EMAC, inc.
EQUIPMENT MONITOR AND CONTROL

2390 EMAC, Way
Carbondale, IL 62902
Phone: (618) 529-4525
Fax: (618) 457-0110
http://www.emacinc.com

PCM-37E12
Users Manual

PCM-37E12 Users Manual rev1.1

Table of Contents

1. Introduction

This document describes EMACs PCM-37E12 multi-port I/O module. The PCM-37E12 is a PC/104 module that provides a wide variety of IO. Controlled by Motorola's powerful HCS12 processor this module provides maximum flexibility with ample processing speed for most control applications. The features of the PCM-37E12 are as follows:

FEATURES

- **I/O COPROCESSOR:** Motorola 68HCS12 16 Bit Processor running with a CPU clock speed of about 50 MHz.
- **DIGITAL I/O:** 8 General Purpose HCS12 Digital I/O lines, 8 Multi-Purpose I/O lines (GP/Counters/PWM), 8 Optically Isolated Digital Inputs, and 8 Optically Isolated Digital High-Drive Outputs.
- **COUNTER/PWM:** 8 Multi-Purpose HCS12 Digital I/O lines (GP/Counters/PWM).
- **ANALOG INPUTS:** 16 channels of 10 bit A/D 0 - 5 volts.
- **COMMUNICATION:** 1 RS232 Port and 1 RS232/422/485 port.
- **INTERFACES:** Character LCD interface with backlight support and a 24 key, keypad interface.
- **FORM FACTOR:** PC/104 Module with Dimensions of 96 mm x 90 mm (3.77" x 3.54").

OPTIONS

ON-BOARD OPTIONS

- **ANALOG:** Analog I/O Upgrade Includes:
    - 8 additional channels of 12 bit A/D, 0 - 5 Volts or 4 to 20 ma. input provision.
    - 4 additional channels of 12 bit D/A, 0 - 5 Volts.
- **CAN:** An Optically Isolated CAN 2.0 A, B Port.

OTHER OPTIONS

- **TERMINAL BOARD:** Screw Terminal Board allows for easy access to the PCM-37E12's I/O. Up to two Screw Terminal Boards can be stacked onto a single PCM-37E12.

A Linux device driver is provided with the board, along with a software demonstration source code for controlling the device. The software section of this document describes that software. More information on the software suite can be found in the software section of this manual, as well as in the source code itself, which is provided for most of the included applications.

2. Hardware

2.1 Specifications

- **VOLTAGE REQUIREMENTS:** 5 volt DC board input voltage.
- **CURRENT REQUIREMENTS:** 150 ma. @ 5 Volts Typical
- **OPERATING TEMPERATURE:** 0 - 70 degrees Centigrade, humidity range without condensation 0% to 90% RH.
- **DIGITAL I/O:** 8 programmable General Purpose TTL level I/O lines with an output drive capability of 10 ma. 8 Multipurpose TTL level I/O lines with an output drive capability of 10 ma. and a maximum total I/O drive of 50 ma. for these 16 lines. These lines can also be configured as Counters inputs and PWM outputs. 8 Optically Isolated dedicated Digital Inputs and 8 Optically Isolated dedicated Digital Outputs. The Isolated Outputs have open collector High-Drive outputs with 500 ma. sink drive capability and a maximum total I/O drive of 1500 ma. for these 8 lines. All Digital I/O lines terminate to standard 50 pin, I/O Rack compatible header connectors.
- **ANALOG INPUTS:** 8 analog inputs are multiplexed into a two 10-bit A/D converters with Sample & Hold for a total of 16 channels with a conversion time of 7 uSec. The analog input voltage range for each channel is 0 - 5 Volts. An optional 12-bit, 8 channel A/D is available bringing the analog input total to 24 channels.
- **ANALOG OUTPUTS:**. An optional 12-bit, 4 channel D/A is available. The analog output voltage range for each output is 0 - 5 Volts with a drive capability of 5 ma.

2.2 Jumpers

This section describes the jumpers of the PCM-37E12.

2.2.1 JB1

IRQ Select These jumpers connect the select the ISA legacy IRQ that the PCM-37E12 uses. This must be configured in software as well.

2.2.2 JB2

Base Addr. These jumpers select the base I/O address of the PCM-37E12. With no jumper attached the board is at I/O base 0x200. A single jumper may be added to increase the I/O base by the number specified on the silkscreen. Only one jumper should be added at a time. This must be configured in software as well.

Example: adding a jumper to the 20 position will locate the board at I/O address 0x220

IMPORTANT! Rev 0 boards have an incorrect silkscreen on the jumper. The +100h silkscreen should read +200h.

2.2.3 JP1

LCD Config. LCD configuration jumpers. These Jumpers allow for different types of LCDs. Not currently used.

2.2.4 JP2

CAN Term. Place a jumper in the T position to terminate the CAN bus.

2.2.5 JP3

Pull-Up This jumper selects the pull-up voltage for the isolated digital outputs. Setting it to the 5V position sets the output pull-up voltage to 5 VDC and setting it to the USR position allows the user to feed the desired pull-up voltage (0 – 24 VDC) to the board.

2.3 JTAG & BDM

This multipurpose header provides a programming interface to the PLD which translates the bus signals as well as a debugging interface directly to the Coprocessor. Currently it is only used internally be EMAC.

Table 1: JTAG & BDM Interface (HDR10)

Pin	Signal	Pin	Signal
1	JTAG_TCK	2	GND
3	JTAG_TDO	4	5V (Vcc)
5	JTAG_TMS	6	COP_RESET
7	NC/Reserved	8	BKGND
9	JTAG_TDI	10	GND

2.4 GP Digital I/O

These GPIO lines are exactly that, general purpose. They are connected directly to the coprocessor, so all communication must with them must be through the packet structure laid out in the software section of this manual. Besides being bit configurable I/O lines they can also be used as PWMs, and PT lines can be used as 16 bit counters. For software purposes PT is referred to as GP port 0, and PP is GP port 1.

Table 2: DIGITAL I/O Connector (HDR3)

Pin	Signal	Pin	Signal
1	PP7	2	GND
3	PP6	4	GND
5	PP5	6	GND
7	PP4	8	GND
9	PP3	10	GND
11	PP2	12	GND
13	PP1	14	GND
15	PP0	16	GND
17	PT7	18	GND
19	PT6	20	GND
21	PT5	22	GND
23	PT4	24	GND
25	PT3	26	GND
27	PT2	28	GND
29	PT1	30	GND
31	PT0	32	GND
33	NC	34	GND
35	NC	36	GND
37	NC	38	GND
39	NC	40	GND
41	NC	42	GND
43	NC	44	GND
45	NC	46	GND
47	NC	48	GND
49	5V(Vcc)	50	GND

2.5 Isolated Digital I/O

These Isolated input and output lines provide connections for heavier industrial relays and switches. They are optically connected to the PLD of the PCM-37E12, and do not require packet translation to use. The board isolation voltage is 250 VDC. The isolated inputs can nominally accept from 5 to 24 VDC. The outputs are open collector and can be pulled up to 24 VDC nominally and can withstand a maximum voltage of 50 VDC. Each output is capable of sinking 500 ma. of current, however the maximum of amount of current available is 1.5 amps for all eight outputs. Each output has a flyback diode, provided jumper JP3 is in place in either of its two positions.

In order to pull-up the outputs to 5 VDC place jumper JP3 in the 5V position. To pull up the outputs to any other voltage (up to 24 VDC) place the jumper in the USR position. If the jumper is placed in the USR position, then it's the user's responsibility to provide the desired pull-up voltage on pin 49 of connector HDR1. If no pull-up voltage is required then remove jumper JP3. Note that if jumper JP3 is removed then the outputs are no longer equipped with flyback diodes.

Table 3: ISOLATED DIGITAL I/O Connector (HDR1)

Pin	Signal	Pin	Signal
1	ISO_IN7	2	GND
3	ISO_IN6	4	GND
5	ISO_IN5	6	GND
7	ISO_IN4	8	GND
9	ISO_IN3	10	GND
11	ISO_IN2	12	GND
13	ISO_IN1	14	GND
15	ISO_IN0	16	GND
17	ISO_OUT0	18	GND
19	ISO_OUT1	20	GND
21	ISO_OUT2	22	GND
23	ISO_OUT3	24	GND
25	ISO_OUT4	26	GND
27	ISO_OUT5	28	GND
29	ISO_OUT6	30	GND
31	ISO_OUT7	32	GND
33	NC	34	GND
35	NC	36	GND
37	NC	38	GND
39	NC	40	GND
41	NC	42	GND
43	NC	44	GND
45	NC	46	GND
47	NC	48	GND
49	Pull-up Voltage	50	GND

2.6 Analog Channels

The HC12 coprocessor on the PCM-37E12 provides two independent 10 bit 8 port A/D modules. These modules provide lines ANI00 - ANI15. In addition to the coprocessors A/D, the 37e12 provides an optional 8 channels of 12 bit AtoD and/or 4 channels of 12 bit D/A. These optional channels are provided through external SPI devices. All of the Analog channels are controlled by the coprocessor and require packet translation.

Table 4: ANALOG (HDR2)

Pin	Signal	Pin	Signal
1	ANI00	2	ANI01
3	ANI02	4	ANI03
5	GND	6	GND
7	ANI04	8	ANI05
9	ANI06	10	ANI07
11	GND	12	GND
13	ANI08	14	ANI09
15	ANI10	16	ANI11
17	GND	18	GND
19	ANI12	20	ANI13
21	ANI14	22	ANI15
23	GND	24	GND
25	ANI16	26	ANI17
27	ANI18	28	ANI19
29	GND	30	GND
31	ANI20	32	ANI21
33	ANI22	34	ANI23.
35	GND	36	GND
37	ANO00	38	ANO01
39	ANO02	40	ANO03

2.7 RS232 UART

The PCM-37E12 provides one RS232 UART which be configured for baud rates between 9600 and 56K.

Table 5: 232 (HDR8)

Pin	Signal	DB9 Description
1	NC	-
2	NC	RxD
3	RxD	TxD
4	RTS	-
5	TxD	GND
6	CTS	-
7	NC	RTS
8	NC	CTS
9	GND	-
10	NC	

2.8 RS422/485 UART

The PCM-37E12 provides one RS422 UART which be configured for baud rates between 9600 and 56K.

Table 6: RS485 (HDR9)

Pin	Signal	DB9 Description
1	TX-	TX-
2	NC	TX+
3	TX+	RX+
4	NC	RX-
5	RX+	GND
6	NC	-
7	RX-	-
8	NC	-
9	GND	-
10	NC	

2.9 CAN I/O

The PCM-37E12 provides a single optically coupled CAN bus with 2 access connectors.

Table 7: CAN (HDR4)

Pin	Signal	DB9 Description
1	NC	-
2	NC	CANL
3	CANL	GND
4	CANH	-
5	GND	-
6	NC	-
7	NC	CANH
8	NC	-
9	NC	-
10	NC	

Table 8: CAN (HDR5)

Pin	Signal	DB9 Description
1	NC	-
2	NC	CANL
3	CANL	GND
4	CANH	-
5	GND	-
6	NC	-
7	NC	CANH
8	NC	-
9	NC	-
10	NC	

2.10 LCD

This header currently supports 2 and 4 line, character LCDs.

Table 9: LCD Interface (HDR6)

Pin	Signal	Pin	Signal
1	VCC	2	GND
3	RS	4	CNTR
5	E	6	R/W*
7	D1	8	D0
9	D3	10	D2
11	D5	12	D4
13	D7	14	D6
15	K	16	A

2.11 Keypad

This header provides an interface for a 4x4, 4x5, or 4x6 Keypad, which can send asynchronous data.

Table 10: KEYPAD (HDR7)

Pin	Signal
1	COL6
2	COL5
3	COL4
4	COL3
5	COL2
6	COL1
7	ROW1
8	ROW2
9	ROW 3
10	ROW 4
11	ESD SHIELD

3. Software

3.1 Introduction

This document describes the PCM-37E12 software package, and provides some general programming guidelines for using the PCM-37E12. It's organized into four main sections, The PCM-37E12 Driver, the Cop_Data structure, outgoing packet functions, and applications.

The driver provides the low level interface for writing to the hardware, no packet parsing or data interpretation is done at this level. It is used for reading raw data, writing to the isolated DIOs, and device configuration. Many of the driver calls have been hidden from the user in wrapper functions, and in future revisions, this abstraction will increase.

The Cop_Data structure is a data structure containing parsed data from the HC12 processor on the PCM-37E12. A parser function is available to translate the packets into the structures elements. Any application that reads from the processor of the PCM-37E12 will need to use one of these structures.

Outgoing packet functions are wrapper functions that encapsulate commands into packets and call the driver write function to transmit them. These commands make direct calls to EMAC's proprietary shared packet library.

Finally, the applications sections details the included applications provided by EMAC, and describes how users can create their own programs.

3.2 The PCM-37E12 Driver

The core functionality of the PCM-37E12 software package revolves around a Linux character driver called 37e12.o.

The driver can be loaded directly with an insmod command ie. insmod ./37e12.o.If you have purchased an integrated EMAC system, the driver will be pre-loaded by the operating system.

When the driver is loaded it dynamically registers a major number in the Linux system. Typically this number will be 254 if no other driver is using it. Nodes using this number will address the 37e12 driver. Integrated EMAC systems are pre-loaded with the node /dev/37e12 c 254 0. The zero is the minor number of the node, which is how the driver distinguishes between stacked PCM-37E12 boards. Minor numbers 0-3 are possible.

Device configuration is done through the IOCTL command, which is detailed in the next section. If your using an EMAC integrated system this will automatically be handled by a bootup script. Device information, IRQ, IOport, etc, is remembered by the driver until it is unloaded.

The PCM-37E12 driver uses the standard Linux character driver API. This subsections below describe how the API calls effect a 37e12 device. Driver functions not listed in the next section are not currently implemented on the 37e12 driver.

3.2.1 Open

Opens a PCM-37E12 device as blocking or nonblocking. A file descriptor is returned which should be used for all subsequent accesses. An open device should be closed before it's opened again. See man 2 open.

3.2.2 Close

Closes a PCM-37E12 device and releases it to be opened again. This will not destroy any of the devices configuration information. See man 2 close.

3.2.3 Read

Reads data from the PCM-37E12 Coprocessor. If the device is opened as non-blocking, this returns immediately with as much data is available. If opened in blocking mode this will sleep until a burst of data is returned from the PCM-37E12, this typically means 1 or more complete packets. Data will have to be interpreted by a Cop_Data object before any meaning can be extrapolated from it. Read will typically be called by the user in blocking mode to wait for data, or in non-blocking mode in response to a Fasync signal. See man 2 read.

3.2.4 Write

Writes data the PCM-37E12 Coprocessor. This command will probably never need to be used directly, as it is wrapped within the e12packetout command. See man 2 write.

3.2.5 IOCTL

The IO control function of the PCM-37E12 is used for 2 things. To configure the driver, and to directly control it's isolated digital IO ports, which are not connected to the HC12 processor. It accepts a cmd argument and an optional 3rd argument. Cmd is used to specify whether a config or digital function is requested.The third argument is used to pass a pointer, to the data in question, to the device. See ISO_in.c, ISO_out.c, and devconf.c for examples of this. Also take a look at the header file E12user.h for the configuration structure. See man 2 IOCTL.

3.2.6 Fasync

This function can be used to provide asynchronous signals to a user space application when new data arrives. Fcntl can be used to set the driver into asychronous mode. Once the driver is in this mode a SIGIO signal will be delivered when a new data burst is pending from the coprocessor. The PCM-37E12 driver uses the POLL_IN band to transmit the signal. See man sigaction for documentation on installing a signal handler. See man 2 fcntl.

3.3 Writing Software

Most of the functionality of the PCM-37e12 is implemented within it's internal HCS12 processor. This processor is pre-programmed with firmware for communicating over the PC-104 bus using a proprietary packet protocol. The firmware is serially updateable and upgrades may become available from time to time to add new functionality to the board. The version and compile __DATE__ of this firmware can be obtained using the version program (described in applications).

To facilitate development EMAC provides a Kdevelop project containing several example applications for communicating with the 37e12 as well as source to coprocessor.c, an object file which provides wrappers to the packet library for writing and interpreting packets to/from the PCM-37e12. This project also provides an API-DOC that goes more in depth into programming than this document. (Help->Project API-DOC from the Kdevelop menu after opening the PCM-37e12 project.)

An abbreviated description of coproccesor.c is detailed below.

When new data is read from the HC12 processor it's in a raw packetized state. To interpret the data a Cop_Data structure is used, which remembers the state of incoming data and maintains a list of handler functions if they are needed. The basic idea is to create a coprocessor structure with CopDeviceCreate and pass any read data to it with E12_packet_route. E12_packet_route then calls handler functions as the data is parsed, and when it's done returns a bitmask of all the packets it has completed. Returned data is stored in the Cop_Data structure. See coprocessor.h for details.

3.3.1 CopDeviceCreate

```
Cop_Data *CopDeviceCreate(int flags)
```

Creates and initializes a new Cop_Data structure which should be used for all subsequent packet routing. This device stores packet state information, so if a packet is only partially received before it is parsed the object will still be able to interpret it if the next incoming data finishes out the packet.

Parameters

`int flags` - Parser options, unused in this software rev
`Cop_Data *return value` - pointer to a new coprocessor data structure

3.3.2 E12_Packet_Route

```
__u32 E12_Packet_Route(Cop_Data *Cop, unsigned char *data, int datasize)
```

Routes a packet to the appropriate handler. Packet information is stored in the Cop_Data structure.

Parameters

`Cop_Data *Cop` - the coprocessor data structure
`unsigned char *data` - the data
`int datasize` - the size of the data being passed to the router
`__u32 return value` - bitmask of packet types completed in the data

These packet types can be identified by using the BITMASK macro in conjunction with the headers provided by packet.h.

3.3.3 CopDeviceDestroy

```
int CopDeviceDestroy(Cop_Data *Cop)
```

Destroys a Cop_Data structure. Currently this just frees the structure.

Parameters

`Cop_Data *Cop` - the device to destroy
`int return value` - returns zero on success

3.4 Outgoing Packet Functions

This section describes the functions used to transmit data to the HC12. These take the form of commands and data requests. One of the arguments to all of these functions is fd, the file descriptor.

The fd should be obtained from an open call to a PCM-37E12 device. If your using an EMAC integrated system this will be /dev/37e12.

3.4.1 e12packetout

```
int e12packetout(int fd,__u8 device,__u8 maskcnt,void *data)
```

A generic packet transmitting function. This function is called internally by all subsequent outgoing communication functions.

3.4.2 E12_PWMControl

```
int E12_PWMControl(int fd,int pwm, int frequency, int duty)
```

Configures a GP port as a PWM. Any of GP port 0 and 1's pins can be used as a PWM.

Parameters

int fd - The fd of the opened PCM-37E12 device to address.
int pwm - The pwm to configure
int frequency - The frequency
int duty - The duty cycle, 0 to 100

3.4.3 E12_CountControl

```
int E12_CountControl(int fd, int counter, int trigger, int threshold,int flags)
```

Configures a GP port as a counter. Any of GP port 0's pins can be used as a 16 bit counter. This function has not yet been fully tested.

Parameters

int fd - The fd of the opened PCM-37E12 device to address.
int counter - The counter to configure
int trigger - rising falling or both, see counter.h
int threshold - counter reload value
int flags - autostop, notify, autoreload, see counter.h

3.4.4 E12_Digitalout

```
int E12_Digitalout(int fd, __u8 data,int port)
```

Writes data to GP port, port.

Parameters

int fd - The fd of the opened PCM-37E12 device to address.
__u8 data - byte to write to the port
int port - GP port addressed, 0 or 1

3.4.5 E12_Digital_Req

```
int E12_Digital_Req(int fd)
```

Requests data from both GP digital ports. Digital data responses return both ports within the same packet in relatively the same time, so for simplicity both ports are always requested.

Parameters

int fd - The fd of the opened PCM-37E12 device to address.

3.4.6 E12_Digital_Cfg

int E12_Digital_Cfg(int fd, __u8 mask, int port)

Configures the GP ports to be inputs or outputs on a bitwise level.

Parameters

int fd - The fd of the opened PCM-37E12 device to address.
__ 8 mask - The new port mask. 1 for output 0 for input
int port - GP port addressed, 0 or 1

3.5 Applications

This section describes the predesigned applications provided for the PCM-37E12. At the time this document was written 5 application programs have been designed which use the PCM-37E12 software package. These programs provide basic Digital IO functionality and are also meant to be a example of programming with the PCM-37E12. For this reason source has been included.

The Coprocessor src is also included, which allows the user to see some of the inner workings of the parser. The Coprocessor.o object file will have to be linked into any new executable making use of the packet parsing functions. The protocol layer is hidden within the libe12packet.so.1.0.0 shared library, and must be referenced in the Makefile if an application makes use of the Coprocessor.c code. I promise it sounds more complicated than it actually is. See the Makefile in the applications directory for an example of how to do this.

Note: Most of these programs will generate a help screen if used incorrectly or used with no arguments.

3.5.1 Coprocessor.c

This is not really an application, but it contains code common to all the applications that use packet parsing. It contains wrappers which call the driver and the packet library.

3.5.2 ISO_in

```
ISO_in device
```

This program reads from the byte wide isolated digital input port of the 37e12 by using an IOCTL call. Returns a Hex byte value. This addresses the Isolated digital input ports and works outside of the processor. It's makes calls directly to the driver.

Arguments

device - any PCM-37E12 device, preinstalled software packages will use /dev/37e12

example: ISO_in /dev/37e12

3.5. ISO_out

```
ISO_out device value
```

This program writes a byte to the isolated digital output port of the 37e12 by using an IOCTL call. This addresses the Isolated digital output ports and works outside of the processor. It's makes calls directly to the driver.

Arguments

device - any 37e12 device, preinstalled software packages will use /dev/37e12
value - the data to write data preceded by 0x will be interpreted as hexadecimal

example: ISO_out /dev/37e12 0xaa

3.5.4 version

```
version device
```

This program displays the version of the firmware on the HC12 processor as well as the date it was compiled. EMAC integrated systems will automatically run this at boot time right after loading and configuring the driver.

Arguments

device - any 37e12 device, preinstalled software packages will use /dev/37e12

example: version /dev/37e12

3.5.5 devconf

```
devconf device IRQ address
```

Devconf configures a 37e12 device to use a specific IRQ and address by using the IOCTL command. EMAC integrated systems will automatically run this at boot time right after loading the driver using an IRQ and address appropriate to the board. This can be changed by modifying the e12startup script in init.d.

Arguments

device - any 37e12 device, preinstalled software packages will use /dev/37e12
IRQ - any valid IRQ number
address - any valid address, addresses preceded by 0x will be interpreted as hexadecimal.

example: devconf /dev/37e12 9 0x220

3.5.6 e12digitalGP

```
e12digitalGP device port command mask
```

This program controls the 2 GPIO ports on the HC12. These ports are from the processor and as such use packet protocol to send/receive data.

Arguments

device - any 37e12 device, preinstalled software packages will use /dev/37e12
port - GPIO port 0 or 1
command - this can be w,r, or i for write,read or I/O mask respectively
mask - for w this is the data to write, for i it's the mask, and doesn't apply for read.

examples: e12digitalGP /dev/37e12 0 i 0xaa
e12digitalGP /dev/37e12 0 w 0xff
e12digitalGP /dev/37e12 0 r

The above example would return aa if the input ports were floating at zero, because the pin state is always returned by a read, even for output pins.

4. More Information

We at EMAC are committed to the constant improvement of our products and technologies and would like to hear from you. If you are have questions or suggestions regarding your PCM-37E12, or would like to contribute software to the project, please email us at mailto: support@emacinc.com.