AcQuisition Technology bv

Headquarters:
Raadhuislaan 27a
5341 GL OSS
THE NETHERLANDS

Postal address:
P.O. Box 627
5340 AP OSS
THE NETHERLANDS

Phone: +31-412-651055
Fax: +31-412-651050
Email: info@acq.nl
WEB: http://www.acq.nl

M321

Stepper Motor Controller M-module with on-board Power Amplifiers

User Manual

Version 1.3

Disclaimer:

The information in this document has been carefully checked and is believed to be entirely reliable. However, no responsibility is assumed for inaccuracies. AcQuisition Technology does not assume any liability arising out of the application or use of any product or circuit described herein; neither does it convey any license under its patent rights nor the rights of others. AcQuisition Technology products are not designed, intended, or authorized for use as components in systems intended to support or sustain life, or for any other application in which the failure of an AcQuisition Technology product could create a situation where personal injury or death may occur, including, but not limited to AcQuisition Technology products used in defence, transportation, medical or nuclear applications. Should the buyer purchase or use AcQuisition Technology products for any such unintended or unauthorized application, the buyer shall indemnify and hold AcQuisition Technology and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that AcQuisition Technology was negligent regarding the design or manufacture of the part.

Printed in The Netherlands.

CONTENTS

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

1. INTRODUCTION

1.1. VALIDITY OF THE MANUAL

This user manual is of revision 1.3. The manual is valid for the M321/R1.x from AcQuisition Technology bv, running firmware version R4.x.

1.2. PURPOSE

This manual serves as instruction for the operation of the M321 Intelligent Stepper Motor Controller, the connection of stepper motors and the integration on an M-module carrier. Furthermore it gives the user additional information for special applications and configurations of the product. The APIS-based example software and library are also discussed. Detailed information concerning the individual assemblies (data sheets etc.) are not part of this manual. In the annex you will find a bibliography.

1.3. SCOPE

The scope of this manual is the usage of the M321 Stepper Motor Controller M-module with on-board Power Amplifiers.

1.4. DEFINITIONS, ACRONYMS AND ABBREVIATIONS

AcQ	AcQuisition Technology bv
APIS	AcQ Platform Interface Software
M-module	Mezzanine I/O concept according to the M-module specification
Platform	Combination of hardware and operating system

1.5. NOTES CONCERNING THE NOMENCLATURE

Hex numbers are indicated with a leading “0x”-sign: for example: 0x20 or 0xff.

File names are represented in italic: *filename.txt*

A code example is printed in `courier`.

The jumpers are designated by a 'J', and a serial number. When specifying whether a jumper should be connected or removed it is referred to solely by this designation if it has only one position (e.g., 'J5 connected'). However, if the jumper has more than one position, it is also indicated which pins are connected to each other (e.g. 'J8,1-2'). Pin 1 of a jumper is always marked in the configuration diagram.

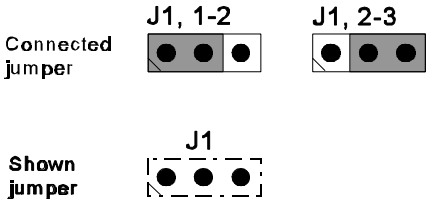


Figure 1 Example jumper nomenclature

In some illustrations jumpers are shown merely for purposes of orientation. In this case they are indicated with a dotted line. Their correct setting is described in another chapter.

Active-low signals are represented by a trailing asterisks (i.e. IACK*).

1.6. OVERVIEW

In chapter 2 a description of the M321 hardware can be found. The next chapter covers the installation and setup of the module as well as the connection of the peripherals. Chapter 4 decribes the operation and usage of the M321 in detail. Commands for the local firmware are explained in chapter 5. AcQ provides APIS based example software for the M321 which is described in chapter 6. The last chapter contains bibliography, component image, technical data, document history and a programming example.

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

2. PRODUCT OVERVIEW

2.1. INTRODUCTION

The M321-module is an Intelligent stepper motor controller M-module, based on an MC68332 micro controller. The M321 has four on-board power amplifiers which can drive four phases. The module can drive two 2-phase motors independently or one 4-phase motor.
Each controller channel has an associated home sensor input up front and a trigger line TRIGA/TRIGB on row-C of the M-module connector.
The MC68332 executes local firmware, downloaded by the host through the M-module interface or booted from ROM. Interaction with the host is done via a command structure in dual ported memory and a mailbox which provides a polling and interrupt mechanism.

2.2. TECHNICAL OVERVIEW

- MC68332 local 32-bit micro controller
- 64kByte RAM shared with the M-module bus
- Up to 1MBit local EPROM or FLASH memory in socket
- A08/D16 M-module interface
- Mailbox with polling, host interrupt and local interrupt mechanism
- Identification EEPROM
- Executes local stepper motor firmware
- Firmware bootable from local ROM or downloadable via the M-module bus
- Firmware compatible with M322 controller for external amplifiers
- Controls two 2-phase motors or one 4-phase motor, jumper selectable
- The two axis in 2-phase controller mode are independent
- Interrupts on position breakpoint, trajectory complete, etc
- Four on-board power amplifiers deliver up to 3A @ 55VDC per phase
- Motor power up to 300W with forced cooling
- Motor power up to 50W without additional cooling
- Amplifiers are bi-polar driving circuits in constant current driving mode
- Over-current protection and thermal shutdown
- Programmable hold current reduction
- User configurable maximum current
- Brake support
- Full-, half-, and micro stepping capability
- Micro stepping: up to 16 micro steps per full step
- Two optical isolated industrial home sensor inputs
- Step rates of up to 50 kHz
- Programmable acceleration of 50Hz/s to 5000 kHz/s
- Synchronous motion via TRIGA and TRIGB signals of the M-module interface.

Intentionally left blank.

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

3. INSTALLATION AND SETUP

3.1. UNPACKING THE HARDWARE

The hardware is shipped in an ESD protective container. Before unpacking the hardware, make sure that this takes place in an environment with controlled static electricity. The following recommendations should be followed:

! Make sure your body is discharged to the static voltage level on the floor, table and system chassis, by wearing a conductive wrist-chain connected to a common reference point.

! If a conductive wrist-chain is not available, touch the surface where the board is to be put (like a table, a chassis etc.) before unpacking the board.

! Leave the board only on surfaces with controlled static characteristics, i.e. specially designed anti static table covers.

! If handling the board over to another person, first touch this persons hand, wrist etc. to discharge any static potential.

IMPORTANT: Never put the hardware on top of the conductive plastic bag in which the hardware is shipped. The external surface of this bag is highly conductive and may cause rapid static discharge causing damage. (The internal surface of the bag is isolating.)

Inspect the hardware to verify that no mechanical damage appears to have occurred. Please report any discrepancies or damage to your distributor or to AcQuisition Technology immediately and do not install the hardware.

3.2. BOOT ROM

The M321 features a 32-pin JEDEC socket that can fit a (FLASH) EPROM containing the firmware. The use of Flash EPROM's, type AM29F010 from AMD, is recommended.
The MC68332 CPU boots from either dual ported RAM or local ROM. When booting from ROM, the local CPU will start running after a system reset. When booting from RAM, firmware must be downloaded and started by the host.
The boot method is configurable with jumper J2 for details please refer to section 3.3.1 and 4.3.

3.3. JUMPER SETTINGS

In the following paragraphs the jumper setting of the M321 is described. The figure below shows the location and orientation of the jumpers.

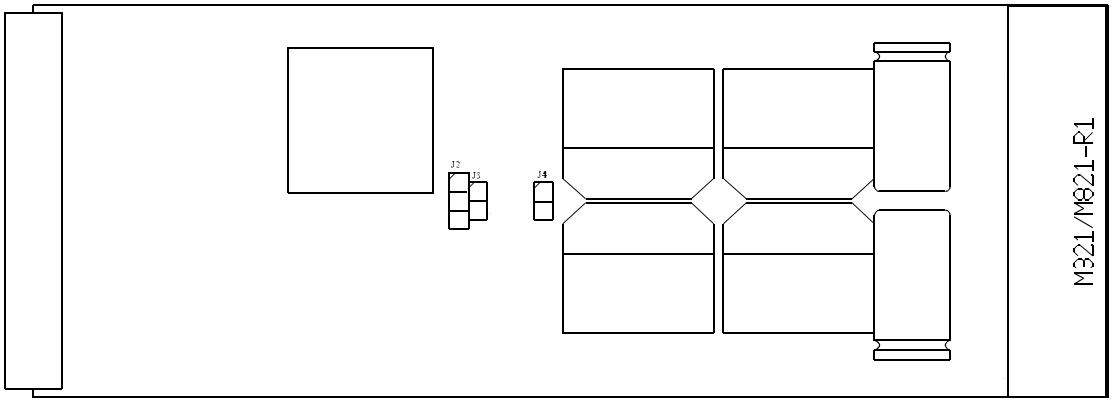


Figure 2 M321 Jumpers

3.3.1. BOOT SELECTION JUMPER J2

With jumper J2, the M321 can be configured for booting from ROM or booting from RAM. The figure below shows jumper J2 positions for both configurations.

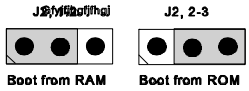


Figure 3 Boot device selection

3.3.2. TWO OR FOUR PHASE SELECTION JUMPER J3

The M321 can be configured for driving either two 2-phase stepper motors or one 4-phase stepper motor. The figure below shows the jumper settings for both controller operation modes.

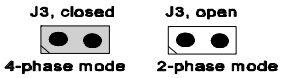


Figure 4 Controller Mode Selection

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

3.3.3. JUMPER 4

Jumper J4 is not user configurable and must be left open.

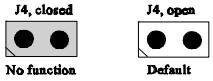


Figure 5 Jumper 4

3.4. CONNECTING THE MODULE

This section gives an overview of the I/O pins of the M321 which are available on the DSUB-25 P3 connector at the front of the module. TRIGA and TRIGB on the M-module connector P1 are also shown.

Signal Name	Pin #	Description
PWRAB	1,14	External power for phase A and B
OUT1A	2	Output 1 phase A
OUT2A	3	Output 2 phase A
SENSEA	15	Current sense output A
OUT1B	4	Output 1 phase B
OUT2B	5	Output 2 phase B
SENSEB	17	Current sense output B
HOMEA+	6	Home sensor input A, anode of opto-coupler input
HOMEA-	19	Home sensor input A, cathode of opto-couple input
PWRCD	13,25	External power for phase C and D
OUT1C	9	Output 1 phase C
OUT2C	10	Output 2 phase C
SENSEC	22	Current sense output C
OUT1D	11	Output 1 phase D
OUT2D	12	Output 2 phase D
SENSED	24	Current sense output D
HOMEB+	8	Home sensor input B, anode of opto-coupler input
HOMEB-	20	Home sensor input B, cathode of opto-coupler input
GND	7,16,18,21,23	Power ground

The picture below shows the connector lay-out.

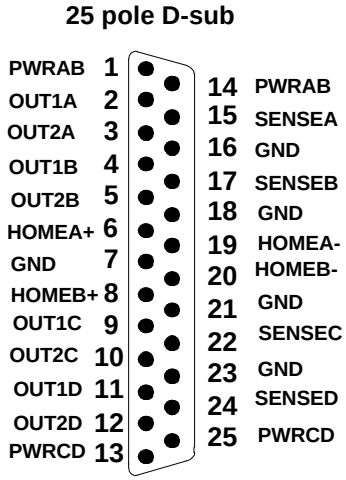


Figure 6 M321 Connector Layout

For cabling, shielded multi-cable must be used with metal DSUB-25 connector shells. The shield must be coaxial terminated to the metal shell.

For details on connecting stepper motors and home sensors refer to chapter 4.5 and 4.6.

The figure below shows the position of TRIGA and TRIGB on the M-module connector P1.

Figure 7 M321 M-module connector orientation

TRIGA and TRIGB are bi-directional and TTL compatible. The signal direction is controlled by the local firmware. For a description of the usage refer to section 5.10. Connections of TRIGA and TRIGB must be made on the M-module carrier board. For a full description of the M-module P1 connector please refer to the M-module specification.

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

4. FUNCTIONAL DESCRIPTION

This chapter gives a detailed description of the M321. The M-module interface is described as well as the power stages and the home sensor inputs.

4.1. BLOCK DIAGRAM

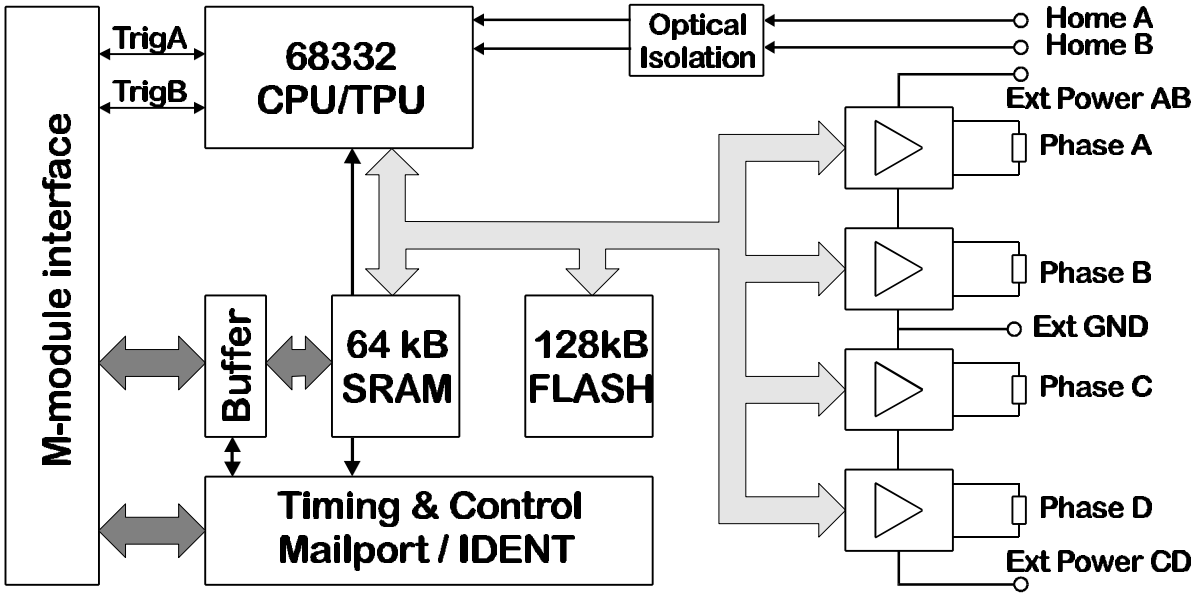


Figure 8 M321 block diagram

4.2. M-MODULE INTERFACE

4.2.1. MEMORY MAP

The following table shows the address map of the M321 module. All addresses are relative to the base address of the module.

Offset	Width	Description
0x00		Dual ported memory window
0x80	16 bit	Page register
0x82	16 bit	Control register
0xfe	16 bit	EEPROM register

4.2.2. DUAL PORTED MEMORY

The M321 features 64kByte dual ported memory. The dual ported memory on the M321 is divided in 512 pages of 128 bytes. The memory can be accessed by the host via a 128-byte memory window and a page register. The dual ported memory window resides at offset 0x00 to 0x7e of the M-module memory map.

CAUTION: The dual ported memory is 16 data bit wide and should only be written using 16 bit write accesses. Byte-writes to the dual ported memory are **NOT** supported, and will result in undefined behaviour.

4.2.3. PAGE REGISTER

The dual ported memory of the M321 is accessed using a page register for the upper address bits. The first 128 bytes of the address space of the M321 M-module is used as a "window" into the dual ported memory as described in the previous section.

PAGEREG Offset: 0x80

15 10	9	8	7	6	5	4	3	2	1	0
not used	A16	A15	A14	A13	A12	A11	A10	. A9	A8	A7
reset	x	x	x	x	x	x	x	x	x	x

Write only

A16-A7 Dual ported memory address bits
These bits determine the current dual ported memory page selected. A page has a length of 128 bytes (A6-A0).

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

4.2.4. CONTROL REGISTER

With the control register the local reset of the M321 M-module can be controlled. Furthermore the control register provides a mailbox. The mailbox controls interrupts to and from the module and provides a polling mechanism.

CTRL Offset: 0x82

15 … 4	3	2	1	0
not used	PBIT	RESET	LIRQ	HIRQ
	0	1	0	0

Read/Write

PBIT Poll Bit
When read, this bit reflects the state of the Poll Bit either ‘0' or ‘1'. This bit can be cleared by the host by writing a one ‘1' and set by the local CPU. Writing a zero ‘0' has no effect.

RESET Local reset
When asserted, the M321 will be kept in reset state and pending interrupt requests will be cleared. After power-up the reset bit is set, however when the module is configured for booting from ROM (J2 in 2-3 position) the reset bit is cleared automatically.

LIRQ Local Interrupt request
When asserted an interrupt request to the local CPU (MC68332) will be made.

HIRQ Host Interrupt Request
This bit reflects the state of the host interrupt line, one '1' means asserted. Writing a one ‘1' to this location clears the pending interrupt.

4.2.5. IDENTIFICATION EEPROM

The identification of an M-module is implemented using a serial EEPROM with a 64*16 word organisation. The industry-standard component 93C46 is used in order to make the identification compatible throughout the complete range of available modules. Access to the identification EEPROM takes place through the following register:

EEPROM Offset: 0xFE

15 … 4	3	2	1	0
not used	unused	CS	CLK	DI/O
	0	0	0	x

Read/Write

CS Chip-Select
This bit corresponds to the chip select input of the EEPROM.

CLK Clock
This bit corresponds to the clock input of the EEPROM.

DI/O Data input/output
This bit corresponds to the data input of the EEPROM when writing, and data output of the EEPROM when reading.

For information on controlling the EEPROM refer to the NM93C46 data sheets. A software example can be found in the file *ideeprom.c* that is part of the software distribution. For information on the memory organization refer to the M-module Specification.

4.2.6. INTERRUPT HANDLING

The M321 is capable of generating interrupts of type A, software-end-of-interrupt. In the interrupt service routine the host must acknowledge the interrupt by writing a one to the host interrupt request bit in the control register. Because the M321 is not capable of delivering an interrupt vector, this must be handled by the carrier board.

4.2.7. TRIGGER LINES

The two trigger lines on the row-C of the M-module connector are used to provide a synchronization mechanism between stepper motor controllers. For details refer to section 5.10.

4.3. BOOTING THE M321

The MC68332 micro controller of the M321 executes firmware from local memory. The module can be configured for booting from ROM or booting from RAM.

4.3.1. BOOTING FROM ROM

For this option an EPROM or FLASH containing valid firmware must be inserted in the JEDEC socket U14 and Jumper J2 must be set in the 2-3 position.
After power-up the module will boot from ROM without host interaction. When the PBIT in the control register is set the module is ready for accepting control commands.

Communication with the M321 takes place via a command structure that resides in the dual ported memory at page 4.

CAUTION: Write accesses to the dual ported memory at pages other than page 4, may result in erroneous behaviour.

4.3.2. BOOTING FROM RAM

When the module is configured for booting from RAM (jumper J2 in 1-2 position), at power-up the M321 is kept in reset state. Before any control operations are available, firmware has to be downloaded to the dual ported memory by the host.
Firmware must copied to the dual ported memory starting at offset 0 in page 0 using 16 bit wide write accesses.

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

The M321 local CPU must be started by clearing the RESET bit in the control register. When the PBIT in the control register is set the module is ready for accepting control commands.

Communication with the M321 takes place via a command structure in the dual ported memory at page 4.

CAUTION: Once the firmware is started any write accesses to the dual ported memory at pages other than page 4, may result in erroneous behaviour.

4.3.3. FIRMWARE DOWNLOAD

The M321 software distribution contains an example for downloading the firmware:

! *m321lib.c*
This is an ANSI-C APIS-based-library that contains host interface functions, one of the functions is `m321_boot()` which loads the firmware image stored in memory. For disk-less applications the firmware image can be supplied as an array of constants declared in ANSI-C source code (*m321firm.c*).

4.4. POWER STAGES

The M321 has four LMD18254 full bridge power amplifiers. Every power stage drives one phase of a bipolar stepper motor. The H-bridge power stages deliver continuous output currents up to 3A at supply voltages between 12 and 55 V.
The M321 can be configured for driving two 2-phase stepper motors independently (two power stages combined, jumper J3 open) or one 4-phase stepper motor (four power stages combined, jumper J3 closed).

NOTE: The maximum power dissipation of the M321 power circuit is 300W with forced cooling. The maximum power without additional cooling is 50W.

A thermal protection circuit shuts down the driver when the junction temperature reaches +155°C. The output current is sensed and controlled independently in each bridge with external sense resistors, internal comparator, and monostable multivibrator. The output current limit can be configured with the external sense resistors.

The figure below shows the power-stage circuitry of phase pair A/B with external sense resistors (the power-stages of phase pair C/D are similar).

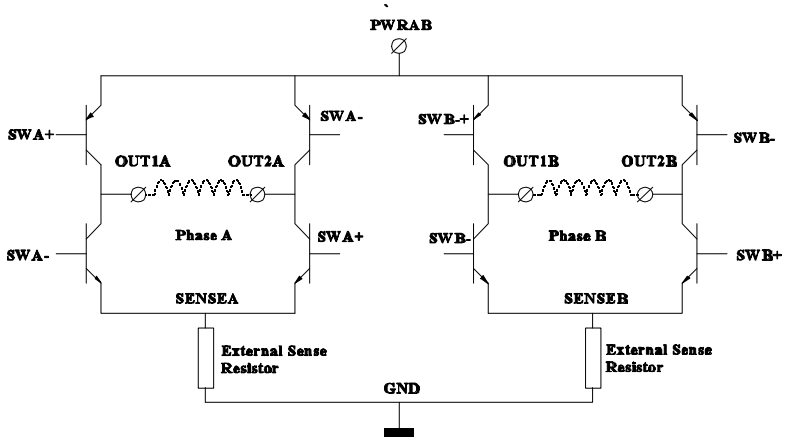


Figure 9 Power-stage pair

! Output Current Limitation

The load current limit is resistor programmable according to the following formula:

$$R_{sense} = \frac{20}{I_{\max}} k\Omega$$

In this formula $I_{\max}$ is the current limit and R_{sense} the sense resistor. The sense resistor must be connected between the sense output of the power stage and the ground. Every power stage has a corresponding sense output (stage A: SENSEA.....stage D: SENSED).

NOTE: The current through the sense resistor has a 1:4000 ratio with respect to the winding current. Therefore the maximum dissipation of the sense resistor is less then 50mW (3A@55V)

! Driving Mode

The power stages are driven in constant current driving mode with an off-time of 50µs. With this method the motor voltage applied to the motor windings must be fairly higher than the rated voltage to keep the current constant.
The power stages can drive bipolar stepper motors. Unipolar motors can be used but must be connected in bipolar mode. The use of motors with low winding resistance is recommended.

By default, stepper motors are driven in full-step mode. However, the M321 is software programmable for driving phases in half-step mode or micro-step mode up to 16 micro-steps per full step.

Micro-stepping is accomplished by gradually increasing and decreasing the phase current, on the M321, this is done by alternating the output current limit dynamically.

! External Power Supply

The external power supply for the drivers is connected to the modules front connector. The power supply consists of two independent circuits, one for the drivers 7A and B and one for the drivers C and D.
The external power supply must be able to drive the motor windings with the configured maximum current I_{max}. The power supply must be in the range of 12 V to 55 V.

For a detailed description please refer to the LMD18254 Data Sheet from National Semiconductor.

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

4.5. STEPPER MOTORS

This section shows how stepper motors can be connected to the M321.

4.5.1. 2-PHASE STEPPER MOTOR

In this mode jumper J3 must be open. The figure below shows how two 2-phase bipolar stepper motors are driven by the M321.

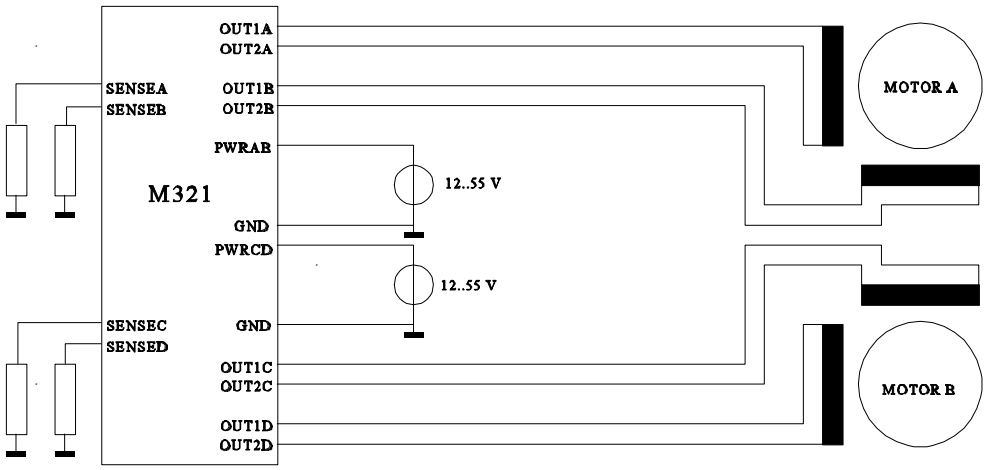


Figure 10 Connecting two 2-phase stepper motors

Although the M321 features bipolar driving circuitry, it is possible to connect unipolar stepper motors. Therefore the winding pairs of the unipolar motor must be connected to one power stage, either in parallel or in serial. Parallel connection is recommended, because constant current driving mode requires a low winding resistance.
For details on driving a unipolar stepper motor with the bipolar driving circuit of the M321 refer to the product documentation of the connected stepper motor.

4.5.2. 4-PHASE STEPPER MOTOR

In this mode jumper J3 must be closed. The figure below shows how a 4-phase bipolar stepper motor is driven by the M321.

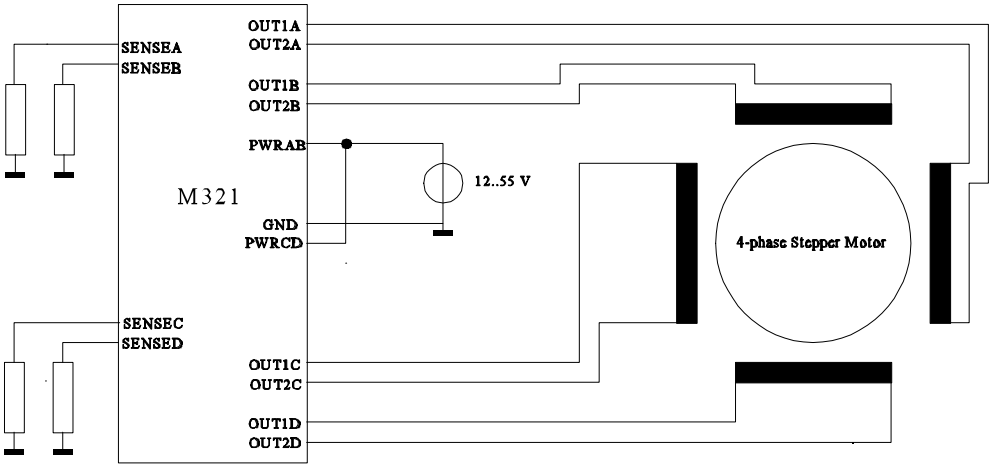


Figure 11 Connecting a 4-phase stepper motor

Depending on the stepper motor, it is possible to connect a unipolar stepper motor in bipolar mode to the M321. Refer to the stepper motor documentation for details.

4.6. HOME SENSORS INPUTS

The M321 features two sensor inputs HOMEA and HOMEB. In 2-phase control mode HOMEA corresponds to stepper motor A (phase A and phase B) and HOMEB corresponds to motor B (phase C and phase D). In 4-phase mode only one stepper motor can be driven, the corresponding home sensor input is HOMEA, HOMEB is not used in this mode.
A home sensor input consists of an opto-coupler input, which is protected for reverse connection and has a leakage current protection. Default the home inputs are rated at 5V@10mA.

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

The figure below shows the input circuitry of home sensor A, home sensor B input is similar.

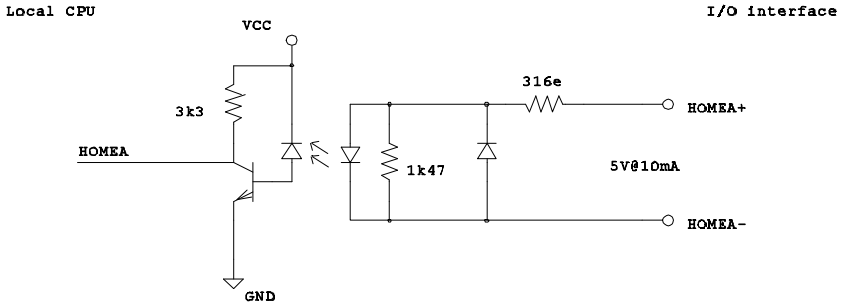


Figure 12 Home sensor input

When the opto-coupler LED is on the logical HOME signal is low, when the opto-coupler LED is off the logical level is high.

To adapt the home sensor input for supply ranges greater then 5V, the current must be reduced to approximately 10mA, by a serial resistor Rs. The table below shows the serial resistors for some supply ranges.

U	Rs
5 V 12 V 24 V	- 620E 1k8

5. THE LOCAL FIRMWARE

This chapter is valid for the local firmware of revision R4.x. The M321 is compatible with the M322 stepper motor controller for external power amplifiers. The local firmware for the M321 is the same as for the M322 and is called ‘m321firm’.

5.1. HOST INTERFACE

Communication between the host and the local CPU takes place through the command structure and the mailbox. The command structure resides at page 4 of the dual ported memory and the mailbox can be accessed via the control register on the M321. The offsets in this command structure are defined in the C-source file *m321defs.h* that can be found in the annex.

The command structure is defined as follows:

```
typedef struct          /* command structure at local address CMDSTRUCT */
{
    UINT16 cmd;         /* command, offset C_CMD */
    UINT16 result;      /* execution result, offset C_RES */
    UINT16 p[32];       /* parameter list, offset C_PRF */
    UINT16 irqstat;     /* interrupt request status, offset C_IRQ */
} CMD;
```

Communication must take place in one of the following two methods:

! Polling
This method uses the POLL bit in the control register. First the host CPU must clear the POLL bit, by writing a logic ‘1', then the host must write the parameters (if any) followed by the command (always last) in the cmd field. The local CPU checks the command and takes the appropriate action. When the command is handled the local CPU writes the return parameters and sets the result, finally the POLL bit will be set which indicates that the command handling is completed.

ANSI-C example of polling based host interface

```
int m321_poll (APIS_HANDLE handle)
{
      UINT16 data;
      int i;
      for (i = 0; i < 1000; i++) { /* Time out mechanism */

            /* read control register-word */
            apis_read(handle, sizeof(UINT16), CTRLREG, &data);
            if (data & POLL)
                  return 0; /* if poll bit is in ready state */
            apis_delay(1);
      }
      return -1;  /* time out! */
}
```


AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

! Host Interrupt.
This method is software selectable using the MSKI command and causes the local CPU to generate a host interrupt request on completion of the command handling. The interrupt service routine of the host must clear the pending interrupt by writing a logic '1' to the HIRQ bit in the control register. In firmware revision R4.0 and up the host must acknowledge interrupts with the IACK command. For details on the IACK and MSKI commands refer to section 5.4.

An ANSI-C example of an interrupt-driven host interface:

```
#define IMASK       0         /* interrupt mask */
#define P_EDGE      0         /* programmable home detection edge */
#define EDGERISE    1         /* Rising edge detection */
#define EDGEFALL    0         /* Falling edge detection */
#define IRQ_HOME_A  0x0020    /* home detected motor A */
#define IRQ_ALL     0x00ff    /* all interrupts */

/* Interrupt-test for a command-executed interrupt */
printf("Test IRQ for HOME input\n");

/*
 * Set Mask to HOME_A_IRQ
 * Give mask-interrupt command
 */
args[IMASK] = (IRQ_HOME_A);
result = m321_cmd(&handle, MSKI, args, 1, 0);
if (result != 0) {
      printf("ERROR: MSKI command failed: 0x%04x\n", result);
      exit(1);    /* Failure */
}

/* Clear all interrupt bits in the status word */
m321_clistat (&handle, IRQ_ALL);

/*
 * Interrupt on falling edge
 * send SIG_HOME_A command
 * await its interrupt
 */
args[P_EDGE] = EDGEFALL;
result = m321_cmd(&handle, SIG_HOME_A, args, 1, 0);
if (result != 0) {
      printf("ERROR: Command failed: 0x%04x\n", result);
      exit(1);    /* Failure */
}

do {
      if (m321_waitforirq() != 0)
            break;      /* interupt received, but not from module */

} while (!(m321_gtistat(&handle) & IRQ_HOME_A)); /* IRQ received? */
```

The table below lists the possible errors and warnings of the result field:

Result Code	Hex	Description
NOERR	0x0000	no error
VIOLATION	0x8888	ERROR: protocol violation
UNKCMD	0x8001	ERROR: unknown command
PRANGE	0x8002	ERROR: parameter out of range
EXCPERR	0x8003	ERROR: general exception error
LOCBERR	0x8004	ERROR: local bus error
LOCAERR	0x8005	ERROR: local address error
LOCILL	0x8006	ERROR: illegal instruction
SPURINT	0x8007	ERROR: spurious interrupt
INVREQ	0x8008	ERROR: invalid request
BUSY	0x8009	ERROR: waiting for external triggering
PROFERR	0x800A	ERROR: error in motion profile
FLASHERR	0x800B	ERROR: flash program error

The table below lists the interrupt status bits of the irqstat field:

Interrupt Status	Hex	Description
IRQ_BLANK	0x0000	Nothing to report
IRQ_CMDDONE	0x0001	Command executed
IRQ_BRKPT_A	0x0002	Breakpoint motor A
IRQ_BRKPT_B	0x0004	Breakpoint motor B
IRQ_TRAJ_A	0x0008	Trajectory complete motor A
IRQ_TRAJ_B	0x0010	Trajectory complete motor B
IRQ_HOME_A	0x0020	Home detected motor A
IRQ_HOME_B	0x0040	Home detected motor B
IRQ_EXCEPTION	0x0080	Internal exception error

An interrupt flag in the interrupt status field set, means that the corresponding event has occurred, the flag must be cleared by the host CPU. Interrupt flags are updated regardless whether or not the corresponding interrupt is enabled.

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

5.2. COMMAND SET

The table below gives an overview of the commands available to the host CPU.

Command	Hex	Description
DONE	0x0000	command completed
SYNC	0x5a5a	synchronize firmware, sign of live
VERSION	0x0001	returns version information
MSKI	0x0020	mask/un-mask interrupts
IACK	0x0030	acknowledge interrupts
SET_MODE_A *	0x0200	set driving mode (full-, half- or micro-stepping) motor A
SET_MODE_B *	0x0201	set driving mode (full-, half- or micro-stepping) motor B
PROF_ABS_A	0x0300	absolute trapezoidal movement motor A
PROF_ABS_B	0x0301	absolute trapezoidal movement motor B
PROF_REL_A	0x0400	relative trapezoidal movement motor A
PROF_REL_B	0x0401	relative trapezoidal movement motor B
GET_POS_A	0x0500	get current position of motor A
GET_POS_B	0x0501	get current position of motor B
SET_POS_A	0x0600	set home position of motor A
SET_POS_B	0x0601	set home position of motor B
SET_BPA_A	0x0700	set absolute breakpoint motor A
SET_BPA_B	0x0701	set absolute breakpoint motor B
SET_BPR_A	0x0800	set relative breakpoint motor A
SET_BPR_B	0x0801	set relative breakpoint motor B
BRAKE_A *	0x0900	shorten motor A windings(phase A and B)
BRAKE_B *	0x0901	shorten motor B windings (phase C and D)
SET_HC_A *	0x0a00	set hold current reduction factor motor A
SET_HC_B *	0x0a01	set hold current reduction factor motor B
SK_HOME_A	0x0b00	seek home motor A
SK_HOME_B	0x0b01	seek home motor B
RD_HOME_A	0x0c00	read home sensor A status
RD_HOME_B	0x0c01	read home sensor B status
SIG_HOME_A	0x0d00	signal transition on home input A
SIG_HOME_B	0x0d01	signal transition on home input B

Command	Hex	Description
SYNC_A	0x0e00	enable synchronized motion A
SYNC_B	0x0e01	enable synchronized motion B
SWSYNC_A	0x0e10	enable software synchronization motor A
SWSYNC_B	0x0e11	enable software synchronization motor B
GO_A	0x0e20	start pending motion A
GO_B	0x0e21	start pending motion B
ABORT_A	0x0f00	abort profile A
ABORT_B	0x0f01	abort profile B
GPOUT0_A **	0x1000	set/clear general purpose output 0 A
GPOUT0_B **	0x1001	set/clear general purpose output 0 B
GPOUT1_A **	0x2000	set/clear general purpose output 1 A
GPOUT1_B **	0x2001	set/clear general purpose output 1 B
GPOUT2_A **	0x3000	set/clear general purpose output 2 A
GPOUT2_B **	0x3001	set/clear general purpose output 2 B
DEBUG	0x8001	service command: debug
FLASHPRG	0x8002	service command: program byte in flash
FLASHVER	0x8003	service command: read byte from flash
FLASHCLR	0x8004	service command: clear flash
FLASHEN	0x8005	service command: enable flash programming

* These commands are only available on the M321. Calling one of these commands on a M322 will result in the error code INVREQ.

** These commands are not available on the M321. Calling one of these commands on a M321 will result in the error code INVREQ.

The following sections give a detailed description of the commands arranged by functionality.

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

5.3. GENERAL COMMANDS

This section describes general commands that are provided for obtaining firmware information.

SYNC — Sign Of Live

Command: 0x5a5a
Inputs: None
Outputs: None

Function:

This function is intended to check whether or not the MC68332 is running. The local software takes no special action and terminates the execution normally: set the result field to NOERR, set the POLL bit in the control register, clear the cmd field and generate a host interrupt if not masked.

VERSION — Get Module Information

Command: 0x0001
Inputs: None
Outputs: VERINFO structure

Function:

This function returns information about the hardware configuration and the firmware. The return information is defined with the following structure:

```
typedef struct {
    UINT16 firm_rev;       /* firmware revision */
    UINT16 mod_type;       /* module type */
    UINT16 ctrl_mode;      /* controller mode */
} VERINFO;
```

The firmware revision is a decimal between 0 and 99, e.g. 10 means revision 1.0. The module type must be 1 which indicates that the firmware is running on an M321.
The controller mode is the jumper configured mode of operation, zero means 2-phase mode and non-zero means 4-phase mode.

5.4. HOST INTERRUPTS

The M321 can generate host interrupts. There are various interrupt sources that use the host interrupt request. Commands are provided to (un)mask and acknowledge interrupts individually. This section gives a description of the interrupt related commands.

An example of an interrupt service routine can be found in *m321lib.c* as:

```
static int m321_irqh (APIS_HANDLE, volatile UINT16 *);  /* int. handler */
```

MSKI — (Un-)Mask Interrupts

Command: 0x0020
Inputs: Interrupt mask
Outputs: None

Function:

A bit set in the interrupt mask will enable the corresponding interrupt, a bit cleared will disable the interrupt. Every interrupt source has a corresponding bit defined in the interrupt mask. The mask is defined as follows:

Flag	Hex	Description
IRQ_BLANK	0x0000	nothing to report
IRQ_CMDDONE	0x0001	command executed
IRQ_BRKPT_A	0x0002	breakpoint motor A
IRQ_BRKPT_B	0x0004	breakpoint motor B
IRQ_TRAJ_A	0x0008	trajectory complete motor A
IRQ_TRAJ_B	0x0010	trajectory complete motor B
IRQ_HOME_A	0x0020	home detected motor A
IRQ_HOME_B	0x0040	home detected motor B
IRQ_EXCEPTION	0x0080	internal exception error
IRQ_ALL	0x00ff	all interrupts

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

IACK — Acknowledge Interrupts

Command: 0x0030
Inputs: Interrupt status
Outputs: None

Function:

The M321 uses one interrupt line for all interrupt sources. Whenever an interrupt request is generated the pending interrupt must be cleared via the control register (see section 4.2.4). The nature of the event(s) responsible for generating the interrupt must be obtained from the interrupt status field in the command register. A bit set means the interrupt request is active. The IACK command must be executed to acknowledge pending interrupts. Parameter 0 must contain the interrupt status.
A bit set in the interrupt status will acknowledge the corresponding interrupt.. Every interrupt source has a corresponding bit defined in the interrupt status. The status is defined as follows:

Flag	Hex	Description
IRQ_BLANK	0x0000	nothing to report
IRQ_CMDDONE	0x0001	command executed
IRQ_BRKPT_A	0x0002	breakpoint motor A
IRQ_BRKPT_B	0x0004	breakpoint motor B
IRQ_TRAJ_A	0x0008	trajectory complete motor A
IRQ_TRAJ_B	0x0010	trajectory complete motor B
IRQ_HOME_A	0x0020	home detected motor A
IRQ_HOME_B	0x0040	home detected motor B
IRQ_EXCEPTION	0x0080	internal exception error
IRQ_ALL	0x00ff	all interrupts

It is allowed to clear more than one pending interrupt simultaneously with the IACK command. As a result of the IACK command not acknowledged (pending) interrupts will cause a new host interrupt request.

The IACK command is a ‘special’ command because on completion no host interrupt is generated regardless wether or not the CMDDONE interrupt is enabled. Also the result field is not affected and the polling bit is not set.

5.5. MOTION CONTROL COMMANDS

The commands in this section are provided to request a motion profile.

PROF_ABS_A — Absolute Motion Motor A

Command: 0x0300
Inputs: Motion profile structure: PROFILE
Outputs: Undefined

Function:

This command is provided to request a trapezoidal movement with an absolute end-position for motor A. The PROFILE structure is defined as follows:

```
typedef struct {
      UINT32 rise_freq;            /* rise-up frequency */
      UINT32 drive_freq;           /* drive frequency */
      UINT32 acceleration;         /* acceleration */
      INT32 position;              /* signed absolute position */
} PROFILE;
```

- rise-up frequency
  The rise-up frequency must have a value between 15 Hz and the requested drive frequency.

- drive frequency
  The drive frequency must have a value between the rise-up frequency and the maximum of 50.000 Hz.

- acceleration
  The acceleration must be between 50 Hz and 5.000.000 Hz/s

- position
  Position is a signed 32 bit value. The stepper motor algorithm maintains a motor position between -2^{31} and $(2^{31}-1)$. At reset the absolute position is 0.

If a profile parameter does not comply with the requirements as mentioned above, the command returns PRANGE.

The motion direction depends on the current position. Internally a relative movement is calculated by subtracting the current position from the required absolute position.

The rise-up and drive frequency are defined in pulses per second. The rotation speed depends on the stepper motor's *number of steps per revolution* and the driving mode (full-step, half-step, etc.).
For instance a stepper motor with 200 steps/rev in half-step mode driven at *drive_freq* has a regular speed of ($drive\_freq$ / 200 / 2) rev/sec.

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

The command will return as soon as the motion profile is started, then the channel is ready to accept new commands. When a profile is completed the IRQ_TRAJ_A flag of the interrupt status in the command structure is set (make sure to clear the flag before the motion command is given).

The figure below shows the trapezoidal drive as a result of a requested motion profile.

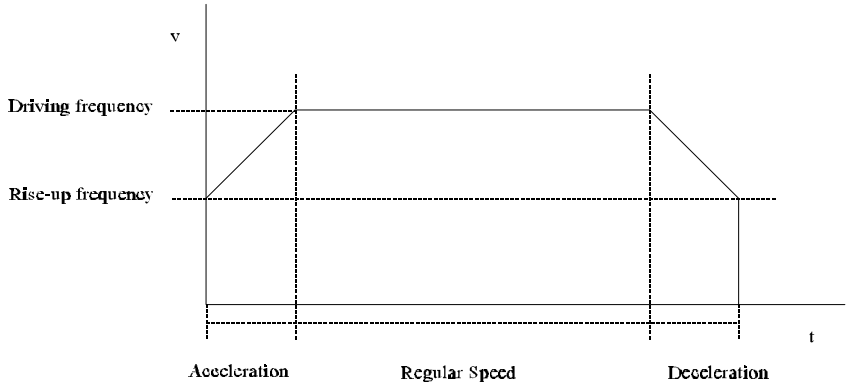


Figure 13 Trapezoidal Motion Profile

On-the-fly changing of the end-point position is allowed with this command. When the new position is in the current direction the end-point is changed, the other profile parameters are discarded and remain as they were. If the new position is in the reverse direction, the current motion is aborted and a new motion profile in the reverse direction is started.

The length of the acceleration trajectory is limited to 5000 pulses and is internally calculated using the following formula:

$$s_a = \frac{f^2_e - f_o^2}{2 \cdot a}$$

s_a = *length of acceleration trajectory in [pls]*
a = *acceleration in [Hz / s]*
f_o = *rise - up frequency in [Hz]*
f_e = *drive frequency in [Hz]*

If the requested motion parameters result in an acceleration trajectory longer then 5000 pulses the code PROFERR is returned.

EXAMPLE:
A motion profile of 200 steps, rise-up frequency of 100 Hz and a drive-frequency of 1000 Hz. The acceleration should be 100 kHz/sec.
The motion profile would look like this:

```
profile->rise_freq = 100;
profile->drive_freq = 1000;
profile->acceleration = 100000;
profile->position = 200;
```

The parameter field of the command structure would look like this:

```
cmd->p[0] = 0x0064;     /* rise-up freq. MSB */
cmd->p[1] = 0x0000;     /* rise-up freq. LSB */
cmd->p[2] = 0x0000;     /* drive freq. MSB */
cmd->p[3] = 0x03e8;     /* drive freq. LSB */
cmd->p[4] = 0x0001;     /* acceleration MSB */
cmd->p[5] = 0x86a0;     /* acceleration LSB */
cmd->p[6] = 0x0000;     /* position MSB */
cmd->p[7] = 0x0200;     /* position LSB */
```

NOTE:
It is the responsibility of the programmer to request a motion profile that suits the connected stepper motor !!!!!

RESTRICTION:
One stepper motor can be driven at the full speed of 50kHz. Two motors can run up to 20kHz simultaneously.

PROF_ABS_B — Absolute Motion Motor B

Command: 0x0301
Inputs: Motion profile structure: PROFILE
Outputs: Undefined

Function:
This command is provided to request a trapezoidal movement for motor B with an absolute end-point.

See Also: PROF_ABS_A

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

PROF_REL_A — Relative Motion Motor A

Command: 0x0400
Inputs: Motion profile structure: PROFILE
Outputs: Undefined

Function:
This command is provided to request a trapezoidal movement for motor A with a relative end-position. This command is comparable with PROF_ABS_A. But now the position parameter is defined relative to the current position. The motion direction depends on the sign of the position parameter.

See Also: PROF_ABS_A

PROF_REL_B — Relative Motion Motor B

Command: 0x0401
Inputs: Motion profile structure: PROFILE
Outputs: Undefined

Function:
This command is provided to request a trapezoidal movement with a relative end-position for motor B. This command is comparable with PROF_ABS_B. But now the position parameter is defined relative to the current position. The motion direction depends on the sign of the position parameter.

See Also: PROF_ABS_A

5.6. Position Maintenance Commands

The following commands are provided for reading and manipulating the current motor positions.

GET_POS_A — Get Current Position Motor A

Command: 0x0500
Inputs: None
Outputs: 32 bit position

Function:
Get the current position of Motor A, 32 bit signed value.

GET_POS_B — Get Current Position Motor B

Command: 0x0501
Inputs: None
Outputs: 32 bit position

Function:
Get the current position of Motor B, 32 bit signed value.

SET_POS_A — Set Current Position Motor A

Command: 0x0600
Inputs: 32 bit position
Outputs: None

Function:
Set the current position of Motor A, 32 bit signed value.

SET_POS_B — Set Current Position Motor B

Command: 0x0601
Inputs: 32 bit position
Outputs: None

Function:
Set the current position of Motor B, 32 bit signed value.

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

5.7. BREAKPOINTS

The following commands are provided to specify breakpoints in a motion profile. On a breakpoint detected, the interrupt status is updated and depending on the interrupt mask a host interrupt request will be generated or not.

SET_BPA_A — Set Absolute Breakpoint Motor A

Command: 0x0700
Inputs: 32 bit position
Outputs: None

Function:

Set an absolute breakpoint for motor A. The input parameter is a 32 bit signed value. The breakpoint is recognized when the motor's current position becomes equal to the breakpoint. This is signalled with the IRQ_BRKPT_A interrupt flag in the interrupt status field of the command structure. When the motor stops on a breakpoint, the breakpoint is signalled only once.

The interrupt status flag must be cleared by the host CPU using the IACK command.

SET_BPA_B — Set Absolute Breakpoint Motor B

Command: 0x0701
Inputs: 32 bit position
Outputs: None

Function:

Set an absolute breakpoint for motor B. The corresponding interrupt flag is IRQ_BRKPT_B.

See Also: SET_BPA_A

SET_BPR_A — Set Relative Breakpoint Motor A

Command: 0x0800
Inputs: 32 bit position
Outputs: None

Function:

Set breakpoint for motor A, relative to the current position. The input parameter is a 32 bit signed value. Internally the absolute breakpoint is calculated as the current position increased or decreased (depending on the sign) with the relative breakpoint.
The breakpoint is reached when the motor's current position becomes equal to the breakpoint. This is signalled with the IRQ_BRKPT_A interrupt flag in the interrupt status field of the command structure.

The interrupt status flag must be cleared by the host CPU using the IACK command.

SET_BPR_B — Set Relative Breakpoint Motor B

Command: 0x0801
Inputs: 32 bit position
Outputs: None

Function:
Set a relative breakpoint for motor B. The corresponding interrupt flag is IRQ_BRKPT_B.

See Also: SET_BPR_A

5.8. ABORT/BRAKE COMMANDS

The following commands are provided to abort motion profiles or to make emergency stops.

ABORT_A — Abort Profile A

Command: 0x0f00
Inputs: None
Outputs: None

Function:
Abort motion profile motor A. This command is provided to abort an active motion of a stepper motor. As a result of this command the motion profile goes directly to the deceleration trajectory.

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

ABORT_B — Abort Profile B

Command: 0x0f01
Inputs: None
Outputs: None

Function:
Abort motion profile motor B. This command is provided to abort an active motion of a stepper motor. As a result of this command the motion profile goes directly to the deceleration trajectory.

BRAKE_A — Assert Brakes Motor A

Command: 0x0900
Inputs: None
Outputs: None

Function:
This command will assert brakes of motor A by shorten the phase windings. Any active motion profile will be terminated immediately. This function is only available on an M321.

WARNING:
This brake command shortens the motor windings, depending on the stepper motor and its load it is possible that the torque is not enough to hold the motor in the same position.

BRAKE_B — Assert Brakes Motor B

Command: 0x0901
Inputs: None
Outputs: None

Function:
This command will assert brakes of motor B by shorten the phase windings. Any active motion profile will be terminated immediately. This function is only available on a M321, configured for two 2-phase motors.

WARNING:
This brake command shortens the motor windings, depending on the stepper motor and its load it is possible that the torque is not enough to hold the motor in the same position.

5.9. HOME DETECTION

The M321 has two home detection inputs. This section describes the home input related commands.

SK_HOME_A — Seek Home A

Command: 0x0b00
Inputs: Motion profile PROFILE and detection edge.
Outputs: Undefined

Function:
As a result of this command motor A starts moving as with the PROF_REL_A command. But when a transition on the corresponding home input is detected the motion profile is aborted, the current position is set to zero and the interrupt flag IRQ_HOME_A is set. If the corresponding interrupt is enabled, a host interrupt is generated. The detection edge is programmable (parm #8) and must be either (0) for falling edge or (1) for rising edge detection.

See Also: PROF_REL_A

SK_HOME_B — Seek Home B

Command: 0x0b01
Inputs: Motion profile PROFILE and detection edge
Outputs: Undefined

Function:
As a result of this command motor B starts moving as with the PROF_REL_B command. But when a transition on the corresponding home input is detected the motion profile is aborted, the current position is set to zero, the current position is set to zero and the interrupt flag IRQ_HOME_B is set. If the corresponding interrupt is enabled, a host interrupt is generated. The detection edge is programmable (parm #8) and must be either (0) for falling edge or (1) for rising edge detection.

See Also: PROF_REL_B

RD_HOME_A — Read Home Sensor A

Command: 0x0c00
Inputs: None
Outputs: Home sensor status

Function:
This command returns the status of the home sensor input A in parameter #0, zero means input low and one means input high.

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

RD_HOME_B — Read Home Sensor B

Command: 0x0c01
Inputs: None
Outputs: Home sensor status

Function:

This command returns the status of the home sensor input B in parameter #0, zero means input low and one means input high.

SIG_HOME_A — Signal Home Detection A

Command: 0x0d00
Inputs: Detection edge
Outputs: None

Function:

This command causes the firmware to signal a transition on the corresponding home input. The transition edge is programmable (parm #0) and must be either (0) for falling edge or (1) for rising edge detection. Signalling is done by setting the IRQ_HOME_A flag in the interrupt status register of the command structure. When the corresponding interrupt is enabled a host interrupt request is generated. After a home signal has been detected the host must clear the related interrupt flag and this function is terminated. For a next home detection the command must be given again.

SIG_HOME_B — Signal Home Detection B

Command: 0x0d01
Inputs: Detection edge
Outputs: None

Function:

Signal home detection for channel B. The corresponding interrupt flag is IRQ_HOME_B.

See Also: SIG_HOME_A

5.10. SYNCHRONIZED MOTION

The M321 features a trigger I/O line for each motor, on row-C of the M-module connector: TRIGA and TRIGB. These lines are provided for synchronizing motion between motors controlled by M321 or M322 modules. All stepper motor channels that have to be synchronized must be connected via their corresponding trigger line.

When this function is enabled for a motor channel the corresponding trigger line is pulled low. When a motion profile is started, the trigger line is switched as an input and the movement begins when every module connected to the trigger line and with synchronous motion enabled, has released the line by switching over to input.

SYNC_A — Enable Synchronized Motion A

Command: 0x0e00
Inputs: Mode
Outputs: None

Function:

This command is provided to enable (parm #0 is 1) or disable (parm #0 is 0) synchronized motion for motor A.

SYNC_B — Enable Synchronized Motion B

Command: 0x0e01
Inputs: Mode
Outputs: None

Function:

This command is provided to enable (parm #0 is 1) or disable (parm #0 is 0) synchronized motion for motor B.

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

5.11. SOFTWARE SYNCHRONIZATION

The M321 provides a set of commands for software synchronization. With this mechanism it is possible to setup a motion profile that becomes pending. With an additional command the pending motion must be triggered, as a result the previously processed motion profile is started immediately.

SWSYNC_A — Setup Software Synchronization A

Command: 0x0e10
Inputs: Mode
Outputs: None

Function:
This command is provided to enable (parm #0 is 1) or disable (parm #0 is 0) software synchronization for motor A. When a motion is requested using PROF_ABS_A or PROF_REL_A with software synchronization enabled, the motion becomes pending. With the command GO_A the motion must be started.

See Also: GO_A, PROF_ABS_A, PROF_REL_A

SWSYNC_B — Setup Software Synchronization B

Command: 0x0e11
Inputs: Mode
Outputs: None

Function:
This command is provided to enable (parm #0 is 1) or disable (parm #0 is 0) software synchronization for motor B. When a motion is requested using PROF_ABS_B or PROF_REL_B with software synchronization enabled, the motion becomes pending. With the command GO_B the motion must be started.

See Also: GO_B, PROF_ABS_B, PROF_REL_B

GO_A — Start Pending Motion A

Command: 0x0e20
Inputs: Mode
Outputs: None

Function:
When software synchronization is enabled with SWSYNC_A and a motion is requested using PROF_ABS_A or PROF_REL_A, the motion becomes pending. With the command GO_A the motion must be started.

See Also: SWSYNC_A, PROF_ABS_A, PROF_REL_A

GO_B — Start Pending Motion B

Command: 0x0e21
Inputs: Mode
Outputs: None

Function:
When software synchronization is enabled with SWSYNC_B and a motion is requested using PROF_ABS_B or PROF_REL_B, the motion becomes pending. With the command GO_B the motion must be started.

See Also: SWSYNC_B, PROF_ABS_B, PROF_REL_B

5.12. POWER STAGE CONTROL

The functions in this section are only available on the M321. If the M321 is configured for 4-phase control then the functions for controlling Motor B are not available.

SET_MODE_A — Set Drive Mode Motor A

Command: 0x0200
Inputs: Drive mode
Outputs: None

Function:
Select a drive mode for motor A. The table below gives an overview of the driving modes:

Mode	Hex	Description	μsteps/ step
FULLSTEP	0x0001	full step	1
HALFSTEP	0x0002	half step	2
QUARTSTEP	0x0004	quart step	4
QUADSTEP	0x0008	quad step	8
HEXSTEP	0x0010	hex step	16
MICROSTEP	0x0020	micro step*	16
ONEPHASE	0x0040	one phase excitation**	1

* Only in 2-phase mode, hex step with torque compensation.
** One phase excitation means that only one phase is powered at a time, only available in 2-phase mode.

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

SET_MODE_B — Set Drive Mode Motor B

Command: 0x0201
Inputs: Drive mode
Outputs: None

Function:
Select a drive mode for motor B.

See Also: SET_MODE_A

SET_HC_A — Set Hold Current Factor A

Command: 0x0a00
Inputs: Hold current reduction factor
Outputs: None

Function:
Set the hold current reduction factor for motor A. When set the hold current is reduced automatically with a programmable reduction factor, after the motor is positioned. Make sure to leave enough current to hold the motor in the same position. The hold current factor can be any integer value between ‘0' and ‘7'. The table below shows the relationship between the reduction factor and hold current.

Factor	Hold Current
0	Imax
1	Imax/2
2	Imax/3
3	Imax/4
4	Imax/5
5	Imax/6
6	Imax/7
7	Imax/8

Imax is resistor configurable refer to section 4.4 for details.

SET_HC_B — Set Hold Current Factor B

Command: 0x0a01
Inputs: Hold current reduction factor
Outputs: None

Function:
Set the hold current reduction factor for motor B.

See Also: SET_HC_A

5.13. SERVICE COMMANDS

The commands in this section are provided for service purposes. The commands are only listed and not explained in detail.

DEBUG — Undocumented

Command: 0x8001
Inputs: Undocumented
Outputs: Undocumented

Function:
This function is provided for service and developing purposes.
For normal operation, do not use this function.

FLASHPRG — Program Byte In FLASH

Command: 0x8002
Inputs: Offset and byte value
Outputs: None

Function:
With this command a byte (parm #1) is programmed into the on-board FLASH memory at the specified offset (parm #0). Flash programming must be enabled with the command FLASHEN. This function is provided for in-system firmware updates, details will be provided with any firmware update. For normal operation, do not use this function.

FLASHVER — Read Byte From FLASH

Command: 0x8003
Inputs: Offset
Outputs: Byte value

Function:
With this command the content of offset (parm #0) in the on-board FLASH memory is returned in parm #1.
This function is provided for in-system firmware updates, details will be provided with any firmware update. For normal operation, do not use this function.

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

FLASHCLR — Clear Flash

Command: 0x8004
Inputs: None
Outputs: None

Function:
This command will erase the on-board flash device provided that programming is enabled with the FLASHEN command.
This function is provided for in-system firmware updates, details will be provided with any firmware update. For normal operation, do not use this function.

FLASHEN — Enable Flash Programming

Command: 0x8005
Inputs: Signature
Outputs: None

Function:
With this command programming of the on-board FLASH memory is enabled. Parameter 0 must contain a signature that must be obtained from AcQuisition Technology bv.
This function is provided for in-system firmware updates, details will be provided with any firmware update. For normal operation, do not use this function.

5.14. M322 COMMANDS

The commands in this section are only available to the M322, and are not explained in detail. Calling these commands will return the result code INVREQ.

GPOUT0_A — Set/Clear GP Output 0 Motor A

Command: 0x1000
Inputs: Output state
Outputs: None

Function:
M322 only

GPOUT0_B — Set/Clear GP Output 0 Motor B

Command: 0x1001
Inputs: Output state
Outputs: None

Function:
M322 only

GPOUT1_A — Set/Clear GP Output 1 Motor A

Command: 0x2000
Inputs: Output state
Outputs: None

Function:
M322 only

GPOUT1_B — Set/Clear GP Output 1 Motor B

Command: 0x2001
Inputs: Output state
Outputs: None

Function:
M322 only

GPOUT2_A — Set/Clear GP Output 2 Motor A

Command: 0x3000
Inputs: Output state
Outputs: None

Function:
M322 only

GPOUT2_B — Set/Clear GP Output 2 Motor B

Command: 0x3001
Inputs: Output state
Outputs: None

Function:
M322 only

5.15. EMERGENCY BRAKE

The firmware provides an emergency brake function. This function cannot be called through the command structure, but by generating a mailbox interrupt to the module.
A mailbox-interrupt can be generated by writing a logic one ‘1', to bit #2 of the control register. As a result of the mailbox interrupt the M321 will immediately abort all motion profiles and shorten the motor windings.

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

6. SOFTWARE

This chapter describes the example software which is available for the M321 Stepper Motor Controller M-module with on-board Power Amplifiers. The example software is available in ANSI-C source code and consists mainly of an M321 function library which provides functions for easy access to the M321 and a demo program which illustrate the usage of the software library.
The M321 library functions are APIS based, physical accesses and interrupt support are handled by APIS, AcQ Platform Independent Interface Software. The next section contains general information on APIS, for detailed information please refer to the APIS' Programmer's Manual.

6.1. APIS SUPPORT

AcQ produces and supports a large number of standard M-modules varying from networking and process I/O to motion control applications. Physically, the M-modules are supported by a large number of hardware platforms: VMEbus, PCI, CompactPCI as well as a wide variety of operating systems: OS-9, Windows NT, Linux etc.
APIS offers a way to program platform independent applications, example- and test software for controlling hardware. Application software written for APIS only needs re-compiling for a particular platform and is operational with little effort (provided that the application is operating system independent).

6.1.1. CONCEPT

Hardware accesses to registers and memory are handled by APIS. Some minor operating system dependent functions frequently used in hardware related software, such as interrupts handling and a delay function, are also provided by APIS.
APIS platform support consists of an Application Programming Interface in the form of definition files coded in ANSI-C and platform dependent modules, e.g. source files, libraries and/or drivers.
In the most simple outline, a platform dependent APIS module consist of nothing more then macro definitions in which APIS calls are substituted by direct hardware accesses. But in most cases an APIS module will consist of a library with interface routines and in some implementations a device driver is needed for interaction with the operating system.

6.1.2. API

The Application Programming Interface for APIS is implemented in two ANSI-C coded definition files: *apis.h* which contains general definitions and *platform_apis.h* which contains platform specific definitions and references to the APIS function calls.
The application source file must include the APIS header file *apis.h*. Porting of the application to a platform, consists of re-compiling the source code with a defined pre-processor macro for selection of the used platform. The APIS header file contains generic APIS definitions and includes a platform specific header file according to the platform selection macro.
APIS calls are translated to the platform specific calls in the APIS header file and the platform specific definition file *platform_apis.h* (*platform* is a name that identifies a hardware and operating system combination, e.g. i4000os9).
The macro PLATFORM must be defined, either via a pre-processor definition provided at compile time or via a macro-definition in the application source.

6.1.3. CODE GENERATION

APIS based example software is available in ANSI-C source code. Source code files must be compiled with the pre-processor definition PLATFORM set to a valid value, conforming the target platform. Building of the example software for the M321 is platform dependent, for details refer to the release notes of the APIS support package of the target platform and the APIS Programmer's Manual.
Examples of APIS supported platforms are i4000os9, i2000dos, i3000win etc.

6.2. TYPE DEFINITIONS AND STRUCTURES

The table below contains a list and description of all types and structures used in the M321 example software (standard ANSI-C types are not listed).

Name	Type	Description
INT8	char	8-bit signed data
UINT8	unsigned char	8-bit unsigned data
INT16	short	16-bit signed data
UINT16	unsigned short	16-bit unsigned data
INT32	long	32-bit signed data
UINT32	unsigned long	32-bit unsigned data
PHA8	volatile unsigned char *	8-bit physical access
PHA16	volatile unsigned short *	16-bit physical access
PHA32	volatile unsigned long *	32-bit physical access
APIS_PATH	unsigned long	APIS physical path ID
APIS_HANDLE	void *	APIS physical path handle
APIS_WIDTH	int	APIS access size in bytes
IDCODE	union {	ID EEPROM contents
	struct {	
	short synccode,	Sync code (0x5346)
	modnum,	Module number (binary coded)
	revision,	revision number
	modchar,	Module characteristics
	res[4];	reserved
	char manstr[16];	manufacturer string
	} id;	ID data
	short reg[16];	raw data
	}	

AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

6.3. FUNCTION REFERENCE

This section contains the reference of the functions provided by the M321 APIS-based software library.

m321_open() — Open APIS path

Syntax: `int m321_open(APIS_PATH apis_pathid, M321_HANDLE *handle)`

Description: Opens a path to the module specified with its hardware address or ID.

Arguments: APIS_PATH apis_pathid
Hardware address or ID of the module
M321_HANDLE *handle
Pointer to m321 handle

Returns: handle.apis_path filled, if successful
handle.irqstat unchanged
0 (APIS_NOERR) when successful
or another APIS-error code when unsuccessful

Example:
```
result = m321_open(module_id, &handle);
```

m321_close() — Close APIS path

Syntax: `int m321_close(M321_HANDLE *handle)`

Description: Closes the APIS-path to the module.

Arguments: M321_HANDLE *handle
Pointer to m321 handle

Returns: 0 (APIS_NOERR) when successful
or another APIS-error code when unsuccessful

Example:
```
result = m321_close(&handle);
```

m321_boot() — Boot the m321

Syntax: `int m321_boot (M321_HANDLE *handle)`

Description: Boot the m321 from RAM or ROM (according to jumper configuration).

Arguments: M321_HANDLE *handle
Pointer to m321 handle

Returns: 0 when successful
-1 in case of hardware failure (timeout)

Example:
```
result = m321_boot(&handle);
```

m321_cmd() — Execute command

Syntax:
```
int m321_cmd (M321_HANDLE *handle, UINT16 command, void
*pArgList, int ni, int no)
```

Description: Copy command parameters to the parameter field in shared RAM of the m-module. Execute command and wait for the command to be completed. Copy the result parameters and return with the result code obtained from the firmware.

A table of available commands you can find in chapter 5.2

Arguments: M321_HANDLE *handle
Pointer to m321 handle
UINT16 command
firmware command
void *pArgList
Pointer to the parameter list
int ni
Number of input parameters
int no
Number of output parameters

Returns: Command result code from firmware
A table of possible result codes you can find on page 22

Example:
```
result = m321_cmd(&handle, SYNC, (UINT16 *)(&params), 0, 0);
```


AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

m321_clistat() — Set m321 interrupt status

Syntax: `int m321_clistat(M321_HANDLE *m321_handle, int mask)`

Description: Clear bits in the m321 interrupt status word.
A table of available interrupt resources and their position in the status word, you can find in chapter 5.4

Arguments: M321_HANDLE *m321_handle
Pointer to the m321 handle
int mask
Bits to clear

Returns: 0 when successful
APIS-error-code when unsuccessful

Example:
```
m321_setstat(&handle, IRQ_HOME_A);
```

m321_waitforirq() — Wait for an interrupt

Syntax: `int m321_waitforirq(void)`

Description: Wait for an APIS-originated interrupt or (non-APIS)_signal. (E.g. an APIS service routine is successfully handled or an external process signal is received)

Arguments: None

Returns: APIS_NOERR when there was an APIS interrupt/signal
APIS_ESIG when another signal was received (e.g. CTRL-C user interrupt)

Example: if (m321_waitforirq() != 0) break;

m321_gtistat() — Get m321 interrupt status

Syntax: `UINT16 m321_gtistat(M321_HANDLE *m321_handle, int mask)`

Description: Get bits of the m321 interrupt status word.
A table of available interrupt resources and their position in the status word, you can find in chapter 5.4

Arguments: M321_HANDLE *m321_handle
Pointer to m321 handle

Returns: interrupt status word

Example:
```
result = m321_gtistat(&handle);
```

m321_int_install() — Install interrupt routine

Syntax: `int m321_int_install(M321_HANDLE *m321_handle);`

Description: Installs the interrupt service routine for the specified m321.

Arguments: M321_HANDLE *m321_handle
Pointer to m321 handle

Returns: 0 (APIS_NOERR) when successful
or APIS-error code when unsuccessful

Example:
```
result = m321_int_install(&handle);
```

m321_int_deinstall() — De-install interrupt routine

Syntax: `int m321_int_deinstall(M321_HANDLE *m321_handle);`

Description: De-installs the interrupt service routine for the specified m321.

Arguments: M321_HANDLE *m321_handle
Pointer to m321 handle

Returns: 0 (APIS_NOERR) when successful
or APIS-error code when unsuccessful

Example:
```
result = m321_int_deinstall(&handle);
```


AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

6.4. SOFTWARE DISTRIBUTION

This section gives an overview of the software distribution.

File	Description
M321\SOFTWARE\m321rel.txt	Release notes
M321\SOFTWARE\LIB\m321lib.c	ANSI-C M321 APIS software library
M321\SOFTWARE\LIB\m321defs.h	Definitions for the M321 library
M321\SOFTWARE\LIB\m321firm.c	Downloadable firmware image for the M321
M321\SOFTWARE\EXAMPLE\m321demo.c	Demo program for M321
M321\SOFTWARE\EXAMPLE\makefile.bor	Borland C makefile for M321 on i2000
M321\SOFTWARE\EXAMPLE\makefile.os9	OS9 makefile for M321 on i4000
MMODID\SOFTWARE\LIB\modideep.c	ANSI-C ID EEPROM APIS software library
MMODID\SOFTWARE\LIB\ideeprom.h	Definitions for the ID EEPROM library

M321 example software is APIS based, therefore APIS support for the target platform is required for code generation.

The following figure is an example of the M321 software integrated in the APIS environment on an i4000/OS-9 target platform.

```
+--PROJECT                                          AcQ's dis-
                                                 tribution
     +---APIS                          APIS basis distribution
     |    +---DOC                      APIS documentation
     |    +---SOFTWARE
     |         | readme.txt            Distribution overview
     |         +---COMMON
     |         |    +---DEFS
     |         |    |    apis.h        General definitions
     |         |    +---OS9TRAP
     |         |    |   +--CMDS
     |         |    |       apistrap   OS-9 trap handler
     |         |    +---....
     |         +---I4000OS9            i4000/OS-9 support
     |         |    relnotes.txt       Release notes / version info
     |         |    apis_i4000os9.h    Platform specific definitions
     |         |    apis_i4000os9.c    Platform support library
     |         +---....
     +---M321                          M321 distribution
     |    +---DOC                      M321 documentation
     |    +---SOFTWARE
     |         | m321rel.txt           Release notes / version info
     |         +---LIB                 M321 support libraries
     |         +---EXAMPLE             M321 example software
     +---MMODID
          +---SOFTWARE
               +---LIB
                    modideep.c        M-module ID EEPROM library
```

Code generation is platform dependent, for information on building the software please refer to the release notes of the target platform and the APIS Programmer's Manual.

7. ANNEX

7.1. BIBLIOGRAPHY

Specification for M-module interface and physical dimensions:
M-module specification manual, April 1996, MUMM.
Simon-Schöffel-Strasse 21, D-90427 Nürnberg, Germany.

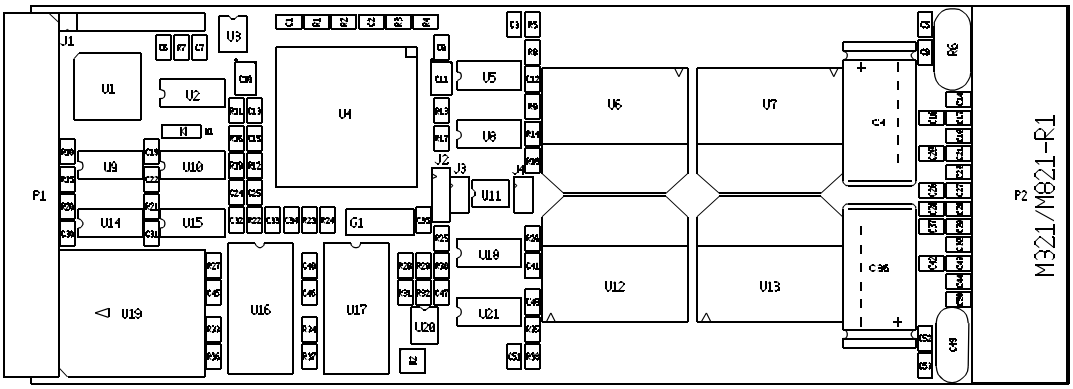
APIS Programmer's Manual
AcQuisition Technology
P.O. Box 627, 5340 AP Oss, The Netherlands.

Data Sheet of the NM93C46
Memory Databook 1992 edition (400069)
National Semiconductor Corporation
1111 West Bardin Road; Arlington, TX76017, United States

Data Sheet of the LMD18254 DMOS Full-Bridge Motor Driver
National Power ICs Databook
National Semiconductor Corporation
1111 West Bardin Road; Arlington, TX76017, United States

7.2. COMPONENT IMAGE

M321 Top View



7.3. TECHNICAL DATA

Slots on the base-board:
Requires one 16-bit M-module slot.

Connection:
To base-board via 40 pole M-module interface.
To peripheral via 25 pole D-sub connector.

Power supply:
+5VDC ±10%, typical 250mA.

Temperature range:
Operating: 0..+60°C.
Storage : -20..+70°C.

Humidity:
Class F, non-condensing.

EMC rules:
Emission: EN55022 Level B.
Immunity: EN50082-2.
Provided that for cabling, shielded multi-cable is used with metal DSUB-25 connector caps.
The shield must be coaxial terminated to the metal cap.

7.4. DOCUMENT HISTORY

! Version R1.0
First release

! Version R1.1
Nyquist industrial control references added
Software routine and filename references changed
Jumper orientation explained
Stepper motor algorithm described in anex

! Version R1.2
Firmware updated to R4.0
Commands ABORT_X and BREAK_X explained in more detail.
RESET bit in control register explained in more detail.
Adapted for firmware revision R4.0
IACK command added
Command interrupt service routine example changed
Interrupt service routine example provided
Result code PROFCORR removed
Description of motion profile changed (PROF_ABS_A command)
Motion profile restrictions removed
Maximum length of acceleration trajectory added
FLASHEN/FLASHCLR commands added
SWSYNC_X and GO_X commands added.

! Version R1.3
New layout
APIS support added

7.5. EXAMPLE CODE

This section contains a programming example written in ANSI-C and uses APIS and M321 functions provided by m321lib.c . The program configures the m321, after this it sets a breakpoint and starts a trajectory.

```
/*
 *  file:     m321demo.c
 *  revision: 1.1
 *  date:     15/02/00
 *  author:   TL
 * ------------------------------------------------------------------------
 *
 * M321 APIS based demo application.
 *
 * This application is part of the M321 ANSI-C library and demonstrates
 * the usage of a M321 or M322 on an APIS supported platform.
 *
 * This demo application uses APIS functions and M321 functions
 * provided by m321lib.c.
 *
 * ------------------------------------------------------------------------
 * Copyright 2000 by AcQuisition Technology B.V. (c)
 * All Rights Reserved
 * Reproduced Under License
 *
 * This source code is the proprietary confidential property of
 * AcQuisition Technology B.V., and is provided to the licensee
 * for documentation and educational purposes only. Reproduction,
 * publication, or any form of distribution to any party other than
 * the licensee is strictly prohibited.
 * ------------------------------------------------------------------------
 * Edition History
 *
 * #    date        Comments                                        by
 * ---  --------    ------------------------------------------  ---
 * 1.0  20-07-99    derived from ms21demo.c                         sp
 * 1.1  15-02-00    APIS structure update                           tl
 *
 */

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include "../LIB/m321defs.h"
#include "../../../APIS/SOFTWARE/COMMON/DEFS/apis.h"
#include "../../../MMODID/SOFTWARE/LIB/ideeprom.h"

/*
 *Globals
 */
volatile int term = 0;          /* program termination flag */

/*
 * Forward declarations
 */
void usage(char *);             /* display usage */
```


AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

```
/*
 * Function:    main
 *
 * Description: M321/APIS Demo program entry
 *
 * Parameters:  int argc
 *                  number of program arguments
 *              char *argv[]
 *                  pointer to program argument list
 *
 * Returns:     0
 */
int main (int argc, char *argv[])
{
    M321_HANDLE handle;     /* APIS Handle */
    UINT16 *args;           /* M321 argument list */
    UINT32 pathid = 0;      /* M321 base address or id */
    int i, j;               /* General idici */
    int result;             /* Command result */
    int repeat = 0;         /* Repeat flag */
    IDCODE idcode;          /* ID EEPROM structure */

    printf("\nM321/M322 Demo\n");
    printf("by AcQuisition Technology B.V. 1999\n\n");

    for (i = 1; i < argc; i++) {
        if (argv[i][0] == '-') {
            for (j = 1; argv[i][j]; j++) {
                switch (tolower(argv[i][j])) {
                    case 'b':
                        if (argv[i][++j] == '=')
                            j++;

                        sscanf(argv[i]+j, "%lx", &pathid);
                        while(argv[i][j])
                            j++;

                        j--;
                        break;
                    case 'r':
                        repeat = 1;
                        break;
                    default:
                        usage(argv[0]);
                        exit(1);
                }
            }
        }
    }

    /*
     * Allocate memory for the M321 command argument list
     */
    if ((args = (UINT16 *)malloc(32)) == 0) {
        printf("Not enough memory\n");
        exit(1);
    }

    /*
     * Open a hardware path to the M321
     */
    result = m321_open(pathid, &handle);
```

```
    if (result != 0) {
        printf("Could not open path: 0x%04x\n", result);
        exit(1);
    }

    /*
     * Read and verify contents of the Identification EEPROM
     */
    if (get_modid(handle.apis_handle, (void *)&idcode) != 0) {
        printf("ID EEPROM error\n");
        m321_close(&handle);
        exit(1);
    }

    if (idcode.id.synccode != IDSYNC) {
        printf("Invalid identification code\n");
        m321_close(&handle);
        exit(1);
    }

    printf("Module ID: M%d\n", idcode.id.modnum);
    if (idcode.id.modnum != 321 && idcode.id.modnum != 322) {
        printf("Error: wrong module\n");
        m321_close(&handle);
        exit(1);
    }

    /*
     * Booting the module
     */
    printf("Booting the module\n");
    if (m321_boot(&handle) != 0) {
        printf("Failed\n\n");
        m321_close(&handle);
        exit(1);
    }

    printf("Done\n");

    /*
     * Get firmware version
     */
    result = m321_cmd(&handle, VERSION, args, 0, 1);
    if (result != 0) {
        printf("VERSION command failed: 0x%04x\n", result);
        m321_close(&handle);
        exit(1);
    }

    printf("Firmware version: %d.%d\n", args[FWVER]/10, args[FWVER]%10);

    /*
     * Set up interrupts with default interrupt vector, level and mode
     */
    result = m321_int_install(&handle);
    if (result != 0) {
        printf("IRQ-install error %d\n", result);
        m321_close(&handle);
        exit(1);
    }

    /*
```


AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands

```
     * Unmask interrupts
     */
    args[IMASK] = (IRQ_TRAJ_A|IRQ_BRKPT_A);
    result = m321_cmd(&handle, MSKI, args, 1, 0);
    if (result != 0) {
        printf("MSKI command failed: 0x%04x\n", result);
        m321_close(&handle);
        exit(1);
    }

    /*
     * Clear interrupt status bits in m321istat
     */
    m321_clistat(&handle, (IRQ_TRAJ_A|IRQ_BRKPT_A));

    do {
        printf("Set Relative Breakpoint to +1000\n");
        args[BRKPT_H] = 0;
        args[BRKPT_L] = 0x3e8;
        result = m321_cmd(&handle, SET_BPR_A, args, 2, 0);
        if (result != 0) {
            printf("SET_BPR_A command failed: 0x%04x\n", result);
            term = 1;
        }

        printf("Moving +5000 steps\n\n");
        args[RISE_H] = 0;           /* rise frequency = 300 Hz */
        args[RISE_L] = 0x12c;
        args[DRIVE_H] = 0;          /* drive frequency = 1000 Hz */
        args[DRIVE_L] = 0x3e8;
        args[ACC_H] = 0;            /* acceleration = 10000 Hz/sec */
        args[ACC_L] = 0x2710;
        args[POS_H] = 0;            /* end position = 5000 steps */
        args[POS_L] = 0x1388;
        result = m321_cmd(&handle, PROF_REL_A, args, 8, 0);
        if (result != 0) {
            printf("PROF_REL_A command failed: 0x%04x\n", result);
            term = 1;
        }

        do {
            if (m321_waitforirq() != 0)
                term = 1;

            /*
             * Verify interrupt flags
             * Unmasked interrupt sources:
             *      IRQ_BRKPT_A
             *      IRQ_TRAJ_A
             */
            if (m321_gtistat(&handle) & IRQ_BRKPT_A) {
                printf("Breakpoint detected\n\n");
                m321_clistat(&handle, IRQ_BRKPT_A);
            }
        } while (!(m321_gtistat(&handle) & IRQ_TRAJ_A) && !term);

        if (m321_gtistat(&handle) & IRQ_TRAJ_A) {
            printf("Traject A completed\n\n");
            m321_clistat(&handle, IRQ_TRAJ_A);
        }
    } while (repeat && !term);
```

```
    /*
     * Mask all interrupts
     */
    args[IMASK] = IRQ_BLANK;
    result = m321_cmd(&handle, MSKI, args, 1, 0);
    if (result != 0) {
        printf("MSKI command failed: 0x%04x\n", result);
    }

    m321_close(&handle);         /* and close path */
    free(args);
    return 0;

}

/*
 * Function:    usage
 *
 * Description: The program usage is displayed and the program
 *              is terminated
 *
 * Parameters:  pointer to the program name
 *
 * Returns:     nothing
 */
void usage(char *pname)
{
    printf("Syntax: %s [<opts>]\n", pname);
    printf("Options (default):\n");
    printf("  -b=<base>     module APIS path ID in hex\n");
    printf("  -r            repeat\n\n");
}
```


AcQuisition Technology bv
P.O. Box 627, 5340 AP
Oss, The Netherlands