3 Functional Description

The module's functionality is based on the 68230 parallel interface timer. This component has a clock rate of 8MHz. Programming of the 68230 is very complex. Therefore this chapter only gives a general overview of the 68230 and a description of M11-specific functions. For a detailed functional description of the 68230, please refer to the data sheet Motorola.

3.1 The 68230

The 68230 is a versatile double-buffered parallel interface and a system-oriented timer. The parallel interface operates in unidirectional or bidirectional mode, and is either 8 or 16 bits wide. In unidirectional mode, a corresponding data direction register determines whether each port is an input or output. In bidirectional mode the data direction registers are ignored and the direction is determined dynamically by the state of four handshake lines. These four handshake lines provide an interface flexible enough for connection to a wide variety of low, medium, or high speed peripherals or other computer systems. The port section of the 68230 can generate vectored or autovectored interrupts. The M11 module does not support the DMA capabilities of the 68230.

The timer contains a 24-bit counter and a 5-bit prescaler. The timer may be clocked by the system clock (8MHz for the M11 module). It can generate periodic interrupts, a square wave, or a single interrupt after a programmed time period. It can also be used for time measurement or as a device watchdog.

Main Features

- 68000 bus compatible
- port modes include:
  - bit I/O
  - unidirectional 8-bit and 16-bit
  - bidirectional 8-bit and 16-bit
- programmable handshaking options
- 24-bit timer
- five separate mterrupt vectors
- separate port and timer interrupt service requests

The 68230 consists of two logically independent sections: the ports and the timer. The port section consists of port A (PA0..PA7), port B (PB0..PB7), four handshake lines (H1..H4), two general input/output pins, and six dual-function pins. The dual-function pins can individually operate as a third port (port C) or have an alternative function related to either port A, port B, or the timer. The four programmable handshake lines, depending on the mode, can control data transfer to and from the ports, or can be used as interrupt-generating inputs.

The timer consists of a 24-bit counter, optionally clocked by a 5-bit prescaler.

3.2 Interrupt Generation

The interrupt request line of the module is activated either by the timer or by changes in the level at the handshake lines. This interrupt must be serviced as a type A or type C interrupt as defined in the M-Module Standard. Type A interrupts are reset by accesses to the 68230 registers (c.f. component data sheet). With type C interrupts, an interrupt vector can be read at the module during the interrupt acknowledge cycle. The vector will be transferred through the base board, and the interrupt request at the module will be reset.

The 68230 has two interrupt request outputs. These are wired or gated. As for hardware, you cannot distinguish whether the interrupt has been triggered by the timer or by the handshake lines. If the module is programmed in a way that both interrupts are possible at the same time, you must trace the cause by software.

3.3 M-Module Identification

The M11 module is supplied with an identification EEPROM in accordance with the M-Module standard.

M-Module Identification (0xFE) (r/w)

15..8	7..3	2	1	0
-	-	CS	CLK	DATA

CS: 1 = chip-select for EEPROM (read)
CLK: serial identification clock (read)
DATA: identification data (read/write)

ANNEXE 6 : M11 DRIVER INTERFACE SYSTEM

1 Low-Level Driver Functionality

1.1 Général

The M11 is an M-Module with a 68230 component that implements binary input/outputs and a 24-bit timer. The MDIS4 driver supports the most important 68230 features.

1.2 Logical Channels

The M11 MDIS driver provides five logical channels that correspond to the I/O ports of the M11 :

Port A and B can be also combined to form a single 16-bit port.

Table 1: Logical Channels

Channel	Function
0	Port A (PA0..7)
1	Port B (PB 0..7)
2	Port C (PC 0..1)
3	Port H (H1..4)
4	Timer

1.3 Parallel I/O Ports

Ports A, B and C can be used for parallel I/O, while the port H pins can be defined as interrupt-capable inputs only.
The M11 MDIS driver does not support any of the buffered input/output modes of the 68230. All channels directly update or read the corresponding port pins. The port H lines (H 1 ..H4) are not used as handshake lines but operate independently.
Ports A and B can be operated in 2 modes:

- M11_2X8 mode (default): port A and B operate as two independant 8-bit ports
- M11_1X16 mode: port A and B form a 16-bit port

To change the mode to 1x16, call

M_setstat (path, M11_PORTCFG, M11_1X16) ;

Note: When the mode is changed, all port A/B pins are defined as inputs and all port H signals and interrupts are cleared.

1.3.1 Data Direction

The data direction of ports A, B and C can be defined independently for each pin using setstat call *M11_PINDIR*. After initialization, all port pins are defined as inputs.
For example, to define PA0 and PA1 as outputs, and all other port A pins as inputs call

M_setstat (path, M_MK_CH_CURRENT, 0) ;
M_setstat (path, M11_PINDIR, 0x3) ;

1.3.2 Modifying Outputs

The value of port A, B and C pins can be changed in three ways:

- *M_write(path, value)* updates all output pins of the selected port.
- *M_setstat(path, M11_SETPINS, value)* sets all pins whose bit is set in *value* and leaves all other bits unchanged.
- *M_setstat(path, M11_CLRPINS, value)* clears all pins whose bit is set in *value* and leaves all other bits unchanged.

1.3.3 Port A/B in 16-Bit Mode

When ports A and B are configured for 16-bit mode, the I/O for this combined port is done over channel 0. The *M_read/M_write* routines operate with a 16-bit value.
The data of ports A and B is presented in this 16-bit value as follows:

Bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Pin	PA7	PA6	PA5	PA4	PA3	PA2	PA1	PA0	PB7	PB6	PB5	PB4	PB3	PB2	PB1	PB0

Any write to channel 0 will simultaneously update the data on port A and B, while a read from channel 0 will take a snapshot of ports A/B pins at the same time.
Logical channel 1 does not exist in this mode. Any attempt to use channel 1 will result in error *ERR_LL_ILL_CHAN.*
Restrictions in 16-bit mode:

- All pins of port A/B in 16-bit mode must have the same data direction. To define port A/B as output, use *M_setstat(path, M11_PINDIR, 0xffff)* or to define it as input *M_setstat(path, M11_PINDIR, 0x0000)*, all other values are rejected.
- Port H pins H3 and H4 are not interrupt-capable when port A/B is configured for 16-bit mode.

1.3.4 Port H

Port H (channel 3) is made up of four pins, HI..H4. The driver supports these pins as inputs only, but each pin can generate an interrupt and may cause a signal to be sent to the application.
To read the state of these pins, just call *M_read(path, &value)*. Hl corresponds to bit 0, H4 to bit 3.

Each port H pin can generate an interrupt on the rising or falling edge. This can be configured through setstat code *M11_HXSENSE*. The default sensitive edge is the falling edge (i.e. transition from "high" to "low" level). To change the sensitive edge of HI and H2 to the rising edge call **M_setstat (path, M11_HXSENSE, 0x3) ;**
To change back to the falling edge call

M_setstat (path, M11_HXSENSE, 0x0) ;

To enable the interrupt and to install the signal that is sent on each sensitive edge for Hl and H2 , call

M_setstat (path, M11_SIGSET_H1, sigCode1) ;
M_setstat (path, M11_SIGSET_H2, sigCode2) ;

Note: Hx interrupts can only occur if the global interrupt of the M-Module bas been enabled via setstat *M_MK_IRQ_ENABLE* !

Each Hx pin can generate a different signal, but it is also possible for all pins to generate the same signal. A signal for a spécifie Hx pin can be installed by only one process.
To clear the signal for H2, call

M_setstat (path, M11_SIGCLR_H2, 0) ;

1.4 Timer

The M11 driver supports the timer unit of the 68230. This is a 24-bit down-counter with interrupt capability. The timer interface is compatible with the general "MDIS timer profile" that is described below.

1.4.1 The MDIS Timer Profile on M11

The Timer Profile implements a count-down counter that counts down from a preload value until zero is reached. This counter implementation can be used to produce one-shot timeouts or periodic interrupts (signals).

The Timer uses channel 4 on the M11.
An *M_write* call to the channel sets the initial timer value (preload) and restarts the timer if it is already running.
An *M_read* call to the device reads the current timer value. The read will not affect the timer state.
The resolution (number of decrements per second) can be queried by the application through getstat call *M_TMR_RESOLUTION*. On the M11 the timer resolution is fixed to 250,000 (4µs).
The number of counter bits must be returned by the driver using *M_LL_CH_LEN* ("24" for M11).
Getstat call *M_LL_CH_TYP* returns *M_CH_PROFILE_TMR* on channel 4.
The timer is started and stopped via setstat *M_TMR_RUN*. The *value* argument determines whether to start or stop the counter and the mode in which the timer is operating:

- *M_TMR_STOP* stops the timer. The timer will keep the current value.
- *M_TMR_START_ONE_SHOT* loads the preload value into the counter. The counter is decremented with each timer clock until the counter decrements to zero. When it reaches zero, the logical state of the timer is set to "stopped" (i.e. *M_TMR_RUN* getstat returns *M_TMR_STOP*). Internally the timer may still continue to run. *M_read* may return random values after the timer passed the zero mark.
- *M_TMR_START_FREE_RUNNING* loads the preload value into the counter. The counter is decremented with each timer clock until the counter decrements to zero. Then the preload counter is loaded again into the counter.

When *M_TMR_RUN* is called as a getstat code it returns information on whether the timer is started or stopped. The values are the same as for setstat call *M_TMR_RUN*.

The application can install a signal that is sent whenever the timer reaches zero (decrements from one to zero). *M_TMR_SIGSET_ZERO* installs the signal while *M_TMR_SIGCLR_ZERO* clears it.

Note: The timer interrupt can only occur if the global interrupt of the M-Module has been enabled via setstat *M_MK_IRQ_ENABLE* !

1.4.2 Using the Timer

Since the M11 driver implements an interface compatible with the "MDIS timer profile", a generic header file and example programs are used.
The header file for the timer profile is *m_tmr_drv.h*.
The example programs can be found in TOOLS/M_TMR.

1.5 IDPROM

Since older M11 M-Modules do not have an ID PROM, the validity check of the ID PROM is disabled by default. To enable this check, set the *ID_CHECK* key in the device descriptor to 1.
If an ID PROM is present, its contents can be read using getstat *M_LL_BLK_ID_DATA*.

2 Device-Specific SetStat/GetStat Codes

The M11 driver provides the following status codes in addition to the standard codes :

Table 2: Device-Specific SetStat and GetStat Codes

Code	Set/Get	Description
M11_PORTCFG	Set/Get	defines whether ports A and B operate separately or together
M11_PINDIR	Set/Get	pin-specific data direction
M11_SETPINS	Set	sets all pins whose bits are 1 in *value*, leaves other bits unmodified
M11_CLRPINS	Set	clears all pins whose bits are 1 in *value*, leaves other bits unmodified
M11_SIGSET_H1.. M11_SIGSET_H4	Set/Get	installs signals for H1..H4 interrupts and implicitely enables interrupts
M11_SIGCLR_H1.. M11_SIGCLR_H4-	Set	clears above signal
M11_HXSENSE	Set/Get	defines the sensitive edge for pins H1..H4
M_TMR_RESOLUTION	Get	gets number of decrements per second of timer's counter
M_TMR_RUN	Set/Get	starts/stops the timer or reads the current state
M_TMR_MODE	Set/Get	defines/reads timer mode
M_TMR_SIGSET_ZERO	Set/Get	installs signal that is sent when timer reaches zero
M_TMR_SIGCLR_ZERO	Set	removes above signal

3 Functional Interface

3.1 Low-Level Driver Interface

The low-level driver is seated directly on the hardware. You can use it to write your own device driver even without profound knowledge of the operating system.

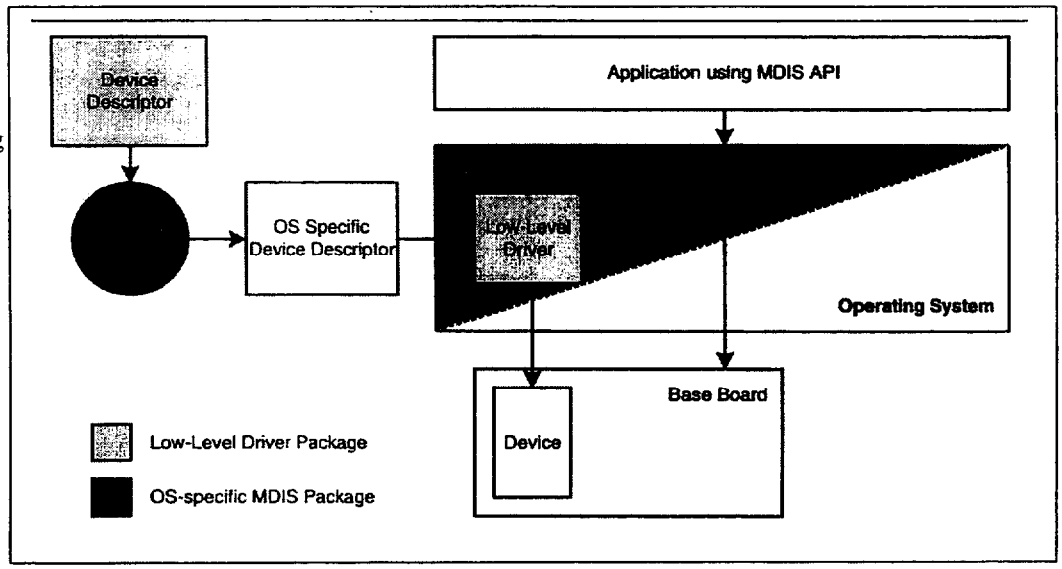
Function	Description
M11_Init	Allocate and return low-level handle, initialize hardware
M11_Exit	De-initialize hardware and clean up memory
M11_Read	Read a value from the device
M11_Write	Write a value to the device
M11_SetStat	Set the driver status
M11_GetStat	Get the driver status
M11_Irq	Interrupt service routine

Table 4 : Overview of Low-level Driver Functions

3.2 MDIS Application Programming Interface (API)

The MDIS API is a function interface for the user to access an MDIS driver.
This chapter only gives a correspondence table of MDIS API calls and low-level driver functions.

Figure 1: MDIS Application Programming Interface (API)



3.2.1 Supported API Functions

Table 3 : Correspondence between MDIS API Functions and Low-Level Driver Functions

API Function	Low-Level Driver Function
M_open open device `int32 M_open( char *device )`	***M11_Init*** Chapter 3.2
M_close close device `int32 M_close( int32 path )`	***M11_Exit*** Chapter 3.2
M_read read from device `int32 M_read( int32 path, int32 *value )`	***M11_Read*** Chapter 3.2
M_write write to device `int32 M_write( int32 path, int32 value )`	***M11_Write*** Chapter 3.2
M_setstat set device parameter `int32 M_setstat( int32 path, int32 code, int32 data )`	***M11_SetStat*** Chapter 3.2
M_getstat get device parameter `int32 M_getstat( int32 path, int32 code, int32 *data )`	***M11_GetStat*** Chapter 3.2
M_getblock block read from device `int32 M_getblock( int32 path, u_int8 *buffer, int32 length )`	not used
M_setblock block write to device `int32 M_setblock( int32 path, u_int8 *buffer, int32 length)`	not used
M_errstring generate error message `char* M_errstring( int32 errCode )`	-

Note on M_setstat and M_getstat
For normal status codes, *data* points to a 32-bit integer value. For block status codes, it is interpreted as a pointer to structure *M_SG_BLOCK*, which contains the size and location of the data buffer.
Structure M_SG_BLOCK

```
typedef struct {
        int32 size;   /* application buffer size */
        void *data;   /* application buffer location */
} M_SG_BLOCK;
```

3.2.2 Programming Example for MDIS API

The following C program is an example of how to use the MDIS API functions with the M11 M-Module. You can find this example program on the driver disk in subdirectory *M11_SIMP*.

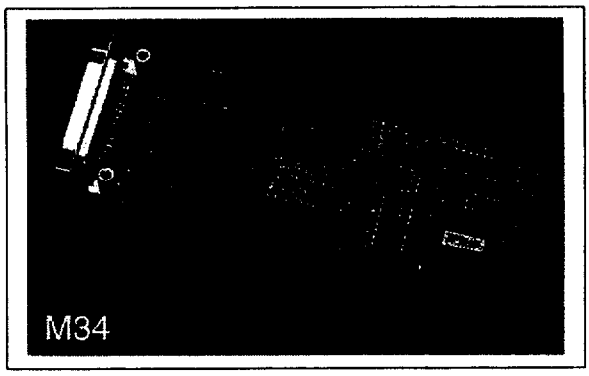
```
#include <stdio.h>
#include <string.h>
#include <MEN/men_typs .h>
#include <MEN/mdis_api. h>
#include <MEN/usr oss . h>
#include <MEN/m11.drv.h>

static void PrintError(char *info) ;

int main (int argc, char *argv[])
{
        int32   path, value ;
        char    *device ;

        if (argc < 2 || strcmp (argv [1] , "-?" )==0)  {
                printf (" Syntax :  m11_simp <device> <chan>\n") ;
                printf (" Function: M11 example for port input\n");
                printf ( "Options : \n" ) ;
                printf ("          device      device name\n");
                printf ("\n") ;
                return (1 ) ;
        }
        device = argv[l] ;
        /*---------------------------
        /       open path       /
        ----------------------------*/
        if ( (path = M_open (device) )  < 0)  {
                PrintError ("open") ;
                return(l) ;
        }
        /*---------------------------
        /       config          /
        ----------------------------*/
        /* channel number */
        if ((M_setstat(path, M_MK_CH_CURRENT, 0))  < 0)  (
                PrintError ("setstat M_MK_CH_CURRENT") ;
                goto abort ;
        }
        /* set port A to input */
        if ((M_setstat(path, M11_PINDIR, 0x00)) < 0) (
```

```
                PrintError ("setstat M11_PINDIR") ;
                goto abort ;
        }
        /* channel number */
        if ((M_setstat(path, M_MK_CH_CURRENT, l))  < 0)  {
                PrintError ("setstat M_MK_CH_CURRENT");
                goto abort ;
        }
        /* set port B to input */
        if ((M_setstat(path, M11_PINDIR, 0x00)) < 0)  {
                PrintError ("setstat M11_PINDIR");
                goto abort ;
        }
        /*---------------------------
        /       Read            /
        ---------------------------*/
        /* channel number */
        if ((M_setstat(path, M_MK_CH_CURRENT, 0)) < 0)  {
                PrintError ("setstat M_MK_CH_CURRENT");
                goto abort ;
        }
        if ( M_read( path, &value )){
                PrintError ("read Port A") ;
                goto abort ;
        }

        printf ( "Port A: 0x%02x\n" , value } ;

        /* channel number */
        if ((M_setstat(path, M_MK_CH_CURRENT, l)) < 0)  {
                PrintError ("setstat M_MK_CH_CURRENT");
                goto abort ;
        }
        if( M_read( path, &value )){
                PrintError ( " read Port B" ) ;
                goto abort ;
        }

        printf ("Port B: 0x%02x\n"  value );

        /*--------------------+
        / cleanup           /
        +--------------------*/
        abort :
        if (M_close (path) < 0)
                PrintError ("close") ;

        return (0) ;
}
static void PrintError (char *info)
{
        printf ( " *** can' t %s : %s\n",  info, M_errstring (UOS_ErrnoGet ( ))) ;
}
```

ANNEXE 7 : M-MODULE M34

The M34/M35 Modules

The M34 and M35 are fast analog input M-Modules with a resolution of 12 bits and 14 bits, respectively. The inputs are optically isolated from the system, which is essential for advanced automotive and industrial applications. For ease of use, the isolated supply voltages (±15V) can be generated on the board. For this purpose, however, a DC/DC converter is required.



A fast A/D converter combined with auto-incrementation of the multiplexer channel makes these modules ideal for high-speed sampling measurements. The complete acquisition time (i.e. the time between two measurements) is less than 8.5µs for the M34 (12 bits) and less than 10µs for M35 (14 bits). External triggering with interrupt generation is also possible. Depending on the access mode, end of conversion is gated with /DTACK. So it is never necessary to poll a status register and it is perfectly possible to have pipelining between measurement and storing of the results.

Features

A/D Conversion:

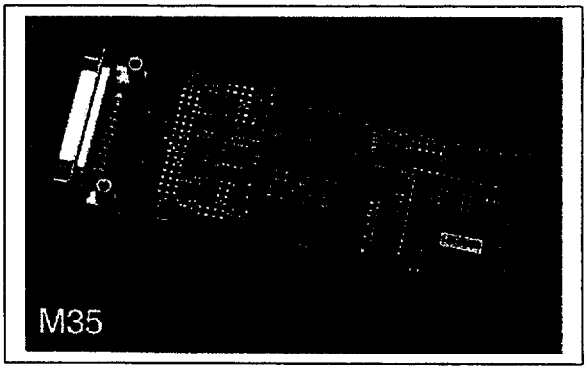
- 12 bits/8.5µs on the M34
- 14 bits/10µs on the M35
- precision: ±1 LSB, ±0,2% typ.
- optically isolated (500V isolation)
- programmable gain factor of 1, 2, 4 or 8 (factor 16 by hardware jumpering)
- software-selectable unipolar or bipolar opération
- sample and hold
- autoincrement of channel number

M-Module Compliance :

- A08, D16, INTA, IDENT

The modules themselves do not have an input signal conditioner and a multiplexer. These are provided by a small additional adapter PCB. This guarantees maximum flexibility for differrent applications. Four kinds of input conditioning are available as a standard. One version has eight-channel input, with as many as eight (!) differential amplifiers for -10V..+10V or -20mA..+20mA measurements. All channels have the same ground reference with a common mode range of -200..+200V (!) Another version is supplied with 16 single-ended inputs referred to ground with the same input ranges as the differential versions (but a different common mode range). Both the differential and the single-ended version have second-order active filtering for each channel.

The M34 is an improved software-compatible version of the M35.



Input Conditioning:

voltage or current inputs
- précision: ±0.5% typ.
- active Filters for each channel: 1kHz
- voltage or current input: impedance 100kOhm (voltage) or 62.5Ohm (current)
  - 16 analog inputs, grounded high input voltage tolerance cross-talk less than 56dB
  - 8 analog inputs, differential high common mode range ±200V cross-talk less than 60dB
- PT100 temperature acquisition

Miscellaneous:

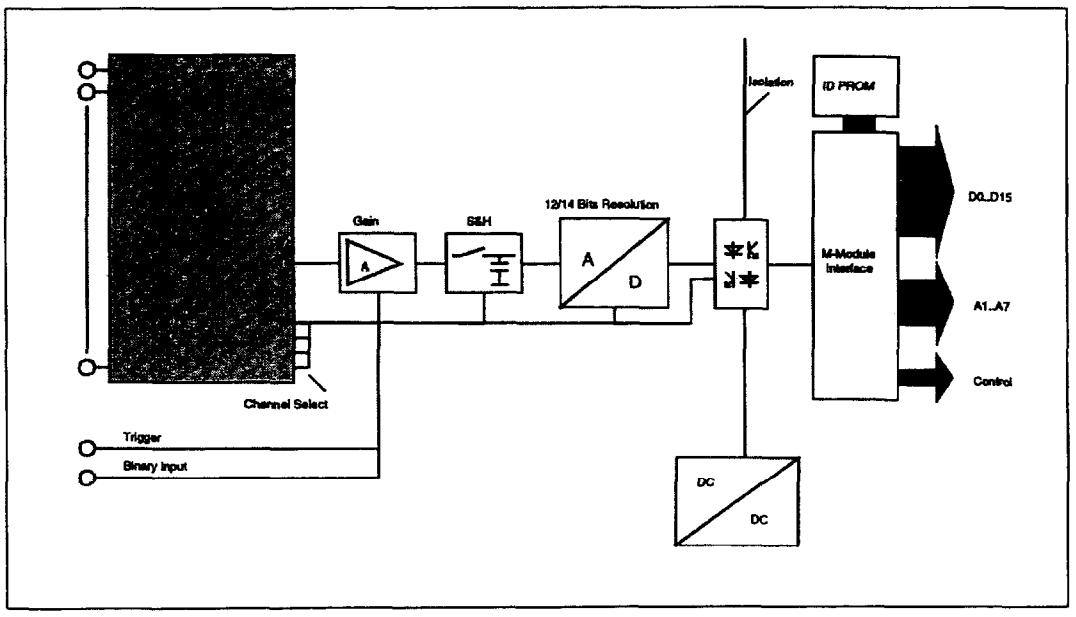
external trigger (isolated, rising-edge sensitive)

- extemal binary input
- ESD/EMC protection

Software Support :

- MDIS driver concept

Block Diagram



Ordering Information

04M034-00 M34, adapter base module for analog input configuration, 12 bits, ESD
04M034-20 M34, as 04M34-00 but with DC/DC converter

04M035-00 M35, adapter base module for analog input configuration, 14 bits, ESD
04M035-20 M35, as 04M34-00 but with DC/DC converter

20M034-00 M34/M35 hardware documentation

08AD01-01 adapter for 16 voltage inputs, grounded
08AD02-01 adapter for 8 voltage inputs, differential

08AD01-02 adapter for 16 current inputs, grounded
08AD02-02 adapter for 8 current inputs, differential

1 Installation and Setup

1.1 First Steps

To test proper functioning of the module you can proceed as follows:

- Install the system with the M-Module base board but without the M34/M35.
- Test the base board.
- If O.K., plug the module into slot 0 of the M-Module base board.
- The M-Module is completely trimmed on delivery.
- If you do not use a DC/DC converter, you must feed ±15V from an external source. Initially, do not change any jumpers or switches.
- Load a suitable debugger.
- Perform a read word access to the base address.
- If a bus error is occurring now, the module is not plugged properly.
- Write the word 0x0000 to the base address.
- Connect pin 13 and 14 to pin l at the D-Sub connector of the module.
- Reading from the base address plus 0x2 should yield the value 0x0000 or a similar value, e.g. 0x0001.
- Observe the installation manuals for operating system dependent software.

1.2 Connection of the Module

1.2.1 Power Supply

Power supply to the logic part is from the base board. The necessary voltage (only +5V) is supplied to the module from the base board. The isolated supply voltages can be generated on the module itself or may be taken from an external source (this depends on the version ordered). However, you should bear in mind that, because of the rather low efficiency of the DC/DC converters (about 60%), development of heat on the module rises. It is therefore essential to ensure adequate ventilation of the assembly. In addition, since a DC/DC converter is a switched power supply, it produces some noise. The M34/M35 is meticulously designed for maximum noise immunity. However, especially for the 14-bit M35, this might present a problem in some applications. If the optional DC/DC converters are not used, the isolated supply voltages (±15V, linear regulation recommended) can be supplied via the periphery connector, resulting in a considerable noise réduction and less development of heat on the module.

If the supply voltages are supplied from an external source, the DC/DC converter must be removed from the assembly (see configuration plan).

1.2.2 Peripherals

There are two possibilities for connecting peripherals:

- connection via 25-pin D-Sub connector or
- connection via the base board.

Pin assignment of the connectors depends on the type of input signal conditioning. There is no difference between M34 and M35 or between voltage and current inputs.

When a base board with a DIN 41612 PCB connection is used for peripheral signals (for example a 6U VMEbus base board), these are fed to the module through the 24-pin plug connector. Connection may be made using the 21-pin module connector (cf. base board manual). When these connectors are used, for each module three pins of the 96-pin DIN 41612 PCB connector cannot be used. The pin numbers for the DIN 41612 PCB connector shown below are valid for module slot number 3. If other module slots (2, l or 0) are used, the value 8, 16 or 24 must be added as appropriate.

The different input adapter PCBs used on the modules (8 channels differential, 16 channels single- ended etc.) result in different standard pin assignments. A detailed description of the available adapters - including the appropriate assignment tables - can be found in chapter 2.

2 Input Adapters

Active input filtering is implemented for each channel. Each channel has its own second-order lowpass filter. Gain is adjusted to an input range of ±10V and the filter frequency is fixed at 1kHz. The major components are implemented as normal (wired, not SMD) components, making it easy to produce customized input stages, which may also have individual gain factors and filter frequencies for each channel.

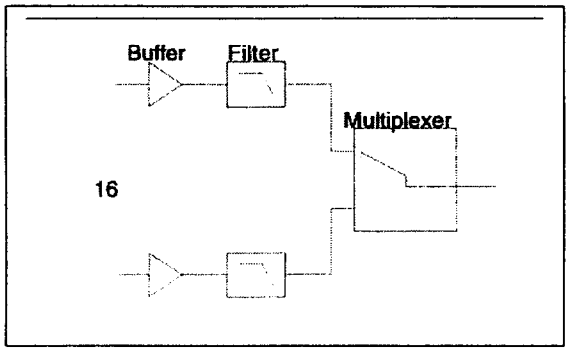
2.1 Notes on Pin Assignment

Pin TRIG can be used to trigger the start of external conversion. Channel and gain will not be changed. TRIG is active on the rising edge.
One bit of additional information can be output on pin BI. This pin does not trigger a function on the module. By connecting this pin to IGND in the connector, you can detect whether the connector on the module is plugged or not.
With external supply, only pins +15V, -15V and IGND must be used; +5V is not an input!

2.2 Single-Ended Acquisition (Current/Voltage)

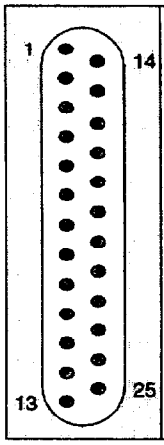
A standard input adapter – **AD01** - for 16 single-ended channels is available (see ordering information). The input voltage range is adjusted to ±10V for full scale and the second-order lowpass filter limits the input frequencies to lkHz. The current measurement version has a 62.5-Ohms shunt installed for full scale at ±20mA. The gain factor selected by software must be 8 !
For each individual channel, space is provided for mounting discrete components, such as resistors and trimmers, shunt resistors, voltage dividers or filter components. None of these components are SMD, in order to ensure a high degree of flexibility for special requirements.

Figure 1: Block Diagram of the AD01 Adapter



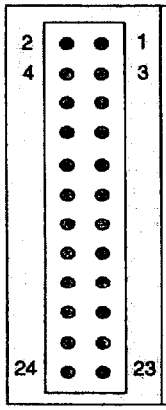
The following tables give the standard pin assignments for use of this adapter.

Table 1a: Pin Assignment for 16 Single-Ended Inputs (25-Pin D-Sub Connector)



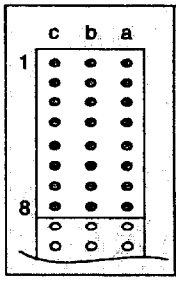
1	I0	14	I8
2	I1	15	I9
3	I2	16	I10
4	I3	17	I11
5	I4	18	I12
6	I5	19	I13
7	I6	20	I14
8	I7	21	I15
9	BI	22	TRIG
10	IGND	23	+15V
11	IGND	24	-15V
12	IGND	25	+5V
13	IGND		

Table 1b: 24-Pin Plug Connector



2	I8	1	TRIG
4	I9	3	I0
6	I10	5	I1
8	I11	7	I2
10	I12	9	I3
12	I13	11	I4
14	I14	13	I5
16	I15	15	I6
18	BI	17	I7
20	-15V	19	+15V
22	IGND	21	IGND
24	+5V	23	IGND

Table 1c: 24-Pin Header Connector



1c	I0	1b	I8	1a	TRIG
2c	I10	2b	I1	2a	I9
3c	I3	3b	I11	3a	I2
4c	I13	4b	I4	4a	I12
5c	I6	5b	I14	5a	I5
6c	BI	6b	I7	6a	I15
7c	IGND	7b	-15V	7a	+15V
8c	+5V	8b	IGND	8a	IGND

Table 2: Technical Data of AD01

Voltage measurement:	
Precision	±0.5%
Max. voltage	±15V
Voltage full scale	±10V
Input resistance	100kΩ, ±10%
Current measurement:	
Precision	±1%
Max. current	±25mA
Current full scale	±20mA
	U_A = ±1.25V
Load resistance	62.5Ω, ±0.1%

2.3 Differential Acquisition (Current/Voltage)

The second standard interface adapter -**AD02**- supports only 8 channels but has differential inputs. The input voltage and current rates are the same as for 16 channels (±10V or 20mA, filter frequency 3kHz). These rates may also be changed at the customer's request.

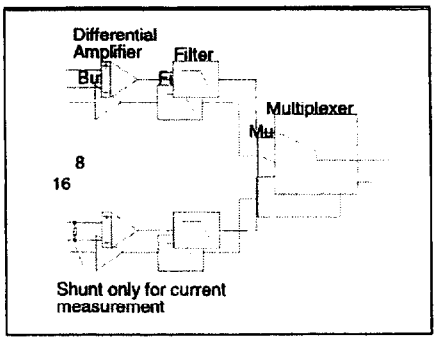
To make proper use of differential acquisition, it is necessary to have basic knowledge on analog measuring techniques. One common mistake is to believe that the two wires of a differential input acquisition system can be used like a hand-held instrument. This is not the case at all. A differential amplifier is an analog computation circuit. This circuit takes two input voltages referred to ground (!) and performs analog subtraction.

Unfortunately, in a normal multiplexed acquisition system each input signal has to pass through a multiplexer. This semiconductor part is only able to handle signals within the supply voltage range.

The problem is best illustrated by a simple example. Imagine you wish to measure the voltage and current of a 24V power supply. If you install a resistor divider between the 24V and ground to obtain a measuring range of 10V, you are not able to get the voltage across a shunt, because it is outside the working range of the multiplexer. Fortunately, the M-Module does not work in this conventional manner. The M-Module uses one differential amplifier for each channel. Of course, the amplifier also needs a reference ground for analog calculation but this amplifier has a common mode range of ±200V. Only the result of the calculation, which is normally between -10V and +10V, is switched by a multiplexer. This is of great practical advantage.

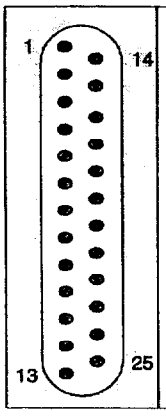
As for single-ended acquisition the current measurement version has a 62.5-Ohm shunt installed for full scale at ±20mA. The gain factor selected by software must be 8 !

Figure 2: Block Diagram of the AD02 Adapter



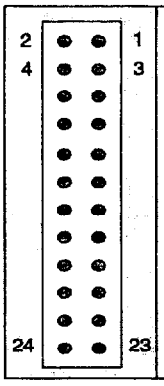
The following tables give the standard pin assignments for use of this adapter.

Table 3a: Pin Assignment for 8 Differential Inputs (25-Pin D-Sub Connector)



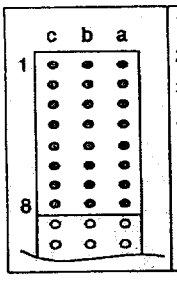
1	I0+	14	I0-
2	I1+	15	I1-
3	I2+	16	I2-
4	I3+	17	I3-
5	I4+	18	I4-
6	I5+	19	I5-
7	I6+	20	I6-
8	I7+	21	I7-
9	BI	22	TRIG
10	IGND	23	+15V
11	IGND	24	-15V
12	IGND	25	+5V
13	IGND		

Table 3b: 24-Pin Plug Connector



2	I0-	1	TRIG
4	I1-	3	I0+
6	I2-	5	I1+
8	I3-	7	I2+
10	I4-	9	I3+
12	I5-	11	I4+
14	I6-	13	I5+
16	I7-	15	I6+
18	BI	17	I7+
20	-15V	19	+15V
22	IGND	21	IGND
24	+5V	23	IGND

Table 3c: 24-Pin Header Connector



1c	I0+	1b	I0-	1a	TRIG
2c	I2-	2b	I1+	2a	I1-
3c	I3+	3b	I3-	3a	I2+
4c	I5-	4b	I4+	4a	I4-
5c	I6+	5b	I6-	5a	I5+
6c	BI	6b	I7+	6a	I7-
7c	IGND	7b	-15V	7a	+15V
8c	+5V	8b	IGND	8a	IGND

Table 4: Technical Data of AD02

Voltage measurement:	
Precision	±0.5%
Max. voltage	±200V
Voltage full scale	±10V
Input resistance	400kΩ typ.
Current measurement:	
Precision	±1%
Max. current	±25mA
Max. voltage to IGND	±200V
Input resistance	62.5Ω, ±0.1%

3 Address Organization

When using the driver software supplied, there is no need to be familiar with the hardware of the M34/M35 module in detail. However, familiarity with the address organization of the board is essential if you wish to write your own software for the module or do low-level development.

The 256-byte I/O area of the M34/M35 module is hardware-mapped. The address at which individual functions can be addressed from the base board is computed from the base address of the module plus the address in the following table.

Table 7: Address Map

Address	Read Access	Write Access
0x0	Read data register	Load channel, gain and operating mode
0x2	Read data register and start next conversion	Start a conversion and subsequently load the channel, gain and operating mode
0x6	Read data register and start next conversion, then increment channel number	-
0xa	Start a conversion and read data	-
0xe	Start a conversion and read data, then increment channel number	-
0xfe	M-Module identification	

4 Functional Description

4.1 Principle of Data Acquisition

The M34 and M35 are based on ANALOG DEVICES Inc.'s AD7870 and AD7872 chips. These are fast, complete A/D converters with a resolution of 12 bits (AD7870) and 14 bits (AD7872). Each component consists of a 2-µs track/hold amplifier, an 8-µs (AD7870) or 10µs (AD7872) successive approximation ADC, a 3-V buried Zener reference and versatile interface logic. The ADC features an integrated dock, which is laser-trimmed to guarantee accurate control of conversion time. No external dock timing components are required.

In addition to the traditional DC accuracy specifications such as linearity, full-scale and offset errors, these components are also fully specified for dynamic performance parameters including harmonic distortion and signal-to-noise ratio.

The components provide a serial data output format, which is used on the M-Modules to transfer data across optical isolation, saving components and power. Serial data transmission is bidirectional. When the A/D converter shifts out the digital information, the same signals transfer the configuration (channel number, gain etc.) from the digital controlling part to the optically isolated analog circuit.

The maximum input voltage level of the module is ±10V. Smaller values can be amplified by a factor of between 1 and 16 by means of the instrumentation preamplifier. The first stage can be programmed by software to factors 1, 2,4 or 8. A second stage can be programmed to 16 by hardware using an on-board jumper.

Channel selection logic supports a multiplexer input PCB. The control logic can be instructed to autoincrement the channel after each conversion.

Differing applications might require special adaptations of the module applying to the source of the measuring value. For this reason, input adaptation is not implemented on the module itself but on a small additional PCB plugged on the module.

4.2 Setting Gain, Mode and Channel

To load the gain, mode and channel number, it is necessary to write a word to a specific address. This is done by a write-only access. You should take care not to use a "Clear" command if you wish to reset all bits, since, with a CPU like the 68000, this command will first perform a read operation. If you use a high-level language you should also verify that your compiler did not generate instructions which read before writing.

Note: The configuration is not transmitted to the isolated part until a measurement has been performed.

Control Register (addr. 0x00 and 0x02)

15..8	7	6	5	4	3	2	1	0
RES	MOD	GN		INT	CH			

RES: reserved
MOD: measuring mode
0 = unipolar 1 = bipolar
GN: gain factor
00 = 1 01 = 2
10 = 4 11 = 8
INT: interrupt
0 = disable 1 = enable
CH: channel number