typing the following command sequence:

F↓	Read the controller status. A Ready is answered by the GS-C.
I3↓	Clear the position counter and the USER outputs.

P↓	Enter the programming mode
S10↓	Set the Start-stop rate to 100 steps/sec.
T100↓	Set the Slew speed rate to 1000 steps/sec.
R50↓	Set the Ramp length to 50 steps.
G500000↓	Goto the target position
E↓	End of the programming session. The GS-C starts the program execution.

The G500000 command can be substituted by the +500000 command. The program can also be stored in the GS-C EEPROM by typing an M↓ command before the E↓ command.

Example 2

The program chosen for this example drills 5 equidistant holes on a metal bar. A GS-C and GS-D motion system is used to control the vertical position of the drill, while a second GS-C and GS-D motion system is used for the proper bar loading and positioning. To better clarify the operations to be executed and to show the program simplicity, the two command sequences and the relative process flowcharts are also reported.

The programming session is entered following the points 1 to 4 of the previous example. The first command sequence, used to correctly position the metal bar, is the following:

S10↓	Set the Start-stop speed to 100 steps/sec
T100↓	Set the Slew speed to 1000 steps/sec
R40↓	Set the ramp length to 40 steps
W1↓	Wait for the external Start
+250↓	Reach the first drilling position
L0↓	Loop starting point
A2↓	Activate the unit 2 forcing UO2 = 1
D1↓	Wait 0.1 sec
C2↓	Then reset UO2
W2↓	Wait until drilling completion
+120↓	Reach the drilling position 120 steps CW
L4↓	Repeat the loop 4 times
A2↓	Activate the unit 2 forcing UO2 = 1
C2↓	Then reset UO2
W2↓	Wait until drilling completion
+250↓	Reach the cutting position 250 steps CW
A1↓	Activate the cutting blade forcing UO1 = 1
D5↓	Wait 0.5 sec
C1↓	Clear cutting command resetting UO1

The second command sequence, used to drill the metal bar, is the following:

S15↓	Set the Start-stop rate to 150 steps/sec
T200↓	Set the Slew rate to 200 steps/sec
R25↓	Set the Ramp length to 25 steps
W1↓	Wait for start
W2↓	Wait for a drilling command from unit 1
A2↓	Activate the drill motor forcing UO2 = 1
+150↓	Pull down the drill
D1↓	Wait 0.1 sec
G0↓	Lift the drill up
C2↓	Stop the drill motor
A1↓	Notify drilling completion to unit 1 forcing UO1 = 1
D1↓	Wait 0.1 sec
C1↓	Then clear UO1

The combination of these two programs operates only on one bar, then the two GS-C become available again to the Host both for the repetition of the program or for the entering of a new command sequence.

If the operation has to be repeated till the exhaustion of bars, it will be sufficient to add, at the beginning of the first sequence, the command;

U3↓ execute until UI3 = 1

which allows the drilling cycle to continue until the controller which takes care of the bar positioning, is notified to stop the operations.

This notification is accomplished by clearing the User input UI3 of GS-C devoted to the positioning.

To demonstrate the efficiency of the GS-C programming language it is worth to mention that the program for the bar positioning uses 50 memory locations, while the program for the drill control needs only 36 memory locations. The two programs can be contained in the GS-C memory thus making the system simpler and easier to maintain.

Figure 8. Automatic Drilling And Positioning System Block Diagram.

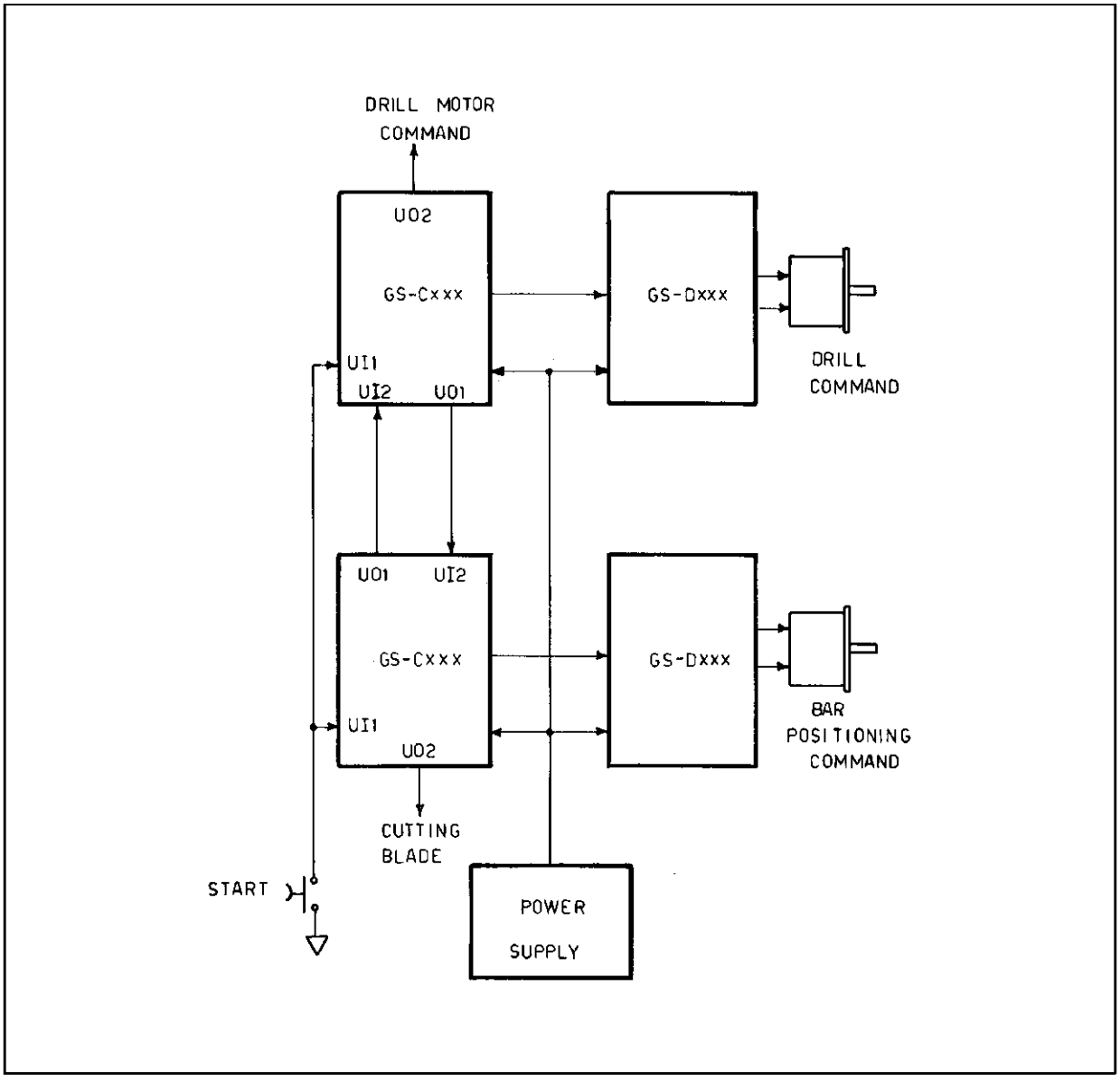
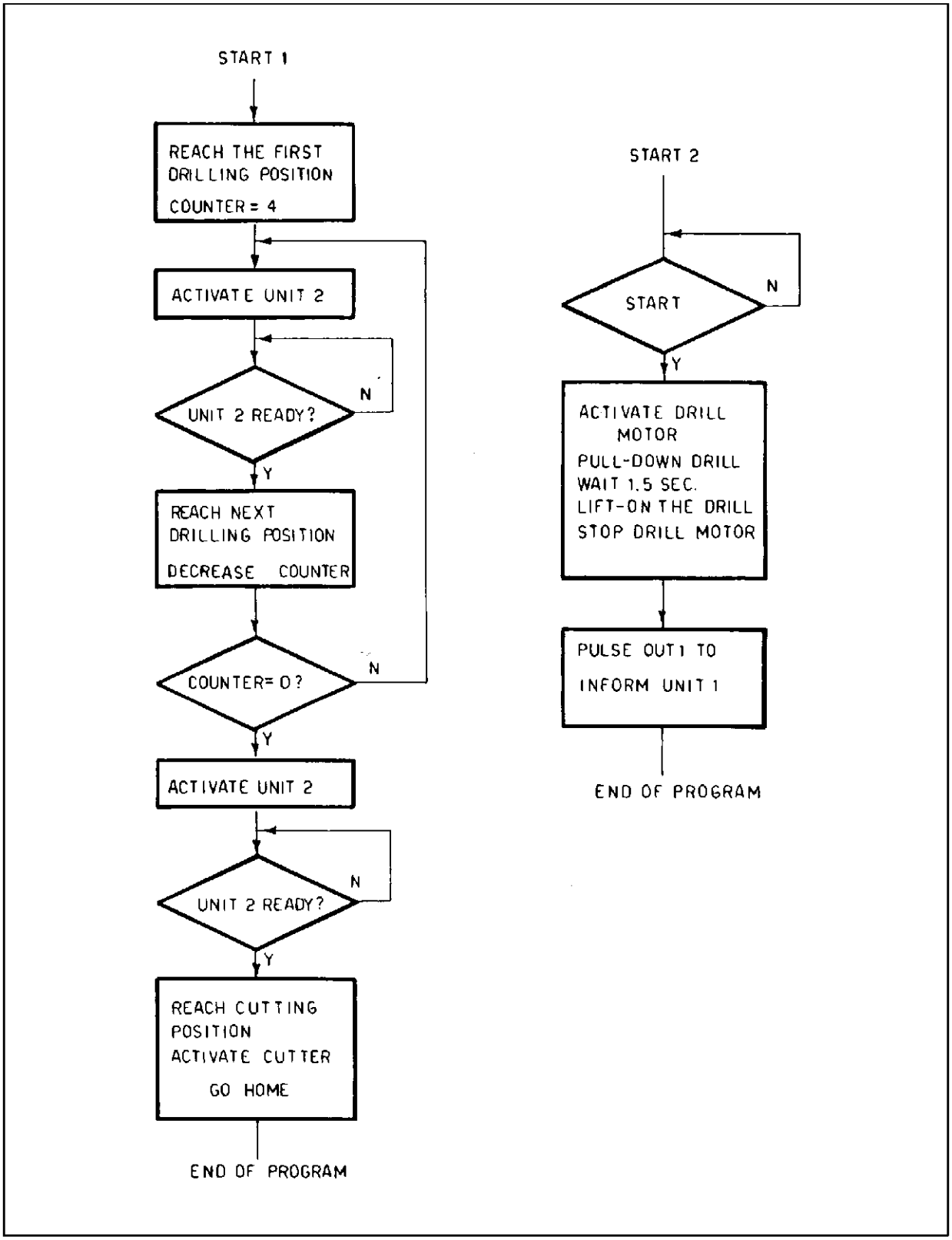


Figure 9. Programs Flow-charts.



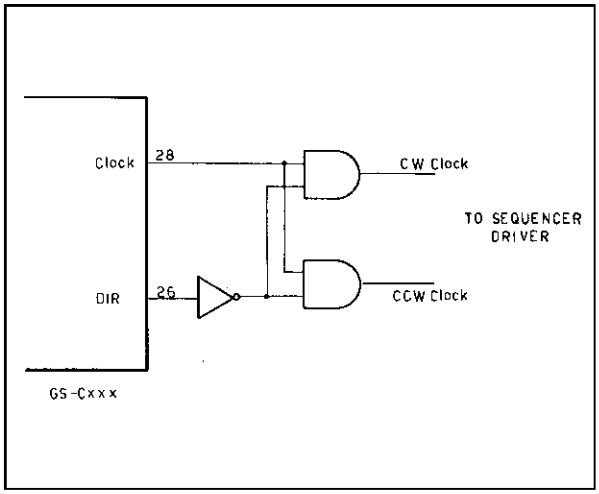
GS-C200 AND GS-C200S APPLICATION

THE SEQUENCER-DRIVER INTERFACE

The GS-C is a general purpose stepper motor controller capable to drive any type of motor, i.e. two, three and five phases motors, by just interfacing it to the right Sequencer-Driver.

Sometime the available Sequencer-Driver requires two separate Clock lines, one for each direction, and this requirement is easily fulfilled by the circuit of figure 10.

Figure 10. Alternative Sequencer-Driver driving.



PHASE CURRENT PROGRAMMING

As already explained, the possibility to modify the phase current of a stepper motor according to different operating conditions, gives substantial improvements in term of efficiency and system reliability because it minimizes the resonance effects and the dissipated power.

The phase current programming can be implemented in various modes, either via a software command by changing the status of the USER output lines, or by hardware. Of course, the Sequencer-Driver must have the current programming capability. An example of a hardware solution, implemented around a GS-C and GS-D200/200S module, is shown in fig.11.

The application utilizes the two outputs:

MOV Movement in execution output (pin 22)

RAMP Ramp in execution (pin 23)

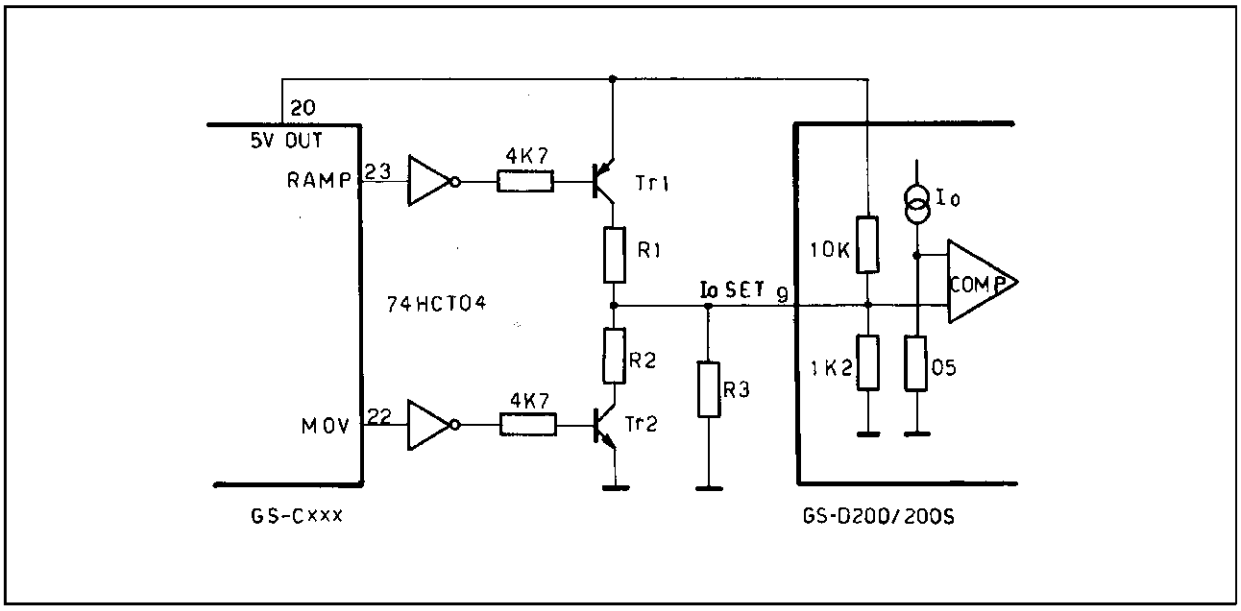
of the GS-C module and the

I_{oset} Current programming input (pin 9)

of the GS-D module.

The phase current has the shape shown in fig.6, i.e. it is minimized when the motor is stopped, it has its maximum value during the acceleration/deceleration ramps, and an intermediate value during the slew phase.

Figure 11. Phase Current Programming of the GS-D200/200S



Let's assume the following values are needed:

I_{rest} = 0.25A

I_{ramp} = 1.5A

I_{slew} = 0.5A

The logic condition of the RAMP and MOV outputs in the various states is:

During the ramping phase both pins 22 and 23 are high: Tr1 is ON and Tr2 is OFF.

During the slew phase pin 23 is low and pin 22 is high: Tr1 and Tr2 are OFF.

In stall condition Tr1 is OFF and Tr2 is ON.

The value of R1, R2, R3 is determined as follows (for further details please see the GS-D200/200S data sheet). The value of R3, that fixes the I_{slew} = = 0.5A (Tr1 and Tr2 OFF), is easily calculated by referring to the GS-D data sheet:

$$R3 = \frac{I_{slew}}{1 - 0.933 \cdot I_{slew}}$$

$$R3 = 937\ \Omega$$

The value of the R2 resistor, when paralleled to R3, fixes the value of I_{rest} = 0.25A (Tr1 OFF, Tr2 ON).

$$R2 \,//\, R3 = \frac{I_{rest}}{1 - 0.933 \cdot I_{rest}}$$

$$R2 \,//\, R3 = 326\Omega$$

$$R2 = 500\Omega$$

The value of R1, that depends on the value of R3 and the resistors contained in the GS-D200/200S module, fixes I_{ramp} = 1.5A (Tr1 ON, Tr2 OFF).

The values of the internal resistors are:

1.2kΩ to ground and 10kΩ to V_{SS} for the GS-D200

750Ω to ground and 10kΩ to V_{SS} for the GS-D200S

Assuming the GS-D200S is used, after some straightforward calculations, it results:

$$R1 = 4245\Omega$$

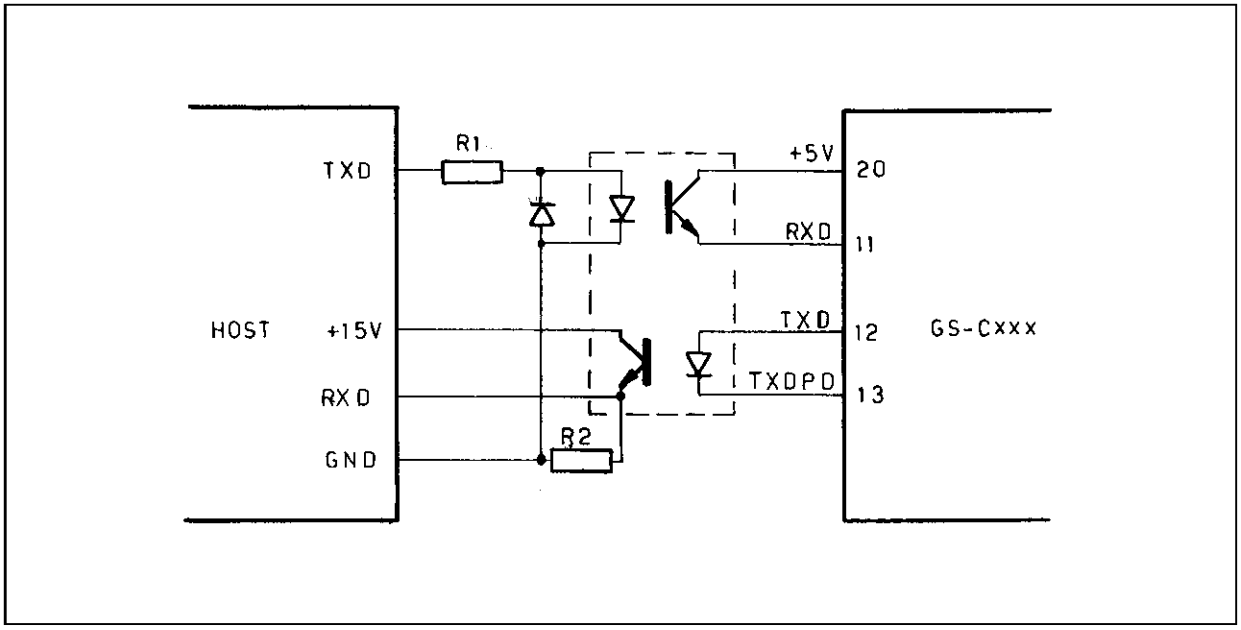
of course all these values do not take into account the transistors saturation losses and in some cases, when a very precise current is needed, a trimming is required.

GALVANIC ISOLATION

The industrial environment, where normally a stepper motor and its driving system operate, is very noisy and for this reason it is often advisable to have a galvanic isolation between the Host computer and the motion system. Because the connection between the Host and the GS-C module requires only three wires (TxD, RxD and ground), the galvanic isolation can be implemented as per fig. 12 that uses only two optocouplers and two resistors, one protection diode and a +12 or +15V source.

A +12 or +15V source is normally available on the pin 6 and 8 of any RS232 connector. The source impedance is quite high (in the range of 220 to 600Ω) and for this reason the value of R2 must be greater than 1000Ω to avoid the source overload.

Figure 12. GS-C200 to Host galvanic isolation.



COMPLEX MOVEMENTS SYNCHRONIZATION

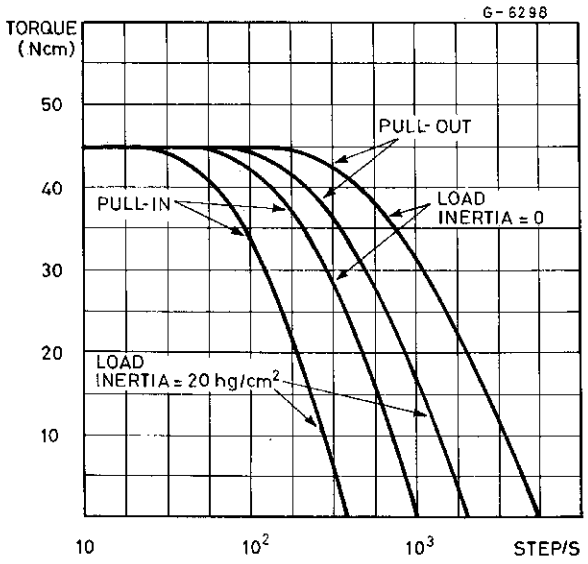
In many applications the synchronization of several movements is quite often required and the GS-C allows this function to be easily implemented either by using the Step Enable input or the User input/output pins. In fig.13A and fig.13B the block diagrams relative to the two solutions are reported.

The solution **A** is the simplest but it has some limitations, i.e. it can be used only when the whole system has to move synchronously. The solution **B** is more complex but also more flexible and it allows the program to control where and when the synchronization must be implemented.

THE START-STOP SPEED (S command) SELECTION

A typical Start-Stop curve (as shown on Fig. 14), shows that for a given driving voltage and phase current, the highest drive frequency allowed at the start (Pull-In Rate) is much lower than the one allowed for the stop (Pull-Out Rate) and that both are influenced by the load value. Of course the higher the current level the higher is the available torque, and the system can be started at a greater speed. A significant increase of the start-stop speed is obtained when the supply voltage is increased but in both cases the problem related to the mechanical resonance must be considered. It is advisable to maintain a significant safety margin against the system torque limit in order to avoid any problem due to the friction variation. A commonly accepted rule fixes the Start-Stop speed equal to the 50% of the maximum theoretical value reported on the motor data sheet; this takes into account friction, load inertia variations as well as motor parameter differences and power supply fluctuations.

Figure 14. Start-Stop Characteristic.



SLEW SPEED (T command) SELECTION

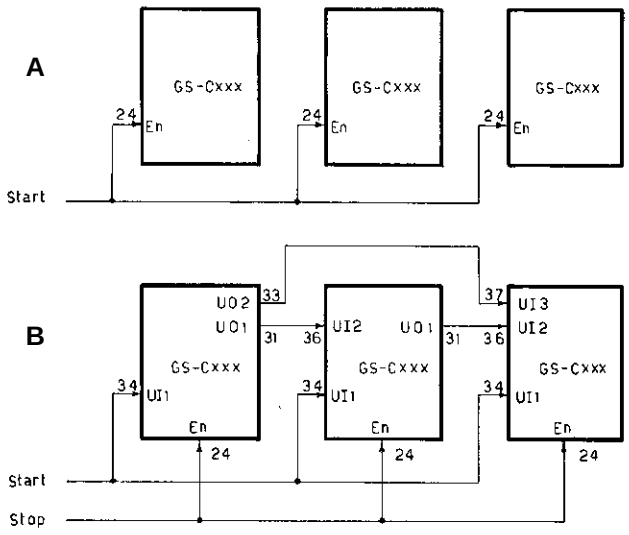
The Slew speed is roughly determined by the load and it can be evaluated by using the following relation:

$$\frac{F \cdot L}{6000 \cdot t} = \frac{T \cdot N}{10}$$

where

F = Strength in Pounds
T = Torque in Ounce/Inch
L = Length in Inches
N = Speed in turn/min.
t = Time in seconds

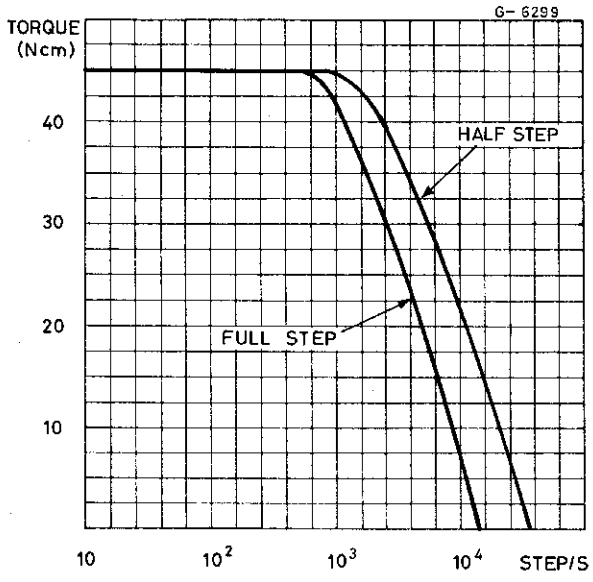
Figure 13. Complex Movements Synchronization



The Slew speed is also limited by the motor electrical and physical characteristics, as shown on Fig. 15 where the behaviour of the minimum available torque versus the driving frequency is reported.

It can be noted that the torque decreases almost linearly starting from a certain frequency, and this frequency depends on the motor windings impedance and the rotor inertia.

Figure 15. Torque/Frequency Characteristic.

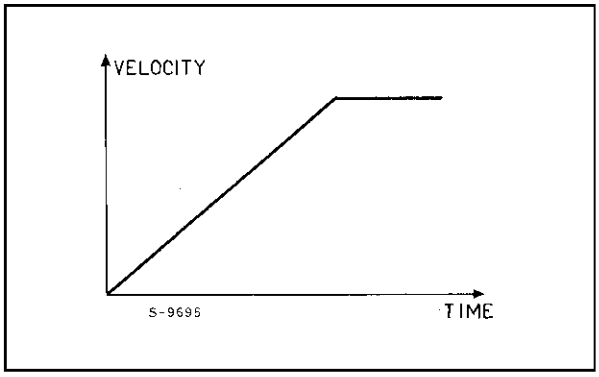


RAMP LENGHT (R command) SELECTION

The acceleration and deceleration ramps are not likely to be calculated and they shall be optimized during the system debugging and testing phase. The testing may start with very conservative ramp gradients, i.e. very long ramps, that will be gradually shortened until the first positioning error is detected.

The acceleration and deceleration ramps generated by the GS-C have the trend shown in fig. 16.

Figure 16. The GS-C200 Acceleration Ramp.



It is important to note that, when the number of steps to be executed does not allow to reach the Slew speed, the GS-C moves to the target position performing a partial acceleration ramp linked to a shortened deceleration ramp. This represents the minimum time consuming way to reach the specified position.

CLOSED LOOP OPERATION

The stepper motor is a device normally driven in an open loop mode and there is no direct control between the cause and the effect. In adverse conditions an issued step may not be performed mechanically because the driving conditions do not match the required torque and speed. In addition, the resonance phenomenon, common to all the stepper motors, can also affect the correct positioning.

In some particular applications, when the load has a very large spread of values and the torque margin is limited, it is sometimes necessary to implement an external electronic circuitry to guarantee the correct system positioning

To this purpose three different methods can be adopted:

a) Digital encoding of the absolute position.

b) Recognition that a step has been executed by the usage of a slotted disk, two optocouplers and some logic.

c) The same as above by the usage of velocity coils and some logic.

The first solution is very expensive and the digitalized position value must be read by the computer through a parallel port by using a specifically written program. A further limitation arises from the fact that every shaft encoder provides just the information relative to the position but it does not take care if more than one turn has been performed by the motor shaft, and an external logic is also required to detect and save this condition.

The second solution is less expensive but it requires a tedious trimming of the mechanical positioning of the optical sources and detectors to be effective. The major drawback of this solution is its sensitivity to dust, and the whole position sensing system must be contained in a dust free box.

The last solution is probably the best under every point of view because it does not require any mechanical positioning adjustement that has been previously made by the motor manufacturer; moreover it is dust insensitive beeing based on flux variation across an air gap and finally no mechanical hardware must be added to the system.

In fig. 17 the block diagram of a closed loop system is reported.

If the step execution is recognized by a movement detector that uses either a slotted disk or the motor velocity coils, two logic signals (x,y) like those reported in fig. 18 are available.

It is possible, by using these two signals as inputs (x, y) of the very simple and inexpensive logic circuit reported in fig. 19, to detect the direction of rotation and the step execution. The output of the circuit is then used to condition the step enable input of the GS-C module allowing the step clock pulse to be issued only if the previous step has been executed.

Figure 17. Closed Loop System.

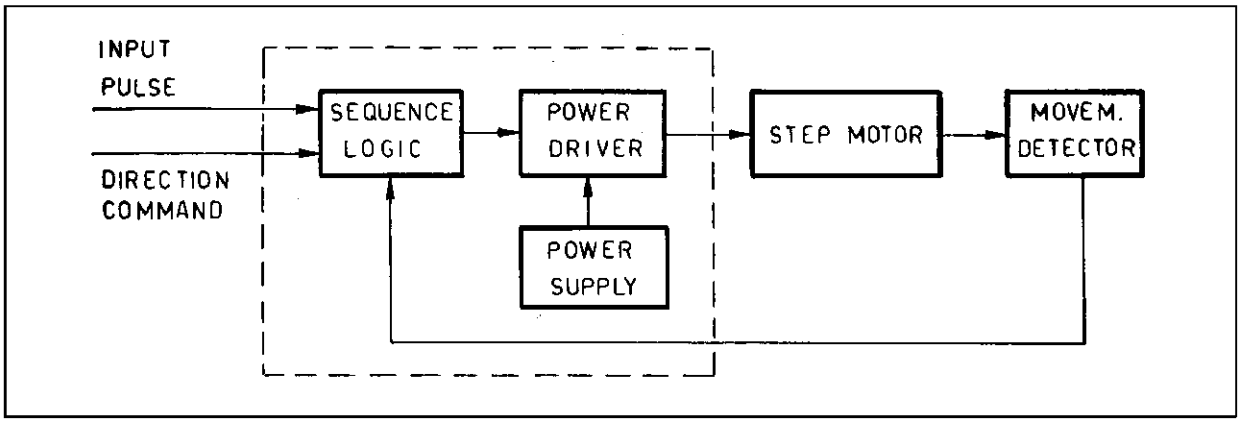


Figure 18. Signal Output of the Movement Detector.

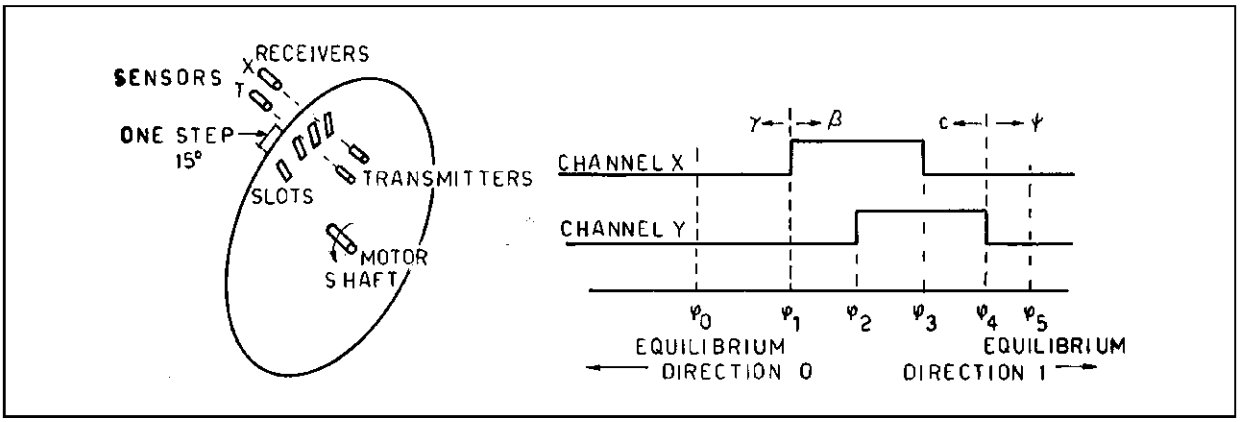
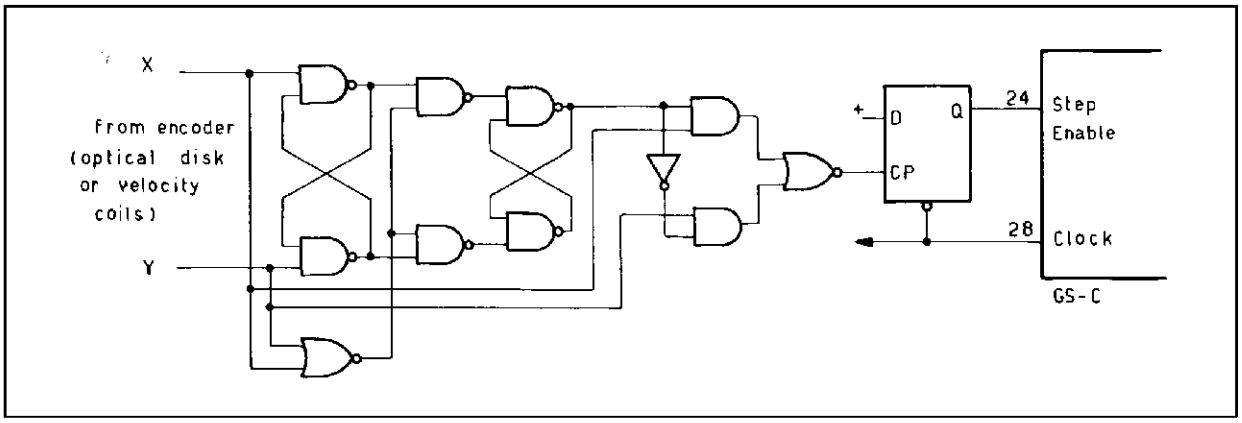


Figure 19. Suggested Logic to Close the Loop.



ELECTRONIC DAMPING

Any stepper motor system when driven at very low stepping rates, has an oscillatory step response as shown in fig. 20.

This oscillatory behaviour is due to fact that the motor reaches the stall position after each excitation change through an acceleration and a successive deceleration. This causes the motor shaft to rotate with jumps instead of uniform motion.

Another consequence of this oscillatory single step response is that the long system settling time can cause mechanical stresses to the driven load.

A second tedious effect is the enhancement of the rotor oscillation when the driving step rate approaches the natural resonance frequency of the motor. If the step rate is lower than this frequency, the motor is behind the equilibrium position and the velocity is near to zero when the next excitation change occurs.

When the step rate is increased to a value close to the natural resonance frequency, an increase of the oscillations also occurs, and as soon as the oscillation amplitude exceeds the step amplitude, the corrispondence between the rotor position and the excitation sequence is lost and any subsequent rotor movement is erratic as shown in fig. 21.

A simple method to reduce the oscillations problem is to use the half step driving, but this also limits the maximum speed of the system.

When this limitation is not acceptable, other two basic techniques may be adopted to damp the system oscillations:

1. A mechanical damper
2. An electronic damping circuit.

Figure 21. Slow Speed Step Response.

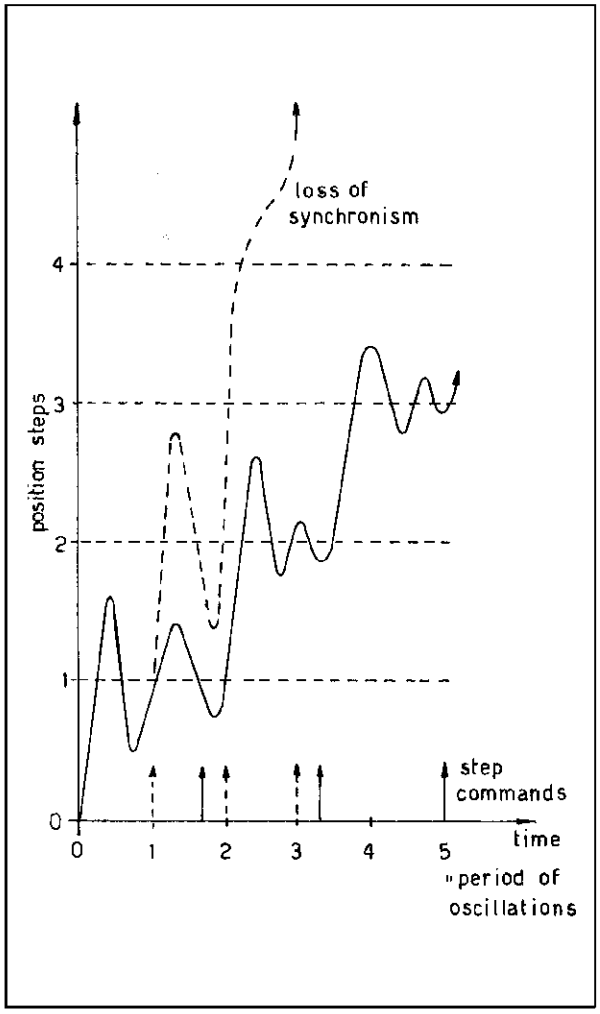
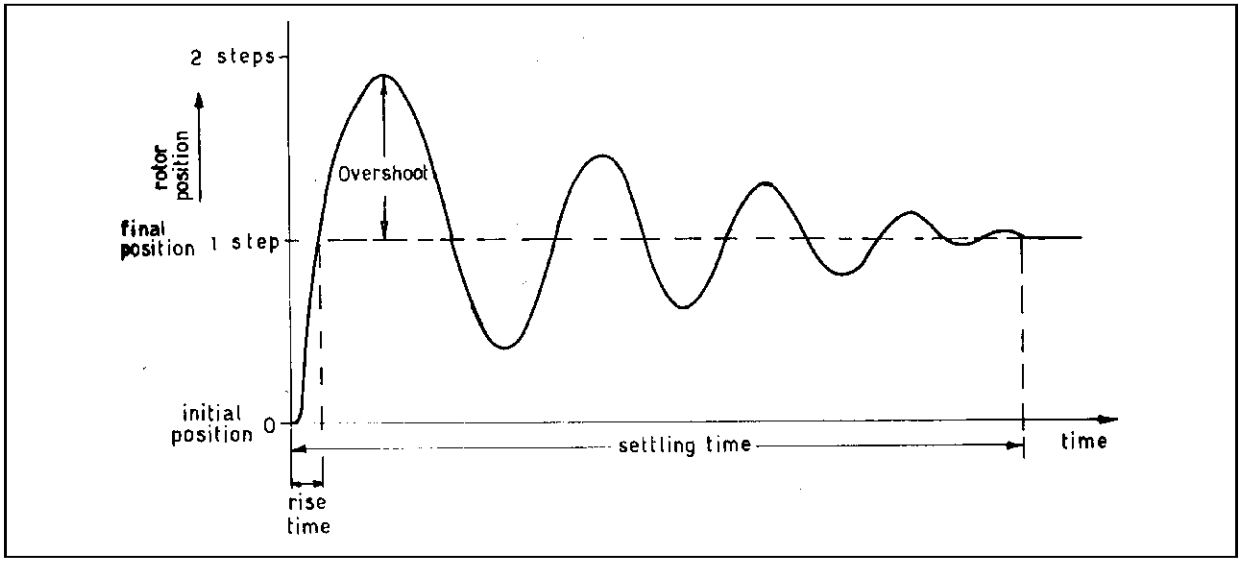


Figure 20. Typical Single Step Response.



The mechanical damping is obtained by the introduction of a viscous friction between the motor shaft and the load. The friction system must be elastic and it will recover the original relative angular shaft alignement to assure the correct final positioning.

The response time of the damping system must be quite fast, and it must be active just for rapid speed changes otherwise a severe limitation in the maximum speed will occur.

The electronic damping is obtained by the proper driving of the motor phases that are switched on and off in such a way to generated a negative torque to decelerate and stop the rotor smoothly. Let's assume the motor is moving from position 1 to the detent position 2, i.e. the phase A is switched OFF and the phase B is switched ON.

The rotor starts moving at t_0 instant (see fig. 22), and after a time t_1, the phase driving is reversed (phase A ON and phase B OFF) generating a braking torque that will allow the rotor to approach the final detent position at a very limited speed. Before the zero speed is reached, (t_2) it is necessary to switch back the phase driving to its original condition in order to stop the system at its target position.

Leaving the phase driving unchanged will cause the motor to stop a step earlier of the correct position because the motor, after the zero speed is reached, will accelerate in the reverse direction returning to the starting position.

The deceleration time as well the damping level is easily adjusted by changing the timing i.e. t_1 and t_2, but it can be quite complicate to compensate a system where large load variation occurs.

In fact, an heavy load variation causes a large variation of the single step response time of the system, and it could be that a system compensated in a no load condition will stop one step behind when fully loaded, while another compensated at full load will probably exibits erratic positioning at no load.

If the load condition is known it is possible to introduce a compensation circuit that can be conveniently driven by one or more User outputs. Fig. 22 shows the motor response to a single step pulse with electronic damping and the relative phase driving. This phase switching reversal method is also known as the bang-bang damping method, and it can be easily implemented by using the GS-C module.

The RAMP and MOV signals allow the user to detect when the last pulses are issued, and to generate, by a simple logic circuit, the delayed phase reversal commands necessary to implement the sequence of fig. 23.

The circuit uses a last pulse detector (G1), and on the falling edge of the A signal (synchronous to the last step command), a timing generator is triggered. The various delays can be trimmed to the values requested by the operating conditions, and the pulse sequence reported in figure 23 (A, B and C signals) in generated.

The A and B signals are used to reverse the motion direction (G_2) while the C signal steps twice the motor (backward and forward).

Figure 22. Single Step Response with Damping.

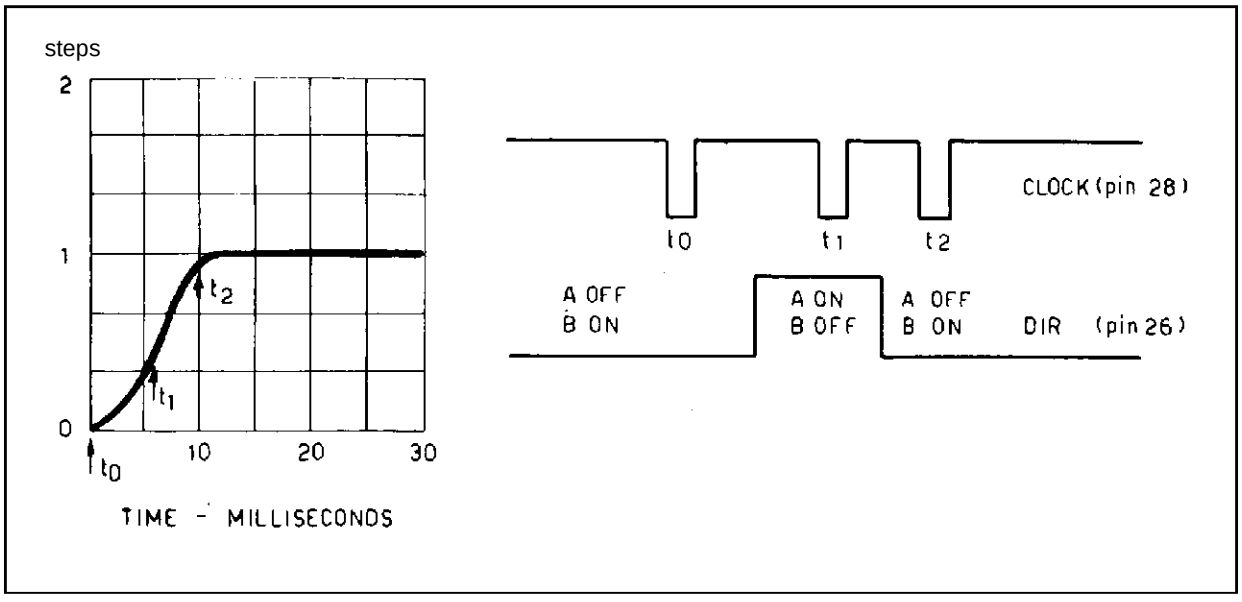


Figure 23. Practical Implementation of the Phase Reversal Damping with the GS-C Module.