The contents of this manual and the specifications herein may change without notice.

TRADEMARKS

Hitachi is a registered trademark of Hitachi America, Ltd.

PC SmartLINK® is a trademark of Octagon Systems Corporation.

GNU is a copyright of Free Software Foundation, Inc.

Remote Processing Corporation
7975 E. Harvard Ave.
Denver, Co 80231 USA
Tel: (303) 690 - 1588
Fax: (303) 690 - 1875
internet: www.rp3.com

NOTICE TO USER

The information contained in this manual is believed to be correct. However, Remote Processing assumes no responsibility for any of the circuits described herein, conveys no license under any patent or other right, and make no representations that the circuits are free from patent infringement. Remote Processing makes no representation or warranty that such applications will be suitable for the use specified without further testing or modification. The user must make the final determination as to fitness for a particular use.

Remote Processing Corporation's general policy does not recommend the use of its products in life support applications where the failure or malfunction of a component may directly threaten life or injury. It is a Condition of Sale that the user of Remote Processing products in life support applications assumes all the risk of such use and indemnifies Remote Processing against all damages.

FCC NOTICE

The RPC-400 was not tested for EMI radiation. When operated outside a suitable enclosure, the board and any cables coming from the board will radiate harmful signals which interfere with consumer and industrial radio frequencies. It is your responsibility to properly shield the RPC-400 and cables coming from it to prevent such interference.

P/N 1683
Revision: 1.2

TABLE OF CONTENTS

TABLE OF CONTENTS

DESCRIPTION

The RPC-400 is a RISC based embedded controller. It is programmable in both assembly and C. Notable features include:

- High speed multimode counters for quadrature encoder, pulse train, and pulse width measurement.
- Twenty four analog inputs. 8 channels @ 10-bit on all RPC-400 configurations. 12- or 16- bit, 16 channels with software programmable gains from X1 to X800. Differential, single ended inputs with unipolar and bipolar ranges.
- Four RS-232 serial ports, one of which is isolated and configurable for RS-422/485.
- LCD character and graphic display and keypad port for operator interface.
- Digital I/O lines. A total of 69 programmable for input, output, or timing. 24 connect directly to an opto rack, 8 are high current output.
- Watchdog timer resets card if program crashes.
- 32K, 128K, or 512K RAM battery backable to save process variables and other data when power is off.
- Up to 512K flash EPROM to save programs and data.
- Load and run a program on power up or reset.
- Programmable in C, C++, or assembly.

The RPC-400 uses a Hitachi SH-1 RISC processor operating at 19.66 Mhz. Typical CPU instruction executes in 53 nS, including 32 bit multiply and divide. Its 6.00" x 10.0" size with 5 mounting holes make it easy to securely mount in a NEMA box. Compactness is enhanced by on-board terminal strips.

C programming language is included in this development system. It is derived from the GNU series of compilers.

Program development takes place on your PC, using your word processor. Programs are then cross compiled on a PC and are downloaded using a serial communication program.

MANUAL ORGANIZATION

This manual provides information to install and operate the RPC-400. Accompanying manuals, such as the compiler and SH-1 hardware, are used as references to specific questions.

Section 2 sets up your PC and runs a program on the RPC-400. Sections following address hardware features of the card.

This manual shows you how to interface the RPC-400 to your PC to compile and download programs and operate all hardware features of the card.

You should be familiar with C programming. If you are not experienced with any C software, you may want to refer to books available through your local book store or college.

MANUAL CONVENTIONS

Information appearing on your screen is shown in a different type.

Example:

```
CMON Copyright Hitachi Micro Systems
Enhancements by Paragon Systems
Additional enhancements by Remote Processing
```

Symbols and Terminology

NOTE: Text under this heading is helpful information. It is intended to act as a reminder of some operation or interaction with another device that may not be obvious.

WARNING:

Information under this heading warns you of situations which might cause catastrophic or irreversible damage.

W[-] Denotes jumper block pins.

< xxx>

Paired angle brackets are used to indicate a specific key on your keyboard. For example < esc> means the escape key.

nS Denotes nano seconds (1/1,000,000).

mS Denotes milli seconds (1/1000).

mA Denoes milli amperes (1/1000).

Numeric notations are in decimal unless preceded by conventional C '0x' notation.

CONNECTOR CONVENTION

IDC connectors are pinned out as shown below. The square pad, visible on the circuit side, is pin 1.

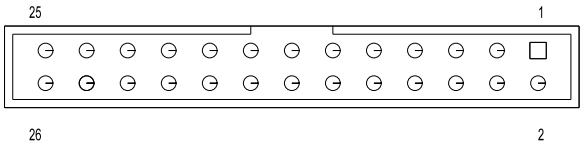


Figure 1-1 IDC connector viewed from top.

Ten and 20 pin connectors are similarly numbered.

TECHNICAL SUPPORT

If you have a question about the RPC-400 and can't find it in the manuals, call us and ask for technical support. Technical support hours are 9 AM to 4 PM mountain time.

When you call, please have your RPC-400 manual ready. Some times it is helpful to know what the card is used for, so please be ready to describe its application as well as the problem.

Phone: 303-690-1588
FAX: 303-690-1875
email: info@remotep.com

Figure 1-2 System layout

INTRODUCTION

The RPC-400 is ready to program as soon as you connect it to a PC and apply power. This section describes the steps needed to get a sign on message and begin programming.

PC requirements to load the compiler and set up your operating environment are discussed. A "Where to go from here" section tells you what sections to refer to in order to use the various capabilities of the RPC-400.

OPERATING PRECAUTIONS

The RPC-400 is designed to handle a wide variety of temperature ranges at low power. These characteristics require using CMOS components. CMOS is static sensitive. To avoid damaging these components, observe the following precautions before handling the RPC-400.

1. Ground yourself before handling the RPC-400 or plugging in cables. Static electricity can easily arc through cables and to the card. Simply touching your PC before you touch the card can greatly reduce the amount of static.

2. Do not insert or remove components when power is applied.

EQUIPMENT

A development system provides all of the parts necessary to program and operate the card. You need to supply a 386 or faster PC with an RS-232 serial port.

Power Supply
The development system includes a power supply capable of operating the board. There are some limitations on this supply in certain situations.

4 - 20 mA Current loop output requires a minimum of + 13 volts to operate using a 500 ohm terminator resistor. The development power supply will not output enough voltage. A 250 ohm resistor will work as the terminating resistor.

If you want to use your own supply, make sure it meets the following requirements. Designations in single quotes are those marked at P1.

'5V' + 5, ±0.25 V @ 0.5A. Reset voltage is about 4.75 volts and below. The board may be operated at up to 5.5V without damage. Current rating is based on no loads at the digital ports and should be increased based on requirements. If you are using an opto rack, each module requires 15 mA. An LCD display with LED back light on requires .5 amps. The 10-bit A-D converter normally uses + 5V as a reference. This is desirable if readings are ratiometric or not critical. See Section 10, *ANALOG INPUT* under "10 BIT A-D CONVERTER" for more information.

'+ 15' should be + 9 to + 16 volts @ 50 to 150 mA. If the system uses 4-20 mA current loop or 0-10V D/A output, supply voltage must be a minimum of + 13V. Current output depends upon number of 4-20 mA or D/A outputs. RS-232 outputs at J8, J9, and J11 use this same supply as does the A-D converter circuitry.

'-15' supply is -8 to - 16 volts @ 50 mA. This supplies D/A outputs, A-D inputs, and RS-232 outputs.

'CLP' supplies the current loop power. Normally it is + 13 to + 36 volts @ 80 mA. This supply is necessary only if you are using 4-20 mA current loop output at P4. You can use + 15 power provided transmission distances are short and load resistance is 500 ohms or less.

'PWR' at P5. This is intended to supply either -21 volts for a graphics LCD display or additional + 5V for a vacuum florescent. Refer to the display section for more information.

Be careful when using some switching supplies. Some switchers require minimum loads that may not be met when connected to this card.

Supply rise time should be as quick as possible. The penalty using a slow supply is garbage output of the RS-232 ports. The CPU is not sending data as it is held in reset. However, as the + and - 15 volt supplies rise, the output changes and can give the appearance of sending data.

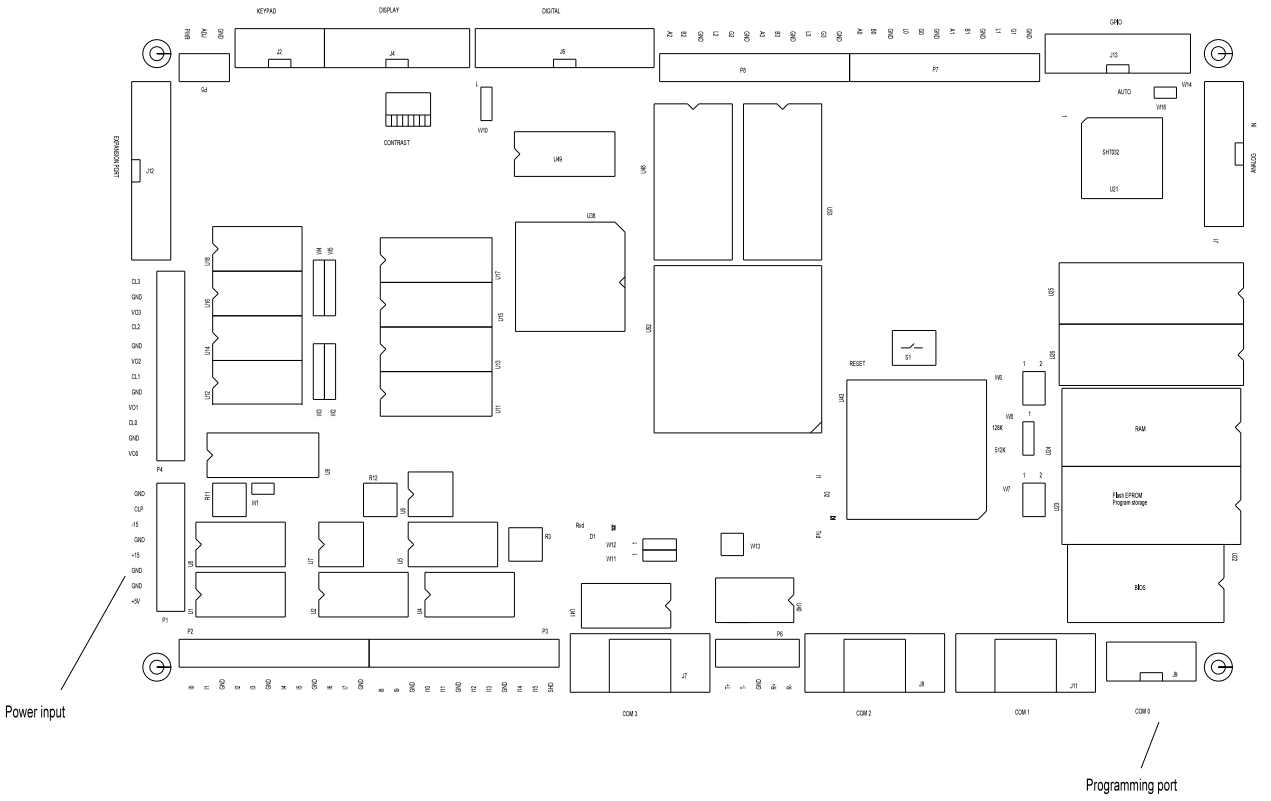


Figure 2-1 RPC-400 board power and programming ports.

Personal Computer (PC)
The PC used to compile and upload programs to the RPC-400 must be a 386 or "better". There are no real minimum speed requirements. The penalty for using a slow computer is increased compile and linking time.

One serial port is used to download, debug, and communicate with the RPC-400. Communication speed is set at 9600 baud.

The compiler, linker, and debugger use a DOS environment. DOS should be 3.3 or higher.

You may write - compile - link - and upload programs over a network. This has been successfully done using Microsoft (tm) Windows (R) version 3.11 and Windows 95 operating systems.

You should have about 10 Meg of free hard disk space available for saving programs.

PC SETUP-COMPILER INSTALLATION

The development system consists of a set of disks. These disks contain the compiler, libraries, source code, and demonstration programs. These are saved to various subdirectories in the computer. The root directory defaults to 'RPC-GNU' unless you supply an additional parameter in the DOS command line.

Loading the Disks
Make sure you are in the DOS root directory.

```
C:\
```

Insert disk 1 into the A: drive.

Type the following:

```
a:install
```

Program is installed to RPC-GNU directory.

The files will now unzip to your drive. Insert the remaining disks as prompted.

Modifying your Environment

The DOS path and environment must be modified before you can run many of the programs. Modify your AUTOEXEC.BAT file as follows:

Add to the path: C:\RPC-GNU\BIN

Set your environment as follows:

```
SET PATH=c:\devl\gnu\BIN;%PATH%
SET GCC_EXEC_PREFIX=c:\devl\gnu\LIB\
SET INFOPATH=c:\devl\gnu\INFO
SET C_INCLUDE_PATH=c:\devl\gnu\include
SET CPLUS_INCLUDE_PATH =
c:\devl\gnu\include\cxx;c:\devl\gnu\include
SET GO32=EMU c:\devl\gnu\BIN\EMU387
```

Set TMPDIR to point to a ramdisk or other temporary directory.

```
SET TMPDIR=c:\temp
```

You can also run the batch file SETENV.BAT and it will set up things for you.

Verifying Software Installation

To verify proper installation, set the current directory to \RPC-GNU\RPC400\DEMO and execute the batch file. For example:

```
c:>\RPC-GNU\RPC400\DEMO demo
gcc -O -x -G demo.c
ld -Trpc400.cmd
hint com1
```

This batch file invokes the compiler (gcc), linker (ld), and HINT terminal program. These programs should be in the RPC-GNU\BIN directory. If they are, then check your DOS path by typing in SET. One of the paths should point to \RPC-GNU\BIN. You will have to modify your AUTOEXEC.BAT file to do this.

Changing Batch Files

Batch files are used to edit, compile, link, and run the debugger program (GDB) or terminal program (HINT). The files assume you will be using DOS EDIT to make program changes. There is a good chance you will want to use your own editor, so make changes to the batch files as necessary.

OPERATING THE RPC-400

This part will power up the RPC-400, get a sign on message, and run the demonstration program.

Become familiar with the locations of connectors before getting started. See figure 2-1. Jumpers are set at the factory to operate the system immediately. For first time installation, do not install any connectors or parts until specified below.

1. Connect power.

   The RPC-400 needs + 5 and ±15 volts, as described above under "Power Supply". The development kit includes a power supply (which should not be plugged in to the AC outlet at this time). Connect the wires from the power supply connector to the board as follows:

PS connector wire color	P1 designator
Red	+ 5V
Black	GND
Orange	+ 15
White	-15

   Connect the cable from the power supply to the power supply connector just attached to the board. Don't plug in the power supply to an AC outlet just yet. If your power supply happens to have an extra black wire, connect it to ground.

2. Hook up to a PC.

   Connect the 10 pin side of the VTC-9F serial cable (P/N 1041) to J9 (also designated as COM 0). Connect the other end to COM1 or COM2 on your serial port.

   NOTE: Batch and command files assume COM1 is the default port. If you cannot use COM1, you must modify all .GSC and .BAT files and change references from COM1 to COM2.

3. Run HINT.
   If you ran the batch file under "Verify Software Installation" above and did not exit out of HINT, you might be ready. The batch program above assumes the RPC-400 is connected to COM1 on the PC.

   If your serial port is not COM2, type < Ctrl> -C to exit the terminal program HINT. Then type HINT COM2.

   If you did not run the above program, make sure you are in the \RPC-GNU\RPC400\DEMO directory.

   At the DOS prompt, type

   ```
   HINT COM1
   ```

   If you are connected to COM2, enter that port instead of COM1.

4. Power up.

   Turn on your power supply. On power up a message is printed.

```
C Monitor.  Hitachi Micro Systems, Inc. for the SH-1
Enhancements by Paragon Innovations, Inc,  [CMON Rev..
Additional enhancements by Remote Processing, Corp.
$S05#b8CMON>
```

The "$S05#b8" text is for the GDB debugger after a reset.

5. Testing.

   The card is now ready to load a program. Hit enter a few times to make sure the CMON> prompt comes back. Type in the following command line to load in the demo program:

   ```
   l:demo.sr
   ```

You will see progress of the program downloading to the card. Download takes about 2 minutes.

When downloading is done, you will see the low, high, and starting address. For example:

```
LOW   ADDRESS: 09002000
HIGH  ADDRESS: 09005747
START ADDRESS: 090025F4
```

Type in the letter 'g' and the start address you see on the screen. For example:

```
g 90025F4
```

The program will now execute. View the screen for further information. To stop execution, press the reset button.

RUNNING DEMO PROGRAMS

All demonstration programs are in the 'RPC400' directory. Each demo program uses either the GDB debugger or HINT communication program. Programs can be linked to use either debugging method.

All demo program directories have a batch file which calls the DOS editor to make changes to the program, compiles, links it and calls one of two programs to load the program into the board. Some demo programs use the GDB debugger while others use the HINT program. View the batch file to determine which one it uses.

Refer to *Section 16, PROGRAMMING NOTES, LINKING PROGRAMS* for information on changing the linker command file for debugging.

A common load format is used when HINT is used to load programs:

```
l:rpc400.sr
```

Enter this command after a board reset or at a command prompt while running HINT. The file 'rpc400.sr' is an S record file. This file name is specified in the linker command file.

It is a good idea to read the C source file before running a demo program. The first part will tell you what the program's purpose is and what to expect as an output. If your card does not have COM2 and COM3 serial ports, you cannot run any of the programs that use them.

WHERE TO GO FROM HERE

If you want to do this:	Turn to Section
Save a program	3
Run a program at power up or reset (Autorun)	3
Know more about serial ports	4
Adding and using data RAM	5
Using digital ports	6
Add a clock/calendar	7
Add battery backed RAM	7
Use display port	8
Connect a keypad	9
Measure voltages (analog input)	10
Enable the watchdog timer	11
Use external interrupts	12
Connect a proximity sensor	13
Connect a quadrature encoder	13
Measure pulse periods	13
Use the voltage outputs	14
Use the 4-20 mA outputs	14
Add other devices	15
How to start writing your own programs	16
Know more about the programming environment	16

INTRODUCTION

Temporary or final programs are stored in a flash type EPROM in socket U23. This socket holds 32K, 128K, or 512K byte devices. They can hold multiple programs or store data. Maximum running program transferrable to RAM is about 240K, excluding any variables.

Programs are saved to flash using the monitor program. The 'SF' command saves the start-up, or autorun program. It can also save secondary programs and data from RAM. An example flash EPROM saving and loading routine is in the EPROM1 directory.

Once a working program is properly saved to flash, it will auto run by removing jumper W16.

The IC in socket U23 is a 5-volt only Flash Programmable and Erasable Read Only Memory (PEROM). Any sector can typically be written to 10,000 times or more. A software locking mechanism prevents accidental modifications. Programs and data can be written over 10,000 times. This is over twice a day for 10 years.

Type	Size Bytes	W7 Configuration
29C256	32K	[3-5],[4-6]
29C010	128K	[3-5],[2-4]
29C040	512K	[1-3],[2-4]

To change the EPROM in U23, remove the IC and replace it with the new one. When installing a 29C256, pin 1 on the IC goes into socket pin 3. The top two rows of pins are empty.

SAVING A PROGRAM

Perform the following steps to save the start up program to flash. This program is automatically loaded to high speed RAM and run on power up or reset when jumper W16 is removed.

1. Load your program to RAM.
   You've already done this many times before while writing and debugging your program. The difference here is you have to use the HINT terminal program.

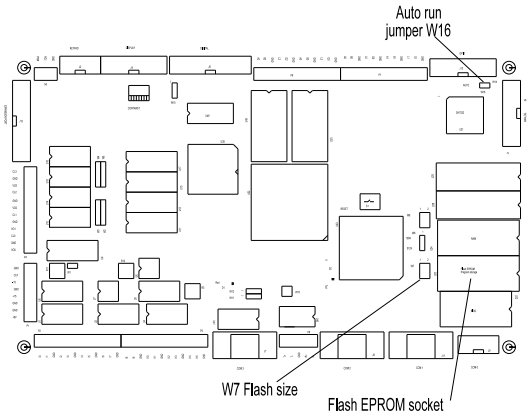


Figure 3-1 Saving Programs

CHANGING EPROM SIZE

The RPC-400 normally comes with a 32K or 512K flash EPROM. The size may be changed at any time. Set W7 according to the type/size.

If you have been using the GDB debugger, you will have to make minor changes to the link command file. The changes make the output from the linker into an 'S' format. Make sure the line (OUTPUT_FORMAT *fname.sr*) is in your .CMD link file. For an example, see the RPC400.CMD file in the AIN3 directory.

HINT instructions are in SECTION 2, page 3. Briefly, at the DOS prompt type:

```
HINT COMn
```

Where *n* is 1 or 2. Press the RPC-400 reset button to get the sign on message. Then type in:

```
l:fname.sr
```

fname.sr is your program. The program will download and the progress displayed on the screen.

2. Determine program size and start address.

   You will see the low, high and start address when downloading is done. To determine program size,

subtract the high from the low address. You may need to use the hex calculator in the monitor.

To use the monitor's hex calculator, use the 'P' command to print the value. For example, suppose the high address is 900E7F8. Low address is 9002000. Enter:

```
CMON> p 900e6e8 - 9002000
```

The result is printed back as:

```
decimal 50920   hex 0000C6E8
CMON>
```

Program length is 50920 bytes. Length is entered as a hex number and rounded up to C800.

Entered parameters are always in hex. Starting address is always 9002000, if you keep to the defaults. When you have determined the size, round it up to the next even page size. Example: Program size (in hex) is 6247, round it to 6400. This is, effectively, the number of bytes flash will use anyway because of programming requirements.

Starting address is displayed on the screen.

3. Save it to flash

   The SF command saves programs to flash. Type in the following command:

```
CMON> SF 9002000 length start_address
```

   Progress is displayed on the screen while programming. Programming time depends upon the length and flash EPROM type. A 512K type programs 512 bytes in about 10 ms.

Testing is as simple as removing jumper W16 and resetting the board. Your program should run.

AUTORUN

The RPC-400 is set to autorun on power up or reset by removing jumper W16. At power up or reset, the BIOS reads this jumper. If it is open, it loads the program previously saved in U23 to RAM. It then jumps to the start address, defined at address 0x9002000. Program execution begins from there.

Problems

The COM0 port is not initialized when autorun is enabled. COM0 will work if the board was powered up in the monitor mode first. Make sure you initialize COM0.

FLASH DEMO PROGRAM

A flash EPROM demonstration program is in the EPROM1 subdirectory. This program uses HINT to download and run the program. C File name is EPROM1 and run time file is RPC400.SR.

Use the sample program for routines to save data or programs to flash.

MEMORY MAP - FLASH EPROM

The Flash EPROM is in CPU memory area 2. It is accessed 8 bits at a time, so its address is 0x2000000 to 0x207FFFF.

RAM in U24 is also in CPU memory area 2. It is accessed starting at 0x2080000. See SECTION 5 for more information on RAM.

See Table 5-1 for a complete memory map.

Access time
Flash EPROMs have access times of 100 nS. The wait state controller WCR1 should be set so area 2 has one long wait plus the number of wait states specified in WCR3. Refer to the SH-1 Hardware manual, Section 8, Bus State Controller, for more information. The BIOS leaves the register settings alone as the power up defaults are adequate.

If you are going to run programs directly from the flash EPROM or RAM, then you can shorten up the time to reflect the access time of the device you are using.

INTRODUCTION

The RPC-400 is available with either 2 or 4 serial ports. The 2 port version has RS-232 I/O. The 4 serial port version has three RS-232 I/O. The fourth is isolated and jumper selected for RS-232 or RS-422/485.

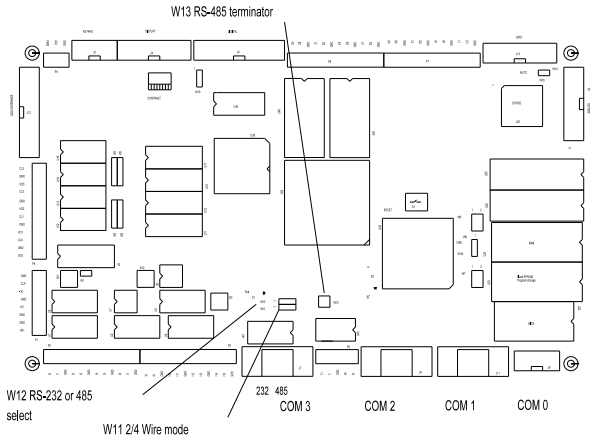


Figure 4-1 Serial Ports

Schematics are on pages 6 and 7.

COM0 AND COM1

Description

Both of these serial ports function identically. COM0 is used as the programming port. During run time, this port may be used like COM1. COM0 is initialized by the BIOS *only* when the auto run jumper is installed. For cold power up, COM0 must be initialized, if used.

Detailed programming information is in the *SuperH RISC Engine Hardware Manual*, section 13.

COM0 and COM1 are programmable for baud rates between 110 and 38400. (When you refer to the baud rate tables, the cpu clock is 19.6608 Mhz.) Other programming capability include data length at 7 or 8 bits, 1 or 2 stop bits, and none, even or odd parity.

COM0, at J9, is a 10 pin IDC connector. Its pin out is at the end of this section. Use the VTC-9F serial cable to connect from COM0 directly to a PC. Table 4-1 is the connector pin out.

COM1, at J11, is a 9 pin male D-SUB. Pin out matches that of a PC. Table 4-2 is its connector pin out. Notice that not all pins are used.

Initialization

The CMON BIOS initializes COM0 on power up or reset *and* auto-run is not selected. It is a good idea to initialize COM0 in your program if you intend to use this port during run time. The problem is when you auto-run your program after a power up, COM0 won't get initialized. Initialized settings are not changed on a push button reset.

COM0 is normally used as a debugging port. However, during run time it may be used as any other serial port *provided* you do not use the GDB debugger. You must use HINT or the device you intend to connect to COM0. You may use the GDB debugger if you don't care about any output from COM0.

All of the demonstration programs use either COM0 or COM1. Initialization routines for COM1 are in the 400IO sub-directory and file. The 400IO.C file also has general purpose get and put character and string routines.

RTS and CTS Lines

CTS and RTS lines are used for hardware flow control. The SH-1 processor does not have these lines as part of its UART. They are brought out digital ports on the CPU where its function may be emulated.

Names for CTS and RTS lines is a source of confusion. Normally, CTS is an output from a PC. On the RPC-400, CTS is an input on COM0 and an output on COM1-COM3. Conversely, CTS is a PC input while it is an output on COM0. The discussion that follows will use the PC convention. Keep in mind that the names and functions are reversed on COM0 only.

RTS is used by a receiver to indicate if "OK to send". A high level at the connector indicates yes, 'send data' while a low level indicates 'hold off'. This line goes to a sender CTS pin to signal conditions stated above.

The CTS input line on COM1 and RTS input on COM0 have a 4.7K ohm pull up to + 5V at the connector. This is to enable communication if this line was missing on the external device.

CTS works nicely as a hardware hold-off. It can generate an interrupt when it goes high and there are characters to transmit. However, when CTS checking is operated in polled mode, the line must be checked every time a character is transmitted. Should the receiver signal "hold off", the program must wait further until CTS goes back high.

There are several complicated schemes where this software CTS/RTS line monitoring problem can be solved, but they are complicated. The next best solution is to have the receiver send an XOFF/XON character to control flow.

CTS/RTS lines must be read or set by software. The program COMM1.C in directory COMM1 shows how these lines are read and manipulated.

COM2 AND COM3

Description
COM2 and COM3 use a 8250 software compatible serial interface chip. This is the same type of UART used in PC's. Capabilities include CTS and RTS control signals, programmable character lengths (5-8), and even, odd, or no parity. Baud rates are programmable from 50 to 115.2K. Each port may generate an interrupt on a transmit or receive.

Programming information about this chip is in Appendix A.

COM3 is optically isolated while COM2 is not. Both ports have PC compatible DB-9 male connectors. Pin out for these connectors are listed at the end of this section.

Initialization
The CMON bios does not check to see if the COM2 and COM3 serial chip (U43) is installed. This chip must be initialized as must the CPU interrupt ports.

Initialization routines and basic drivers (putchx, putsx, getchx) are in 400IO.C, in the 400IO directory. These routines are for polled mode only. For interrupt driven serial routines, look in the COMM02, COMM22 and COMM23 directories. Logic for COM3 RTS is reversed. Read next paragraph.

COM3 RTS Logic
Logic for COM3 RTS output is reversed from COM2 and normal PC operation. This was done because this line controls the RS-485 transmitter. The logic is such that on power up, the 485 transmitter is off. The RS-232 RTS output line on COM3 will be high, signaling OK to transmit to another device.

Interrupt Driven COM2 and COM3
Two sample routines, COMM22.C and COMM23.C operate COM2 and COM3 in interrupt mode. The major difference is COMM23.C operates the RS-485 serial port.

Buffer sizes were arbitrarily chosen at 256 bytes. It could be increased to any size. The buffers are circular. Read the notes in the files for more explanation.

There is plenty of processing power to operate all serial ports (COM0 - COM3) in interrupt driven mode at maximum baud rate. Using the sample COMM22.C program as a bench mark, interrupt service time is about 12 micro-seconds. Assuming a character is also received, 24 X 4 ports + overhead ≈ 100 micro-seconds per character. At the maximum baud rate of 57600 (not available on COM0 and COM1), there is still over 70 micro-seconds available for other processing, including interrupts.

ISOLATED COM3 SERIAL PORT

The COM3 serial port is an optically isolated RS-232 or RS-422/485 serial port. Isolation is adequate for low voltage (less than 100 V) circuits, when the board is operated in an environment relatively free of moisture and dust (Reference UL 1950, Table 3, Pollution degree 1 and 2). Transients should be less than 750 V.

> ***NOTE:*** The RPC-400 board was NOT tested to UL 1950 or any other standards. We do not imply it will meet this or any other standard. Isolation is provided to remove small AC ground and DC offsets normally found in long distance connections.

Both ports share the same UART. Consequently, only one port can be used at a time.

RS-422/485 are from screw terminals at P6. RS-232 is from a DB-9 male. See Figure 4-1 for connector location.

LED Activity
There are two LED's on the RPC-400 which blink on when there is transmit or receive activity. D1 is amber and blinks when receiving. D2 is green color and blinks when transmitting. Brightness is dim when sending few characters at a high baud rate.

Configuring Isolated RS-232
Set jumper W12[2-3] to enable RS-232 receive.

The CTS line at COM3 is pulled high through a 4.7K resistor to enable communication. This is provided should the external device be missing the CTS line.

Configuring Isolated RS-422/485
There are three sets of jumpers affecting RS-422/485. Refer to Figure 4-1 for the location of these jumpers. Each is described below.

Set jumper W12[1-2] for all 422/485 communication. This jumper selects the receive signal.

Jumper W11 is set in one of two positions, depending upon the communication standard and mode desired.

W11[1-2] 4 wire RS-485 mode (separate TX and RX lines) or RS-422 mode. This always enables the receiver.

W11[2-3] 2 wire RS-485 mode. Externally wire TX+ to RX+ and TX- to RX-. This turns off the receiver when transmitting. The RTS3 line is manipulated to do this. (More later)

W13[1-2][3-4] Network terminators. Set only when the RPC-400 is physically the last board in a RS-485 system. Otherwise, remove these jumpers. In RS-422 systems, keep these jumpers in.

The RTS3 line acts the same way as in RS-232. The difference here is it controls both the receiver and transmitter. In two wire mode, it prevents the data sent out from looping back and getting received. RTS3 line is set in the UART MCR to '0' to enable transmitting and disable receiving and '1' to disable transmitting and enable receiving. See demo program COMM23.C for a sample of RS-485 in 2 wire mode.

RS-485 is best when operated in interrupt mode. This way, the transmitter is shut off when the last character is sent out. See COMM23.C for example.

In practice, there is little difference between 2- and 4-wire RS-485. Hardware simultaneously controls the receiver and transmitter. The only real difference is the board cannot receive data while transmitting in a 2-wire system.

The RS-232 transmitter is always enabled. Use this port to monitor transmit activity on RS-485 to a terminal or another PC.

RS-422 is a long distance version of RS-232. The transmitter and receivers are always on. To use P6 as a RS-422, simply set RTS3 line high. See program comm23.c for code and comments. Set jumper W11[1-2] and W12[1-2]. The terminator at W13 should be set to reduce ringing and noise.

Wiring for RS-485
Four wire systems simply connect corresponding TX± and RX± lines. Unfortunately not all systems are marked the same so you may have to play around with signal wires.

The 'GND' line at the 485 connector is for the cable shield. This line goes through a 100 ohm resistor to an internal, floating ground.

Using RS-485
RS-485 is used in a multi-drop (networked) environment. The authors of this standard do not specify a protocol. RS-485 is only a hardware specification. 32 units over a 4,000 foot range can be connected together.

There are several questions users have when using RS-485. Two of the most popular are: What baud rate should I use (or what is the maximum) and the second is about the protocol.

The maximum baud rate depends upon the environment all of the boards will operate in. If you are installing a network, or can specify cable type, make sure it is one for RS-485. This cable has low capacitance, twisted signal wires inside a shield. Belden type 9842 or 9844 is a good starting point for cables. Critical characteristics are twisted pair with shield, ≈100 ohm impedance cable, and low capacitance. Capacitance is not critical when distance is short (< 1000 ft).

Ultimately, the only way to determine this is to view the RS-485 signal using an oscilloscope. Set up the "nearest" transmitter to continuously send a signal. Look at the result on the "farthest" receiver. A rough rule of thumb is maximum baud rate should be 6 times or more of the worse of rise or fall times. The idea is to present a clean, steady signal to the receiver. Rise and fall time should be less than 30% of the total bit width. Measurement should be taken at the farthest receiver with all devices connected. The farthest receiver should have its terminator installed. No programs have to be

running. Our customers experience is has been 4800 to 19200 baud using 3000 to 5000 feet of cable.

The second question is about protocol. RS-485 does not specify one. Generally, you can treat the data format the same as RS-232.

A 4-wire multidrop network includes a host and one or more devices. This is a master-slave system. the host directs all communication. Nodes "do not speak unless spoken to."

There are many master-slave protocols. For this example, a protocol might look something like this:

```
>22M1B1
```

The protocol starts with the < cr> character. This character synchronizes all units and alerts them that the next few characters coming down are address and data. In this case, "> 22" is the units address. "M" is the command, "1" is data, and "B1" is the checksum. The command is terminated with a < cr> character.

Figure 4-2 Data packet

The response depends upon the nature of the command. Suppose the command M means "return the belt speed" and 1 is the belt number. The RPC-400 could read the port and respond with A2.34< cr> . The first A is an acknowledge, that is no errors were detected in the message. The data, 2.34, can mean feet per unit of time (minute or second).

A 2-wire network can have multiple hosts. This "peer to peer" network can be a bit more complicated. The problem is avoiding or resolving conflicts when two peers speak at the same time. 2-wire protocols can be master-slave, the same as 4-wire.

A slow, but reliable peer to peer network is to use round-robin communication. Suppose on power up unit 1 sends out a message. All the other units recognize to whom the message is intended. If the receiver is off line, nothing is returned. Unit 1 can't send another message until it is spoken to or its turn is up. After a period of time, unit 2 has permission to use the line. In a similar manner as unit 1, it sends out a message and waits for a response. If unit 2 is off line, unit 3 will speak after unit 2's time out is complete.

Additional messages can be generated to speed things up. For example, one is to tell the next unit in line "I have nothing, you go ahead."

SERIAL DEMO PROGRAMS

COM0 and COM1 demo programs are peppered in all of the sub-directories. All of the counter programs (PWMx) use COM1 for main output while analog input (AINx) use COM0.

COM1 initialization routine is in 400IO.C, in the 400IO subdirectory. COM0 initialization is the same except for the addresses. Interrupt routines are contained in the entire program. This includes the interrupt vector table and enable routines.

Print routines such as getchx, putchx, putsx, and printfx are either in 400IO.C or PRINTFx.C files. These routines are in the 400IO.A library, called by all of the other programs. The 'x' after the function name (i.e. putch*x*) refers to the port number.

Other serial port routines are:

COMM01	Read/control CTS/RTS lines for COM0 and COM1
COMM02	Interrupt driven terminal program, with error handling, for COM0 and COM1. Drivers in 400IO.C library are not used but are in this program.
COMM21	Basic terminal program for COM2 and COM3.
COMM22	Interrupt driven example for COM2 and COM3. Note that routines in 400IO.C library are not used.
COMM23	Interrupt driven RS-485.

SERIAL CONNECTOR PIN OUTS

Table 4-1
COM0 connector pin out is as follows:

Pin	Name	Direction from card.
3	Tx	Out
4	RTS	In
5	RXD	In
6	CTS	Out
9	Ground	
10	+ 5V	

A VTC-9F serial cable for COM0 is made by simply taking a 10 pin female IDC connector and crimping a 9 wire ribbon cable to it. The, crimp a 9 position female D-SUB connector to the other end of the cable. Wiring is one to one.

Table 4-2
COM1 - COM3 RS-232 pin out is as follows

D-SUB Pin	Name	Direction from card.
2	Rx	In
3	TX	Out
5	Ground	
7	RTS	Out
8	CTS	In

Pin out from the 9 pin male D-SUBs matches those on a PC. Use a null modem adapter when connecting between a PC and COM1 - 3.

COM2,3 ADDRESS AND INTERRUPTS

Port	Base Address	IRQ
COM2	0x6000080	1
COM3	0x6000100	2

See Appendix A for detailed U43 UART programming information. IRQ 1 and 2 are CPU interrupts.

INTRODUCTION

There are several different memory types and areas. Their types, functions, and address locations are shown in Table 5-1 below.

The application program is stored in a 5-volt only Flash Programmable and Erasable Read Only Memory (PEROM) in socket U23. See *SECTION 3, SAVING PROGRAMS* for saving information.

Socket U24 holds 32K, 128K, or 512K of RAM. Optionally, a clock/calendar with battery backup may be installed in this socket (see SECTION 7 for installation information). Memory in this socket is intended to hold data, although programs could run from it at greatly reduced speed.

Sockets U25 and U26 use 128K byte high speed RAMs for program execution. These sockets are organized as 128K X 16, for a total of 256K bytes.

Sockets U25 and U26 can be configured for 512K X 8 high speed RAMs through jumper W6. This option was not available at the time this manual was printed. Contact Remote Processing for an update.

ACCESSING RAM

Review the sample program in 'EPROM1' and RAM1 directories. EPROM1 transfers data from RAM to flash a character by character basis. Integers and other numbers are read from or written to in a similar manner. RAM1 stores all data types to RAM. View the results using the monitor.

INSTALLING RAM

Data RAM U24
Socket U24 holds 32K, 128K or 512K RAM. Set jumper W8 to the '128K' position for 32K or 128K RAMs and to the '512K' position for 512K.

To install RAM, orient the chip so pin 1 is near the outside edge of the board. If you are installing a calendar/clock module or battery backup module, install this into socket U24 first. When installing 128K or 512K RAMs, simply insert the RAM into the socket.

32K RAMs are installed by leaving pins 1,2, 31, and 32 open in the socket. Orient the RAM so pin 14 goes into socket pin 16. The top rows are open.

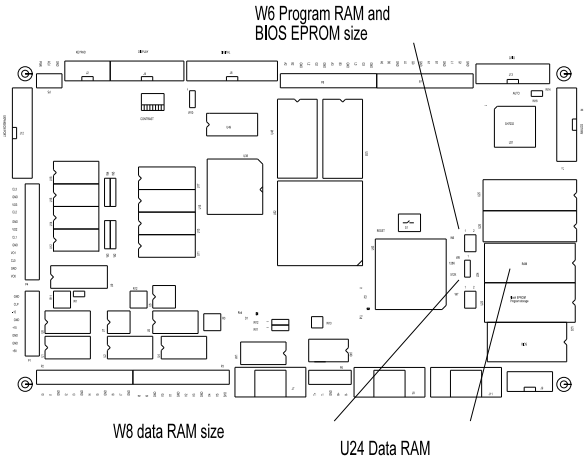


Figure 5-1 Memory

Location	Type	Function	Address range	Memory Area	Access Width	Max size
U22	ROM	BIOS	000 0000 - 000 FFFF	0	8	64K
U23	Flash	Program storage	200 0000 - 207 FFFF	2	8	512K
U24	RAM	Data and clock	208 0000 - 20F FFFF	2	8	512K
U25, U26	Fast RAM	Program execution	900 0000 - 90F FFFF	1	16	1 M
U21	CPU RAM	General purpose	FFF E000 - FFF FFFF	7	32	8K

Table 5-1 Memory Map

Program execution U25, U26
High speed 512K RAMs in DIP form were not readily available at the time of this printing. Remote Processing may make an adapter board so higher density RAMs may be installed. This section deals with changing the jumpers.

Jumper block W6 sets the BIOS and program execution RAM size. To use 128K RAMs, set jumper W8[3-4]. For 512K RAMs, set W8[4-6].

RAM installation will be a matter of removing the old parts and installing the new ones.

BIOS EPROM

The BIOS resides in socket U22. The BIOS is accessed on power up and during program development. As shipped, it is a 32K byte device. Normally, you will not have to worry about it.

If you modify the BIOS and program size exceeds 32K, you will need to use a 27C512 EPROM. Jumper W6[1-2] must be set to access this 64K device.

INTRODUCTION

Digital I/O lines are used to interface with opto-module racks, switches, low current LED's, and other TTL devices. These lines are brought out via STB-26, STB-20, or MPS-XX opto racks.

Digital lines are available from multiple sources. Connector J6 is considered the primary source. Additional lines usable as digital I/O are available from the keypad (J2), display (J4), and CPU (J1 analog input and J13 GPIO) for a total of 69 digital input and output lines. Some of these lines are intended for other functions and its use must be considered in the system design.

General warnings and precautions

WARNING:

Apply power to the RPC-400 before applying a voltage to the digital I/O lines to prevent current from flowing in and powering the board, damaging devices. If you cannot apply power to the RPC-400 first, contact technical support for suggestions appropriate to your application.

When lines are configured for outputs at any of the 82C55 ports (J2, 4, and 6), outputs go low until set high. Low lines turn on opto modules and potentially other devices. One solution is to set the output lines immediately after configuring them. Depending upon how you have written your program, lines can be low for less than a micro-second. This low time can be enough to cause problems with some devices. Power opto modules are generally not affected.

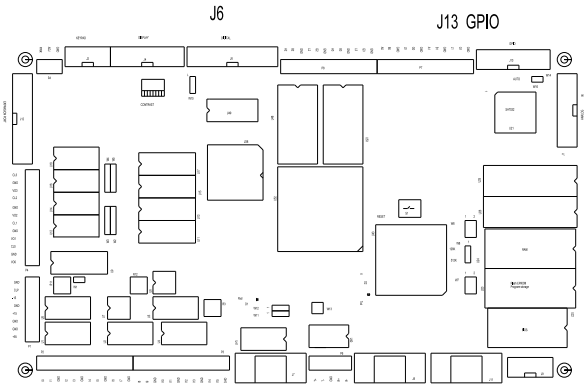


Figure 6-1 Digital I/O

J6 DIGITAL I/O

This port is considered the main digital I/O port. It is used to interface opto modules (using MPS series racks), drive small relays, solenoids, motors, or lamps, and provide general purpose TTL I/O to other logic devices, or mechanical switches. J6 is shown on schematic page 5. Its address is 0x6000000 through 0x6000006. See the program in the DIO directory for an access example.

Lines on J6 are divided into 3 eight bit groups from an 82C55. Refer to table 6-2 for a list of configuration commands to write to the 82C55. A byte is written to address 6 (0x6000006). These command bytes configure ports A, B, and C for inputs and outputs as shown.

When a line is configured as an output, it can sink a maximum of 2.5 mA at 0.4V and can source over 2.5 mA. Outputs sink over 15 mA at 1.0v, enough to drive opto modules.

Port B is connected to a high current sink through U49. See "High Current Output" later in this section.

Digital I/O lines are pulled to + 5 volts or ground through 10K or 100K resistor packs using jumper W10. 82C55 port A is pulled up or down through 10K. Ports B and C through 100K. Upon reset or configuration as an input, the lines will then be pulled high or low as your system requires.

Jumper W10 configuration is as follows:

W10[1-2] Pull up
W10[2-3] Pull down

Setting W10 for pull up makes interfacing to switches and "open collector" collector TTL devices easy. See "Interfacing to Switches and other devices" below.

High Current Output

Eight lines at J6 can be used as high current drivers. These outputs switch loads to ground. Outputs are controlled by Port B on the 82C55.

Logic outputs are inverted. When a 1 is written to a line, the output is switched ON and goes low.

The output driver chip U49 can be replaced with a DIP shunt jumper so it is like the other lines at J3. To do this, remove U49. Install a DIP shunt so pin 1 goes to pin 18. Pins 9 and 10 are open.

NOTE: High current outputs are not compatible with TTL logic levels and should not be used to drive other logic devices.

Each of the high current outputs can sink 500 mA at 50V. However, package dissipation is exceed if all outputs are used at the maximum rating. The following conservative guidelines assume the number of outputs are on simultaneously:

# of outputs on	Maximum current per output
1	500 ma
2	400 ma
3	275 ma
4	200 ma
5	160 ma
6	135 ma
7	120 ma
8	100 ma

The thermal time constant of the package is very short, so the number of outputs that are on at any one time should include those that overlap even for a few milli-seconds.

Incandescent lamps have a "cold" current of 11 times its operating current. Lamps requiring more than 50 mA should not be used unless a series resistor is installed.

Protection diodes must be used with inductive loads. Refer to Figure 6-2.

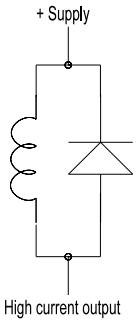


Figure 6-2 Inductive load protection

Do not parallel outputs for higher drive. This could result in damage since outputs will not share current equally.

Outputs at U12 are open collector.

Interfacing Digital I/O to an opto-module rack

I/O lines can be interfaced to an MPS-8, 16, or 24 position opto module rack. Lines not going to an opto module connect to a screw terminal on the racks. This feature allows you to connect switches or other TTL type devices to the digital I/O lines. The MPS-XX series boards accept G4 series modules.

A CMA-26-24 connects J6 on the RPC-400 to the MPS-XX board. Cable length should be less than 2 feet. Excessive cable lengths cause a voltage drop and consequently unreliable operation. This is because the 28 gauge wire in the ribbon cable has sufficiently high resistance. Make sure you connect + 5 V and ground to the MPS-XX racks.

Before a line is set, the 82C55 chip must be initialized. Refer to Table 6-2 for initialization parameters.

Refer to Table 6-1 for Opto module position, port number, and connector pin out. If opto channels 16-23 are used, U49 should be replaced by a DIP shunt jumper.

Interfacing to switches and other devices
Switches and other digital I/O devices may be connected

directly to J6. The STB-26 terminal board provides a convenient way of interfacing switches or other digital I/O devices. Lines at J6 are connected to the STB-26 with a CMA-26 cable. Digital devices are then connected to the screw terminals on the STB-26. Refer to Table 6-1 for J6 connector pin out description. The MPS-XX series opto racks also provide a way to access digital I/O lines.

Switches may be connected directly to a line. When jumper W10 configures the resistors as pull ups, a switch closure to ground at a line is read as a 0. When running long leads (greater than 5 feet) or in noisy environments, connect a 1K ohm resistor between + 5V and the switch.

When W10 configures the input resistors as pull downs, one end of the switch must be tied to + 5 volts. If this is not possible or convenient, a 10K resistor can be tied between an input and + 5 volts to force it high when a switch is open.

Connector Pin Out - J6

Table 6-1 Connector pin out - J6

Pin #	82C55	Description	Opto Channel
19	Port A, line 0		8
21	Port A, line 1		9
23	Port A, line 2		10
25	Port A, line 3		11
24	Port A, line 4		12
22	Port A, line 5		13
20	Port A, line 6		14
18	Port A, line 7		15
10	Port B, line 0	High current	16
8	Port B, line 1	High current	17
4	Port B, line 2	High current	18
6	Port B, line 3	High current	19
1	Port B, line 4	High current	20
3	Port B, line 5	High current	21
5	Port B, line 6	High current	22
7	Port B, line 7	High current	23
13	Port C, line 0	Lower C	0
16	Port C, line 1	Lower C	1
15	Port C, line 2	Lower C	2
17	Port C, line 3	Lower C	3
14	Port C, line 4	Upper C	4
11	Port C, line 5	Upper C	5
12	Port C, line 6	Upper C	6
9	Port C, line 7	Upper C	7
26			Ground
2			+ 5V

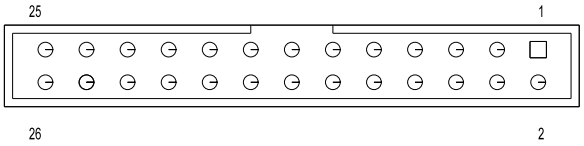


Figure 6-3 J6 pin out (viewed from top)

Table 6-2 82C55 Commands for J2, 4, and J6

Command value	Port A	Port B	Port UC	Port LC
0x80	OUT	OUT	OUT	OUT
0x81	OUT	OUT	OUT	IN
0x82	OUT	IN	OUT	OUT
0x83	OUT	IN	OUT	IN
0x88	OUT	OUT	IN	OUT
0x89	OUT	OUT	IN	IN
0x8A	OUT	IN	IN	OUT
0x8B	OUT	IN	IN	IN
0x90	IN	OUT	OUT	OUT
0x91	IN	OUT	OUT	IN
0x92	IN	IN	OUT	OUT
0x93	IN	IN	OUT	IN
0x98	IN	OUT	IN	OUT
0x99	IN	OUT	IN	IN
0x9A	IN	IN	IN	OUT
0x9B	IN	IN	IN	IN

Port A and B are either all inputs or all outputs. Each half of Port C is programmable. Upper C (UC) is bits 4 through 7 and Lower C (LC) is bits 0-7.

J13 GPIO

J13 is a general purpose I/O port. It is made up of multi-function lines programmable as timers, interrupts,

and bit programmable I/O. Refer to schematic page 3 for a wiring diagram of this port. Table 12-1 is a connector pin out and general alternate function.

Your system requirements determine the function of the lines. This section discusses using some or all lines for simple I/O functions. Refer to the interrupt and counter sections of this manual for other uses.

Port B I/O

Primary CPU registers affecting port B functions are: PFC_PBIOR, PFC_PBCR1, PFC_PBCR2, and PBDR. PFC_PBIOR determines which lines are inputs and outputs. PFC_PBCR1 and PFC_PBCR2 determine the function of a line such as interrupt, timing pattern, I/O, or other as described in the hardware manual. See SH7032 hardware manual sections *"Pin Function Controller"* and *"Parallel I/O Ports"* for more information.

On power up, the BIOS programs port B as follows:

PB0 - PB7 are outputs
PB8 - PB11 is serial communications
PB12 - PB15 are inputs

Three registers are used to configure and access port B. Section 15.3.3 in the *Hitachi SH7032 Hardware Manual* describes PBIOR, which, in general, determines what lines are inputs and outputs. PBCR1 and PBCR2, described in section 15.3.4, select the functions of the pins. Section 16.3.2 describes PBDR, which is the actual data I/O register. A write to this register determines the output level while a read returns the current status.

Port B can also be used for counting/timing functions. Refer to *Section 13, COUNTING/TIMING* for more information on alternate uses.

See the DIO directory for port access example.

Pins are capable of driving standard TTL loads. The table below shows the current output at a level.

Item	Current (ma)
Output low level per pin	10
Output low level total	80
Output high level per pin	2.5
Output high level total	25

These levels apply only to J13. Input currents are 1 ua. Do not exceed + 5V positive or go lower than 0V.

J2, J4 OPERATOR INTERFACE

Ports J2 and J4 are intended to interface to a keypad and display, respectively. If you are not using one or the other, then they are available as digital I/O in a manner very similar to J6, discussed above.

These ports interface to an 82C55. Consequently, programming and electrical characteristics are the same as for J6. The exception is neither J2 or J4 has a high current output. Use Table 6-2 above to configure the 82C55 ports.

This port is accessed from 0x6000180 to 0x6000186. The schematic for J2 and J4 is on page 4.

If you use only a few of the lines on either port as inputs, be sure to tie any unused inputs to ground or + 5V. Failure to do so could cause the board to draw excess current and may damage the 82C55 device. Unused ports may be configured for output in the initialization section of your code.

Table 6-3 and 6-4 show the pin outs for J2 and J4. The STB-20 brings out lines from J4 to a terminal strip for easy access.

Table 6-3 Connector pin out - J2

Pin #	Function
1	ROW 1, 82C55 port C, bit 0
2	COL 3, 82C55 port C, bit 6
3	COL 2, 82C55 port C, bit 5
4	ROW 2, 82C55 port C, bit 1
5	ROW 3, 82C55 port C, bit 2
6	COL 1, 82C55 port C, bit 4
7	COL 4, 82C55 port C, bit 7
8	ROW 4, 82C55 port C, bit 3
9	COL 5, 82C55 port B, bit 0
10	COL 6, 82C55 port B, bit 1

Note that two lines from 82C55 port B go to this connector.

There is a 10K pull up resistor to + 5V on ROW1 through ROW 4 only. All other lines are open.

Table 6-4 Connector pin out - J4

Pin #	Function

1	+ 5V supply
2	Ground
3	82C55 port A, bit 4
4	LCD contrast bias
5	82C55 port A, bit 6
6	82C55 port A, bit 5
7	82C55 port B, bit 7
8	82C55 port B, bit 3
9	82C55 port B, bit 2
10	82C55 port A, bit 7
11	82C55 port A, bit 1
12	82C55 port A, bit 0
13	82C55 port A, bit 3
14	82C55 port A, bit 2
15	82C55 port B, bit 4
16	82C55 port B, bit 6
17	82C55 port B, bit 5
18	To P5, 'ADJ' pin
19	To P5, 'PWR' pin
20	Ground

There are no pull up resistors on J4. All lines are open.

J1 ANALOG INPUT AS DIGITAL IN

J1 is a 10 bit, 8 channel A-D converter. The converter is an integral part of the CPU. The technique here is to simply perform A-D conversions and read the result. Results above 0x300 are a logic '1' while those below 0x100 are a logic '0'.

Use this input for higher voltage (> 5V) inputs. A series resistor is necessary to scale down the voltage. The principle remains the same as reading TTL lines.

Refer to section 10 for information using J1.

MEMORY MAP - DIGITAL I/O

The following are addressees used to access the various digital ports. Refer to the demonstration disk for driver examples.

Port	Address	Function
J6	0x6000000	82C55 port A
J6	0x6000002	82C55 port B
J6	0x6000004	82C55 port C
J6	0x6000006	82C55 configuration register
J4	0x6000180	Display, 82C55 port A
J4	0x6000182	Display & keypad, 82C55 port B
J2	0x6000184	Keypad, 82C55 port C
J2,J4	0x6000186	82C55 configuration register
J13	0x5FFFFC2	CPU port B data, PBDR
J13	0x5FFFFC6	CPU port B I/O, PBDR
J13	0x5FFFFCC	CPU port B configuration, PBCR1
J13	0x5FFFFCE	CPU port B configuration, PBCR2

INTRODUCTION

An optional DS1216DM or DS1216D512 calendar/clock module may be installed in U24. These modules also battery back RAM. The DS1216DM backs up 32K and 128K RAMs while the DS1216D512 backs up 512K bytes.

These modules from Remote Processing are a modified version of the Dallas DS1216D. Internal lines are cut and soldered, depending upon the version.

Battery life depends greatly upon the ambient temperature. Battery life degrades up to 50% at 50° C, using 25° C as a reference. Generally, you can expect a battery life of 3 to 5 years. The clock module uses a dual battery system, meaning the RAM battery can get used up before the clock. There is no software way to detect a low battery.

Accuracy is about 1 minute/month and is not adjustable.

INSTALLATION

The clock module is installed by first removing the IC in socket U24 (if installed). See Figure 5-1 for IC location. Install the DS1216 module into the socket. Note the notch on the socket designating pin 1. Align this with the notch on the board.

SETTING AND READING THE CLOCK

Review the sample program (rtc.c) in the RTC directory.

The following briefly describes the operation and registers in the DS1216D module. For detailed information, contact Dallas Semiconductor at 214-450-0448, Fax: 214-450-040, or www.dalsemi.com.

Operation
The clock module is turned off as shipped from the factory. The clock is turned on as part of the initialization routine.

The SmartWatch is read and set via serial bit stream on the data bus. Normally, the module operates transparently for RAM. When a specific 64 bit pattern is sent, it is unlocked. At this time, the next 64 read or write cycles extract or update data in the Smartwatch.

Normally a specific address is used while communicating with the chip. During the unlocking phase, RAM data is modified. If you have a RAM installed, data at the address used should be saved. The RTC.C program in the RTC directory does this. This program also has programming information pertaining to specific bits and modes of operation.

After unlocking, the registers are either read or written to. Date and time information is in BCD format. All registers must be written to or read from as the RAM chip is disabled until 64 cycles are completed.

12/24 Hour Mode
Bit 7 in the hours registers selects 12/24 hour mode. A 1 selects 12 hour mode. When 12 hour mode is selected, bit 5 indicates AM/PM. PM is indicated when bit 5 is high. In the 24 hour mode, this bit is the second 10 hour bit.

Module Control
Two bits in the day register control the reset (bit 4) and oscillator (bit 5) functions. The reset line was used to abort a data transfer. This line is cut by modifications in the DS1216D and DS1216D512 and is not a consideration. Set bit 4 to a 1.

The oscillator turns clock on and off. When bit 5 is set to 1, the oscillator is off. When set to 0, the watch is operational.

Both bits are set to 1 at the factory.

Zero Bits
Registers 1-6 have one or more bits which always read 0. Writing a 1 or 0 to these locations is OK.

Year 2000
The DS1216D series modules return years as 00 to 99. It is your responsibility to determine the millennium.

BATTERY BACKED RAM

RAM is not backed up until power is first applied to the module. After that, 3 volts is supplied to pins 14 and 28 (32K) or 16 and 32 (128K or 512K) RAMs when power is off.

INTRODUCTION

The display port supports vacuum florescent (VF) character, LCD character, and LCD graphic displays. LCD graphic display is also supported on the expansion bus at J12.

Sample programs are located in the following directories:

Display type	Directory
LCD 4 x 20	LCD420
LCD 4 x 40	LCD440
Vacuum Florescent	VFD
LCD graphics @ J4	GRAPH1
LCD graphics @ J12	GRAPH2

Sample programs provide basic character positioning and printing routines. The graphics display routines also provide dot and line drawing examples.

Power for VF and graphics displays is brought in at P5. VF displays draw about 500 mA. Bringing + 5V and ground from the power supply to P5 at "PWR" reduces line losses and ground loops on the board.

The graphics display require external parts. -21V is required for graphics displays. It is brought in to P5 "PWR". A 10K-50K contrast pot is also connected to P5. The center wiper of the pot is brought to "ADJ", while the other ends are brought to "GND" and "PWR". The "PWR" pin may have two wires in it if the pot is directly connected to P5.

The graphics display may be connected to the expansion bus at J12 for slightly faster operation or dual displays. The same display cable may be used. The only modification is a 26 pin IDC connector is crimped on to the cable. Pin 1 on the cable aligns with pin 1 on the connector. Driver example for this mode is in directory GRAPH2. The primary difference between GRAPH1 and GRAPH2 programs are basic drivers to the display. Writing and display algorithms remain the same.

A 82C55 is used to drive displays at J4. This same 82C55 is used for the keypad port at J2. To use the display port, 82C55 ports A and B are configured for outputs. Refer to table 6-2 for 82C55 configuration commands.

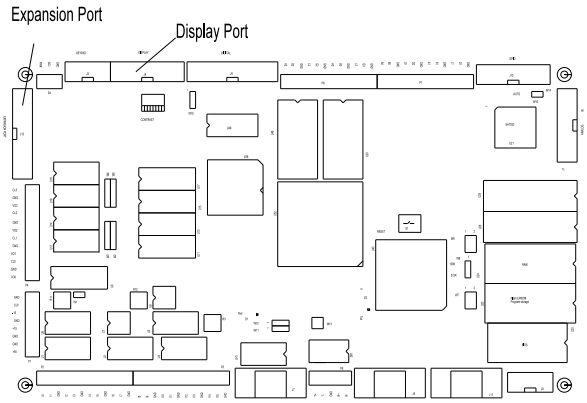


Figure 8-1 Display and Expansion ports

INTRODUCTION

Up to a 24 position keypad is plugged into keypad port J2. Keys are arranged in a matrix format. A key is detected by software when a row and a column connect. Software scans the key port by writing to port B and upper port C and reading from lower port C on an 82C55. The 82C55 must be initialized. Refer to Table 6-2 for configuration commands. This 82C55 is also used for the display port.

The demonstration program in the KEYPAD directory scans and debounces the keypad every 50 ms. Keypad presses are returned as a number from 1 to 16. Up to 8 key presses are buffered.

Keypads from Remote Processing simply plug into J2. Keypad cable length should be limited to less than 5 feet.

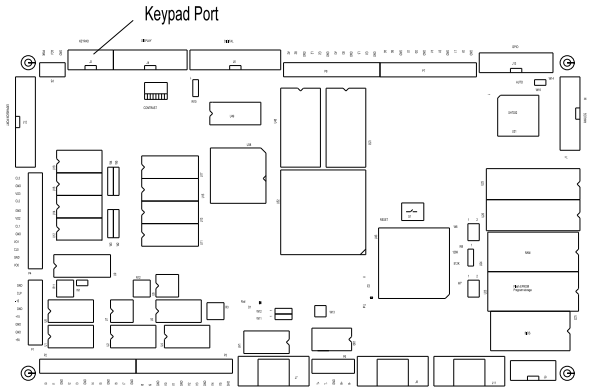


Figure 9-1 Keypad connector

KEYPAD PORT PIN OUT - J5

The keypad port uses ports B and C from an 82C55. Lower port C is configured as an input. Upper port C and port B bits 0 and 1 are outputs.

The table below lists J5's pin out, 82C55 port and bit, and its intended function.

Pin	82C55 Port/bit	Function
1	C/0	Row 1
2	C/6	Column 3
3	C/5	Column 2
4	C/1	Row 2
5	C/2	Row 3
6	C/4	Column 1
7	C/7	Column 4
8	C/3	Row 4
9	B/0	Column 5
10	B/1	Column 6

INTRODUCTION

The RPC-400 has two analog to digital (A-D) converter systems. One is 10 bit resolution and is built into the CPU. The other is a 12 or 16 bit A-D. The schematic for this subsystem is page 1. Each section is described below.

12/16 BIT A-D CONVERTER

The RPC-400 is available in 12- or 16- bit A-D converter models. Operationally they are identical. The same basic software is used to access a 12 bit converter will work with 16 bit converters. Of course, scaling factors change in order for readings to make sense.

There are hardware differences between the 12- and 16-bit models. The 16 bit has trim pots and associated components, a separately regulated A-D supply, and U9 is different.

There are several demonstration programs: ain1.c and ain2.c. ain1.c is interrupt driven while ain2.c is not. ain3.c prompts for user input to convert a specific channel at a gain and mode. It is useful for quickly determining filter, gain and mode parameters of an input signal. ain4.c is used for the 10 bit converter, discussed later in this section.

From here forward, assume operation between the 12- and 16- bit converters is the same unless specifically stated otherwise.

Both converters have 16 software programmable gains from X1 to X800, 16 single ended or 8 differential inputs in any combination, and unipolar/bipolar ranges. Additionally, a shield driver is available to help reduce noise on long lead, low level signals.

Input Ranges
The maximum single ended full scale input voltage at P2 and P3 are 0 to 5V in unipolar mode and ±5V in bipolar mode, at X1 gain. If you are using a gain other than X1, maximum input voltage is determined by the following formula:

$$\text{Full scale input voltage} = \frac{5}{\text{gain}}$$

For example, a gain of 10 limits the maximum input voltage to 0.5V.

Each channel has programmable gain. Apply this formula to any channel that changes gain.

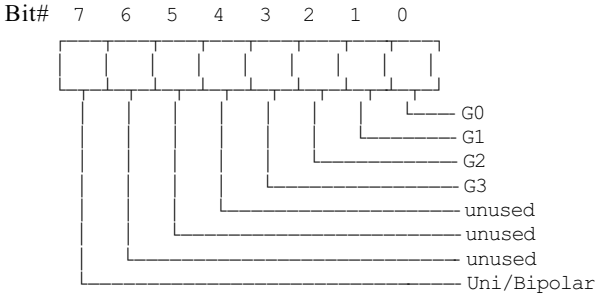
Bipolar range is the same for negative voltage as positive.

Programming Gain, Bipolar/Unipolar
Sixteen gains from x1 to x800 are programmable for any channel, single ended or differential, bipolar or unipolar inputs. Gain and polarity are programmed by writing to register RW4 at address 0x60002a0.

Input range is programmable for 0 to 5V or ±5V (or full scale if using gain) using the uni/bipolar bit.

The gain chips are U5 and U7. They are programmed as shown below.

Table 10-1 Gain and polarity



Where:
G0 - G3 select system gain.

G3	G2	G1	G0	Gain	% Error
0	0	0	0	X1	0.01
0	0	0	1	X2	0.015
0	0	1	0	X4	0.015
0	0	1	1	X8	0.015
0	1	0	0	X10	0.025
0	1	0	1	X20	0.03
0	1	1	0	X40	0.03
0	1	1	1	X80	0.03
1	0	0	0	X100	0.045
1	0	0	1	X200	0.05
1	0	1	0	X400	0.05
1	0	1	1	X800	0.05

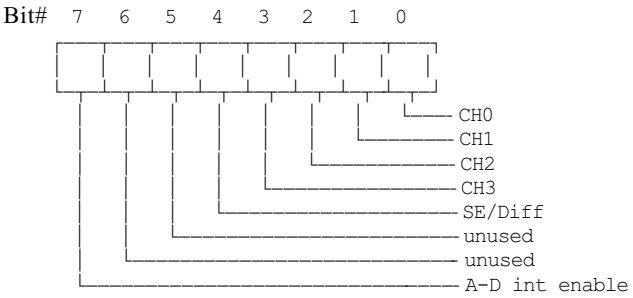
Bit 7 controls unipolar/bipolar modes. 0 = unipolar (0-5V) and 1 selects bipolar (±5V).

The "% Error" column is the percentage of gain error possible. It is discussed under *Conversion Accuracy* below.

Channel Selection & Differential or Single Ended Inputs

Channel selection is performed by writing to register RW3 at address 0x6000298. The table below shows values for a particular mode and channel.

Table 10-2 channel select, mode, and interrupt



Where:

CH0 - CH3 are channel to convert per table below.

Channel	SE/Diff.	No. to write
0	SE	0x00
1	SE	0x01
2	SE	0x02
3	SE	0x03
4	SE	0x04
5	SE	0x05
6	SE	0x06
7	SE	0x07
8	SE	0x08
9	SE	0x09
10	SE	0x0a
11	SE	0x0b
12	SE	0x0c
13	SE	0x0d
14	SE	0x0e
15	SE	0x0f
0,8	Diff	0x10
1,9	Diff	0x11
2,10	Diff	0x12
3,11	Diff	0x13
4,12	Diff	0x14
5,13	Diff	0x15
6,14	Diff	0x16
7,14	Diff	0x17

When the SE/Diff bit = 0, single ended mode is selected. 1 = differential mode.

Channels are switched dynamically. You could assign channel 0 and 8 as a differential input and have channels 2-4 as single ended.

Setting bit 7 high enables A-D interrupts while a 0 does not enable them. Set this bit high before starting a conversion.

Differential Mode

Relatively positive inputs go to channels 0-7 while the more negative are on channels 8-15. The converter should be operated in bipolar mode in most applications. When the difference is negative, the most significant bit of the A-D conversion is set.

Differential inputs are fed through U4 and U5 (see schematic page 1). Measured Common Mode Rejection Ratio (CMRR) on a prototype board was at least -85 dB at 120 Hz and -78dB at 400 Hz at X800 gain.

For 12-bit systems, this means that there could be about 16 counts of error induced by CMRR at the maximum gain of X800. In 16 bit systems, noise could be closer to 256 counts of error, or change, from average maximum to minimum. You will see these kinds of error when the common mode input is 14 volts p-p.

Maximum common mode voltage is about 2 volts below the maximum + V supply and 2 volts above the -V supply.

A simplified schematic of a bridge circuit interfacing with the front end is shown below.

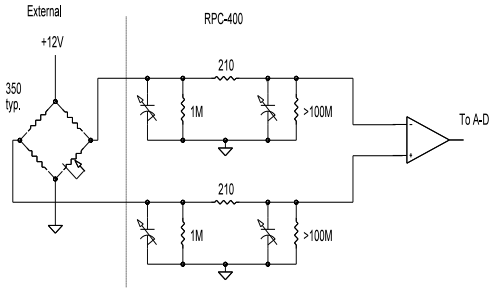


Figure 10-1

'External' section is a typical bridge circuit used in pressure transducers. 350 ohms represents typical resistance.

The variable capacitors represent cable, board, and IC capacitance. The 1 Meg ohm resistor is the input resistor while 210 is the resistance of the multiplexers.

A large source of 'noise' is due to phase differences between input signals. This phase difference occurs when signal sources are high impedance. One way to correct phase differences is to put a small variable capacitor (10-35 pf) at one of the inputs and monitor shield driver output on P3. Adjust the capacitor for minimum AC output. This technique works only when the source impedance is high (> 10K ohm).

Another way to reduce noise errors due to CMRR is to take multiple samples over the common signal period. For a 60 hz common mode signal, this means taking several measurements for 16.66 ms. Try to take only enough samples to cover the period of one common mode signal. Taking samples over a 20 ms period in a 60 Hz environment will amplify positive or negative signal components.

Shield Driver
The shield driver sums the common mode signal from both + and - inputs on the currently selected channel pair. Use this driver to help reduce noise induced in a cable. Note the shield driver is not ground or common reference. Do not tie the shield to ground at any point.

Allow a few micro-seconds settling time after switching channels and gain before starting a conversion. The driver OP amp and previous circuitry need some time to adjust to new signal conditions.

Settling Time
You will need to allow a certain amount of time after switching channels before starting a conversion. The amount depends upon the gain and difference in voltage between the two channels.

At X1 gain, the slew rate is 0.4 V/micro-second. This means if there is a 4 volt difference (positive or negative) between two channels, you should wait at least 10 micro-seconds before initiating a conversion. At X800 gain, slew rate is worse. To get the best reading, wait at least 30 micro-seconds after changing channels before starting a conversion.

Starting a Conversion
Be sure to read the information under *Settling Time* above before starting a conversion on a new channel.

A conversion is started by writing to register RW2 in the CPLD (U52) (address 0x6000290). Bits 0-7 are not used but should be set to 0.

The 12/16 bit A-D converts in under 10 micro-seconds. Our experience is that it is closer to 8.

Polled Mode
Up to 150 CPU instructions can be executed between the time a conversion is initiated and data read (the number of instructions will vary, depending upon the type). If your process has nothing better to do during the conversion time, you can simply wait for the busy bit to go back high. Review code in demonstration program ain2.c.

Interrupt Mode
During the time a conversion is started to the time it is completed, 150 CPU instructions can theoretically be executed. The number of C instructions varies. Simply checking the status of a flag incrementing a counter runs about 15 loops. In the below program (taken from demonstration program ain1.c), j counts to 15 after starting a conversion.

```
j = 0;
while(iflag)
    j++;
```

This code takes 4 computer instructions, which is 200+ nS/loop. (The reason it doesn't count higher is because branches take up more time and there is some setup time.)

We advise putting in some sort of timeout in case the A-D does not cause an interrupt within a reasonable time. The following fragment bails out after a time.

```
j = 0;
while(iflag)
{
    j++
    if (j = 15)
        iflag = 2;   /* this says data not good */
}
```

Keep in mind it takes two conversions back to back to get a current reading. See *Reading Results* below.

Reading Results
The 12/16-bit A-D converter is of the type that returns the results of the last conversion while sampling and converting the current input. In other words, if you change a channel or gain, it takes *two* converts to get a reading on the new channel, and the result returned is what was present when the first conversion was initiated.

This could be good or bad, depending upon how you are making readings. If you are reading all channels sequentially, and the voltages and gains are nearly the same, it won't make much difference. The conversion results will be one channel off. If you are reading the same channel, you have to remember that the reading is from the last convert command. Random channel selection is worse case as you will have to wait for settling time and perform 2 converts before getting a current reading.

Results are read in two bytes. LSB is at 0x06000298 and MSB is at 0x060002a0. The following code fragment shows how to read and convert results to a number from 0 to 4095 or 65535:

```
iain = *(ctrptr + 0x18);
iain = *(ctrptr + 0x20) * 256 + iain;
```

See ain1.c for example. The 12 bit converter automatically returns 0's for data bits 12 - 15, so the same formula works for both converters.

Conversion Accuracy and Sources of Error
There are a number of sources which affect accuracy of A-D readings. Errors are more pronounced 16 bit converters simply because of the resolution.

The first source of error are the gain amplifiers U5 and U7. Their worst accuracy is at the higher gains. This is shown in the table under *Programming Gain, Bipolar/Unipolar* section above. The values shown under "% Gain" are *typical* and not maximum. Maximums are about 5 to 10 times worse, depending upon the gain and chip. For a 12 bit systems with 4096 counts, a typical high gain error of 0.05% corresponds to about 2 counts at full scale. A 16 bit system with 65536 counts, the typical high gain error of 0.05% results in about 33 counts of error.

Another source of error in the gain amplifiers is temperature. Gain errors range from 0.00035%/°C to 0.0031%/°C at gains of X100 and above. These are *typical* as described by Burr Brown. Maximum is not specified. For the 12 bit converter, these errors are not significant. Sixteen bit converters can expect a 2 count error / °C change at X100 gain and above. An 8°C change is required to make a 1 count error in 12 bit systems.

Signal source impedance can also cause errors. All analog inputs have a 1 Meg ohm resistor to ground. High source impedance will make a small voltage divider at P2 or P3. For 12-bit accuracy, source impedance should be less than 244 ohms. Sixteen bit systems should have less than 15 ohms impedance. Since these resistances are usually constant you can factor them in any conversion equations. If desired you can remove the resistor networks R1 and/or R2 to dramatically improve input impedance (> 50M ohm). Make sure you ground unused inputs.

The A-D chip itself (U9) contributes up to ±3 bits of error. This is not to say you will always get 3 bits of change continuously, just some of the time. Data sampling follows a bell curve. If you take a 1000 samples using a rock solid reference and other circuits, occasionally (0.3% of the time) you will get a few outliers that are 5 or more counts out. (Refer to the references at the end of the chapter to get an explanation of why this is so.)

This type of error also depends upon the input voltage to the A-D. Different voltages "flatten" out the bell curve. For these reasons it's always good to take a few readings just to make sure the data is stable. Use demonstration program ain3.c to see how stable a conversion is.

The number of readings is another source of error (or accuracy). The best accuracy for 16-bit converters is obtained by averaging 64 or more readings. Our experience has shown you can get good readings by converting as low as 8 times. The effective number of bits for one conversion is about 13.5 using the 16-bit converter. Effective number of bits for the 12-bit converter is about 11.3. Averaging a few 12-bit readings provides optimal performance.

Noise is effectively canceled by averaging several readings at any gain. It is almost necessary at higher gains. Running the histogram program, ain3.c, gives you an idea of how noise is distributed at different gains. Our testing showed noise becomes very significant at gains of X80 and above. Lead lengths, filtering, and signal quality become very apparent. With the gain set between X100 to X800, the converter effectively becomes a 10-bit A-D. Averaging 8 readings removes most errors due to noise in 12-Bit systems. Averaging 32 to 64 times in 16 bit systems helps a lot also. We have found averaging much more than this does not really reduce random noise errors.

Be careful of 60 (or 50 hz) power line noise getting induced by lights or other sources. This becomes apparent when amplifying low level signals.

Settling time when switching channels and/or changing gain also has to be considered. If you know the level between two channels is about the same (within 2%) and gain does not change, then you could be safe in converting immediately.

Another factor affecting settling time is the signal source output impedance. A low output impedance (> 1K ohm) is ideal. Many bridge circuits used in pressure and weighing are lower than this. If you measure before the signal has settled at the A-D, you risk getting bad data. A general, overall rule of thumb is to wait 35 micro-seconds after changing channels and gain before converting.

The input circuit has 1 Meg Ohm resistance to ground before it goes into a multiplexer. If you are operating in differential mode, this resistance may be an important factor. Resistors are generally well matched. But differences of 0.1% can reduce common mode rejection significantly.

There are other factors which affect readings. These are input offset voltages of the gain amplifiers at different gains and temperatures, errors induced by the multiplexers U1, U2, U4, and U8, thermocouple effects between wires and connectors, and noise. Input offset voltages *could* induce errors of ±3% of FS or more. For example, at X800 gain, you could read ±260 mV with the inputs grounded. Unfortunately there is no typical as offsets are specified as ± micro-volts. Offsets from the amplifier can be trimmed out by adjusting R3.

The way to obtain highest system accuracy is to calibrate the card in the application. This means saving calibration constants in RAM or Flash EPROM. Calibration constants include offset voltages and gain errors. Fortunately the offset voltage can be accounted for dynamically by grounding an input and taking a reading. Checking gain requires an external reference.

The A-D itself has accuracy errors at any voltage. These are typically ±3 bits.

Noise is probably the most difficult to fix and explain. There are three basic sources of noise: 1) External to the card. 2) On card induced. And 3) inherent noise in the A-D chip. For our purposes here, noise is defined as the random change in output for a constant input.

The wide variety of applications the RPC-400 is useful for many applications and precludes any specific recommendations for curing external noise. The usual precautions of grounding to one point and grounding on one side are the most effective. A shield driver, described below, may be used to reduce noise on low level input signals.

Input impedance is 1 Meg ohm. This is nice for some high resistance devices such as piezoelectric transducers. However, high impedance makes it susceptible to noise pickup through wires. If you are connecting to the card with just a single wire, you will get 6 counts of noise in a 12-bit system. If your response time can afford it, put a cap between the input and ground right at P2 or P3. How much depends. Usually 0.01 to 0.1 mfd takes care of most noise.

Another problem with high impedance sources is the residual capacitance on the board and circuit will load or dump into the signal. If your previous reading was + 3V and the next one is near 0, that 3 volts has to go somewhere. It will discharge into the source. How this will affect readings depends upon the resistance and capacitance of the signal source. This is most noticeable when you apply a voltage to one channel and leave the one next to it open. The adjacent channel may have some reading on it.

The RPC-400 is a 4 layer board, and is designed to minimize noise due to power and grounds. The 5V power layer is completely isolated around the analog section. Analog ground is tied into digital ground at one point on the board, near P4.

Another source of error is the A-D chip itself. This is especially true for the 16-bit systems. There are many formulas and discussions explaining why you don't get 16 bits from a 16 bit converter[1]. The most effective way to get rid of noise is to average it out. Every doubling of samples decreases noise by 1 over the square root of two. Averaging 64 readings reduces noise contributed by the A-D chip by about 1/10. Fortunately, this also works to reduce noise contributed by the MUXes, amplifier, and board layout. When sampling at gains > X100, take more samples then average them. At X800 gain, 1000 samples gives reasonably consistent (±6 counts) results.

Our testing shows that noise is reasonably distributed across ±5 counts for both 12- and 16-bit systems.

Calibration

The 16-bit and, to a lesser extent, 12-bit converters can be calibrated. Calibration can be used to tweak a system for a particular gain or even channel. Systems with a

16-bit converter have circuitry to fine tune the converter. Both have an adjustment to zero out the offset voltage from the gain amplifiers.

You will need a volt meter at least as accurate as you intend the system to be. For 12-bit systems, this is 0.024% and 16-bit it is 0.0015%. Zeroing is done without a voltmeter.

First, remove power to the board. Put a wire jumper from the channel of interest to a ground point. Power up and run ain3.c program. Set sampling for 1000 times, channel of interest + 128 (bipolar) + gain to 1. Adjust R11 for a reading of 8000 under Norm or Avg. R3 most effective when the gain is higher. Press a key to re-enter parameters. Use the same parameters except change the gain to the highest you will use or to 800 (enter 139 at the gain and bipolar prompt). Adjust R3 for a reading of 8000 under Norm or Avg.

The next step is to apply a known voltage at the input. To calibrate at a particular gain, apply a voltage to the input which reaches near full scale or the particular voltage of interest. Example: Gain is X100, so a 50 mv maximum voltage is applied to the channel to measure. At a gain of X1, then a voltage near 5V should be applied. Press a key and re-enter the parameters. Set the gain for uni- or bi-polar input as appropriate for your situation.

Read your input voltage and calculate what the output should be. If your input voltage is 35.037 mv and gain is set to 100, output voltage is supposed to be 3.5037 volts. The number of counts should read 45967 (3.5037/.000076293) or 0xB38F.

Adjust R12 to match the input voltage as figured above.

10 BIT A-D CONVERTER

The 10 bit analog to digital converter is built into the CPU. Detailed information about this converter is found in the *Hardware Manual, Hitachi SH7032/7034* in Section 14, page 415. RPC-400 schematics for this converter are on pages 1 and 3.

Input impedance is 100K ohm to ground. Conversion time is approximately 6.7 micro-seconds.

Input voltage range is nominally 0 - 5V. By changing the reference input at J1-19, this range can be lowered.

Accuracy of this converter, in the 0 - 5V range, tracks the 5V supply. Thus, if your devices use the 5V supply also, conversions will be ratiometric. For increased accuracy, connect the 2.5V reference output at J1-17 to J1-19. This limits input voltages to 0 - 2.5V. An external reference can be applied to J1-19. Range must be between 0 and 5V.

Reference Input
The reference for the converter goes through a 100K resistor to + 5V digital supply. Thus, output accuracy is only as good as the 5V supply. A precise 2.5V reference is available at J1-17. By tying P1-17 to P1-19, the input range will be 0 to 2.5 volts.

Converting Inputs
Refer to sample program in the AIN4 directory to convert all channels. Results from all 8 channels are printed to the screen.

External triggers are also possible. A low going pulse for 50 nS minimum starts a conversion on a selected channel. An interrupt can be generated at the end of a conversion.

SCALING MEASUREMENTS

Inputs are converted to "real numbers" by scaling them. Demo programs return values from 0 to 1023 (10 bit), 0 to 4095 (12 bit), or 0 to 65535 (16 bit). To change these numbers into something more meaningful, use the following formula:

var = k * ain(*n*);

ain(*n*) returns the reading on channel *n*. 'k' is the scaling constant. k is obtained by diving the highest number in the range by the maximum ain count (1023, 4095 or 65535).

NOTE: At the time this manual was written, the GNU C compiler did not support floating point constants. Review the software section of this manual for additional information on making floating point constants.

Example 1: To measure the results of an A/D conversion in volts and the voltage range is 0 to 5V, divide 5 by 1023 for the 10 bit converter, 4095 for 12-bit systems, or 65535 for 16-bit to obtain k.

10-bit	12-bit	16-bit

k = 5/1023 k = 5 / 4095 k = 5 / 65535
k = 0.00195 k = 0.001221 k = 0.000076295

Example 2: Measure a 0 to 200 PSI pressure transducer with a 0 to 50 mv output. Set the gain for X100. This makes the measurement range 0 to 5V. Next, divide 200 by either 4095 or 65535 to obtain the constant k.

12-bit	16-bit
k = 200 / 4095	k = 200 / 65535
k = 0.0488	k = 0.0030518

Measuring 4-20 mA current loops
Current loops are a convenient way to transmit a value and still assure the integrity of a signal. If the line should break, 0 volts (or nearly so) is returned.

A 4-20 mA current loop is converted to 1 - 5V by placing a 250 ohm resistor across the input of the channel to ground. Current loop readings are converted to engineering units by scaling as described earlier. Since the measurement range is 1 to 5V, the count range is reduced by 20% to 3276 for 12-bit and 54428 for 16-bit systems.

If pressure were measured as in example 2:

12-bit	16-bit
k = 200/3276	k = 200/54428
k = 0.6105	k = 0.0367459

There is one additional factor. Since the lowest value read is 1 V, this offset is subtracted from all readings. A 1V offset is 1/5 of 4095 or 65535 counts, or 819 for 12-bit systems and 13107 for 16-bit. The code then becomes:

```
a = k * (ain(n) - 819);     /* 12 bit     */
a = k * (ain(n) - 13107); /* 16 bit     */
```

Note that if the current loop line breaks, a negative value is returned. The function *ain(n)* returns a reading for channel *n*.

DEMO PROGRAMS

There are 4 demo programs, AIN1 - AIN4. AIN1.C operates in interrupt mode at fixed gain. AIN2.C operates in polled mode. All channels are displayed. in AIN1.C and AIN2.C

AIN3.C allows you to select the channel, gain, mode, and number of samples. It also displays the average and deviation from the first reading to determine stability in a particular setting. For 16 bit systems, this is valuable in determining the number of samples for a good reading.

AIN4.C displays the 10 bit converter at J1.

Table 10-3 connector Pin out - J1

Pin #	Function
1	Analog input channel 0
2	Ground
3	Analog input channel 1
4	Ground
5	Analog input channel 2
6	Ground
7	Analog input channel 3
8	Ground
9	Analog input channel 4
10	Ground
11	Analog input channel 5
12	Ground
13	Analog input channel 6
14	Ground
15	Analog input channel 7
16	Ground
17	2.5 Volt reference output
18	Ground
19	10 bit converter ref. input
20	External A-D trigger

MEMORY MAP A-D

Address	Function
0x6000290	RR2 - A-D status, interrupt source
0x6000290	RW2 - Start conversion
0x6000298	RR3 - Read LSB of 12/16 bit conversion.
0x6000298	RW3 - Channel select, SE/Diff mode, int enable.
0x60002A0	RR4 - Read MSB of 12/16 bit conversion.
0x60002A0	RW4 - gain, unipolar/bipolar select

REFERENCES

Burr-Brown Application Bulletins AB-095, AB-098,
ADS7808, 7809 data sheets.

INTRODUCTION

A programmable watchdog timer is built into the CPU. It performs a partial system reset on overflow, and acts like a non maskable interrupt. On overflow, many, but not all, internal registers are reset. No external hardware is affected.

The watchdog timer can be configured as an interval timer, generating an interrupt. When configured as a watchdog timer, the WDTOVF signal goes low at J13-19. This signal can be tied to the RST line at J12-7 to force a push button reset. However, when this is done the WOVF flag in the RSTCSR register is cleared. There will be no direct way to detect if there was a watchdog timer reset.

The watchdog timer is reset by periodically writing to a special register. This register is different to access as specific data is written to initialize and reset it. Reset times are programmable from about 25 µSec to 105 mS.

See *Section 12, Watchdog Timer* in the *SH7032 Hardware Manual* for programming and technical information. Review the code in the WDT directory for an operating example.

INTRODUCTION

The RPC-400 has five interrupts available between J12 and J13. These are: NMI, IRQ4, IRQ5, IRQ6 and IRQ7. IRQ7 goes to J12 and J13. The rest are exclusive to J13. All pins are TTL level compatible.

On power up, the IRQ pins are programmed as inputs. To use them as interrupts, you must modify PCF_PBCR1. This register is described in the SH7032 hardware manual starting on page 444. The demonstration program in directory IRQ6 shows the initialization sequence and operation of IRQ 6.

INTERRUPT RESPONSE TIME

The program IRQ6.C in the IRQ6 subdirectory can indicate how long it takes to respond to an interrupt. As written, it is about 3 micro-seconds from the time the interrupt is created to the time it is cleared.

Table 12-1 Connector pin out - J13

Pin #	Description	Interrupt type or alternate function
1	PB15	IRQ7
2	PA15	IRQ3
3	PB14	IRQ6
4	PB13	IRQ5
5	PB12	IRQ4
6	NMI	NMI
7	Ground	
8	PB7	Timing/clock
9	PB6	Timing/clock
10	Ground	
11	PB5	Timing/capture
12	PB4	Timing/capture
13	Ground	
14	PB3	Timing/capture
15	Ground	
16	PB2	Timing/capture
17	Ground	
18	PB1	Timing/capture
19	Watchdog overflow	
20	PB0	Timing/capture

See SH7032 hardware manual sections *"Pin Function Controller"* and *"Parallel I/O Ports"* for programming information.

INTRODUCTION

There are a variety of counter/timing capabilities on the RPC-400. Four inputs are available at P7 and P8. These programmable inputs up/down count pulses to 20 Mhz, read quadrature inputs, and measure various kinds of pulse and period widths to a resolution of 203.45 nS. An interrupt can be generated on a carry, borrow, or measurement complete for any counter.

Counting at P7 and P8 is done through a LS 7266 counter. Information about this chip is found in Appendix B.

The second group of pulse timing is at GPIO port J13. This port can both output pulses and measure inputs.

Both ports are discussed in this section.

P7 AND P8 OVERVIEW

Signals from P7 and P8 go to one of two LSI LS7266 24-bit dual-axis quadrature counters. The data sheet for this part is in Appendix B. By combining a gate array IC on board (U52), a number of pulse measurements are possible. Additionally, interrupts may be generated under certain conditions.

This section describes how to program the chips for some of the counting modes. In general, when a function for a counter is programmed for one function, the other counters are not affected.

The schematic for the counter input section is on page 8. Demonstration programs are under 'PWM*x*' subdirectories.

Definitions
Counters are numbered 0 through 3. The table below correlates a counter to a chip 'U' number and axis in that chip.

Counter	Chip U #	Axis
0	53	X
1	53	Y
2	48	X
3	48	Y

The letters 'pwm', in lower case, mean pulse width measurement.

The term 'no meaningful data is returned' is used to describe some registers requiring a read to work. Data returned is simply the status of a floating bus. No information is output by the CPLD.

High voltage input
Counter 0 inputs A0 and B0 on P7 can directly interface to proximity sensors or other devices. Input range is ±15 volts. TTL devices can also directly interface to these inputs.

These inputs invert the signal to the counter. Thus a high at the B0 input causes a down count in normal counting mode. Leaving B0 open, grounding, or applying a negative voltage causes an up count in normal counting mode.

The buffer is a 1489 type. This is a standard RS-232 interface chip. The switching threshold is set by resistors R40 and R41. When the input goes above approximately 3 volts, the output at pins 8 or 11 at U44 will go low. When the input then goes low to approximately 2 volts, the output will switch back high. This is a hysteresis of about 1 volt. Threshold levels are adjusted by changing R40 and/or R41. Lower value resistors raise the threshold level.

Quadrature filter
The inputs from a quadrature encoder are digitally filtered. The filter frequency is programmable for each channel in the LS 7266 chip using the PSC register. The input filter clock ($\mathbf{f}_{FCK}$) is 2.4585 Mhz.

The final filter clock is determined by the formula:

$$f_{FCKn} = 2458500 / (PSC + 1)$$

Where: PSC = 0 to 0xff

The final filter clock should be 5 times faster than the maximum frequency of the quadrature clock. Any pulses or noise that are faster than this are filtered out.

CONTROL REGISTERS

A CPLD IC (U52) controls interrupts, pwm modes, and the A-D. The following is the list of read and write registers in this chip.

Table 13-1 Register descriptions

Ref.	Type	Function
RW0	Write	Counter 2 IRQ and pwm control
RR0	Read	Clear IRQ latches for counter 2
RW1	Write	Counter 3 IRQ and pwm control
RR1	Read	Clear IRQ latches for counter 3
RW2	Write	A-D convert
RR2	Read	A-D status, interrupt source
RW3	Write	Analog channel select
RR3	Read	Lower 8 bits of A-D conversion
RW4	Write	Set A-D gain, start conversion
RR4	Read	Upper 4/8 bits of A-D conversion
RW5	Write	Start counter 2 pwm
RR5	Read	Unused
RW6	Write	Start counter 3 pwm
RR6	Read	Counter 0 & 1 IRQ status
RW9	Write	Enable counter 0 & 1 IRQ
RR9	Read	Clear counter 0 IRQ latches
RR10	Read	Clear counter 1 IRQ latches

Addresses for above registers and the two counter IC's (U48 and U53) are:

Table 13-2 Address location

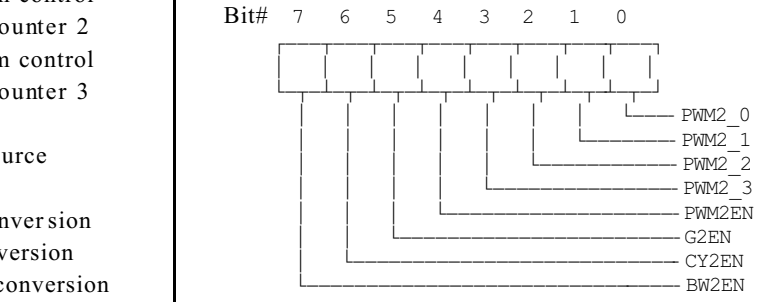
Register	Address	Base offset
RR0, RW0	0x6000280	0
RR1, RW1	0x6000288	0x08
RR2, RW2	0x6000290	0x10
RR3, RW3	0x6000298	0x18
RR4, RW4	0x60002a0	0x20
RR5, RW5	0x60002a8	0x28
RR6, RW6	0x60002b0	0x30
U53 X data	0x60002b8	0x38
U53 X command	0x60002ba	0x3a
U53 Y data	0x60002bc	0x3c
U53 Y command	0x60002be	0x3e
U48 X data	0x60002c0	0x40
U48 X command	0x60002c2	0x42
U48 Y data	0x60002c4	0x44
U48 Y command	0x60002c6	0x46
RR9, RW9	0x60002c8	0x48
RR10	0x60002d0	0x50

Most registers are set to 0 on power up or push button reset. Some registers are cleared by reading.

The basic function of each register is described below. More detail is provided later as appropriate to the register.

RW0

Table 13-3 Counter 2 IRQ enable, counting mode



Where:

BW2EN	=	Borrow, counter 2 (U48, X axis). Enables interrupts when borrow line goes low.
CY2EN	=	Carry, counter 2 (U48, X axis). Enables interrupts when carry line goes low.
G2EN	=	Gate, counter 2 (U48, X axis) (used for pwm)
PWM2EN	=	Enable pwm, counter 2 (U48, X axis)
PWM2_3	=	pwm mode bit 3
PWM2_2	=	pwm mode bit 2
PWM2_1	=	pwm mode bit 1
PWM2_0	=	pwm mode bit 0

Bits 6 and 7 enable an interrupt when counter 2's carry or borrow bit goes low. Pulses from the carry and borrow lines can be short enough the CPU can miss them. The carry and borrow signals are latched when the respective line goes low.

Bit 5 enables a pwm counting mode interrupt

Bit 4 enables pwm counting.

Bits 0-3 define the pwm counting mode.

All bits are 0 upon push button or power up reset.

RR0

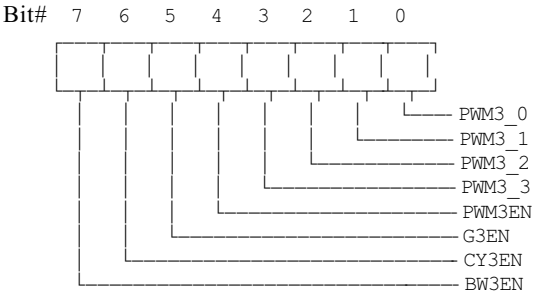
Clears all counter 2 IRQ latches caused by a carry,

borrow, and/or gate for counter 2 by reading this port. No useful data is returned. RW0 is not affected by a read.

A latch is set to 1 if enabled in RW0 and the appropriate counter line (carry, borrow, or gate) goes low momentarily.

RW1

Table 13-4 Counter 3 IRQ enable, counting mode



Where:

BW3EN	=	Borrow, counter 3 (U48, Y axis). Enables interrupt when borrow line goes low.
CY3EN	=	Carry, counter 3 (U48, Y axis). Enables interrupt when carry line goes low.
G3EN	=	Gate, counter 3 (U48, Y axis) (used for pwm)
PWM3EN	=	Enable pwm, counter 2 (U48, Y axis)
PWM3	=	pwm mode bit 3
PWM2	=	pwm mode bit 2
PWM1	=	pwm mode bit 1
PWM0	=	pwm mode bit 0

Bits 6 and 7 enable an interrupt when counter 3's carry or borrow bit goes low. Pulses from the carry and borrow lines can be short enough the CPU can miss them. The carry and borrow signals are latched when the respective line goes low.

Bit 5 enables an interrupt in the pwm counting modes.

Bit 4 enables pwm counting.

Bits 0-3 define the pwm counting mode.

All bits are 0 upon power up or push-button reset.

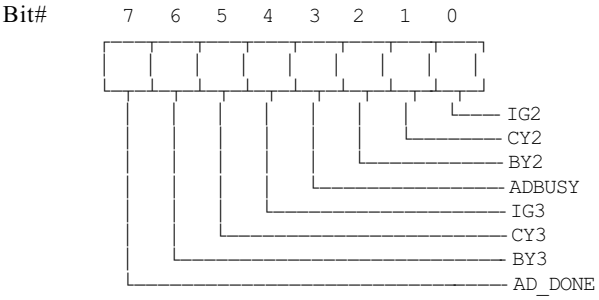
RR1

Clears all counter 3 IRQ latches caused by a carry, borrow, and/or gate by reading this port. No meaningful data is returned. Register RW1 is not affected by a read.

A latch is set to 1 if enabled in RW0 and the appropriate line (carry, borrow, or gate) goes low momentarily.

RR2

Table 13-5 Read counter 2 and 3 IRQ status.



Where:

IG2	=	Counter 2 gate (for pwm)
CY2	=	Counter 2 carry
BY2	=	Counter 2 borrow
ADBUSY	=	Status of A-D conversion only
IG3	=	Counter 3 gate (for pwm)
CY3	=	Counter 3 carry
BY3	=	Counter 3 borrow
AD_DONE	=	A-D converter (U9) is done

If a bit is high, then that line caused the interrupt. (ADBUSY is the A-D status only. It will not generate an interrupt. AD_DONE does that.)

RW2

Starts conversion to 12/16 bit A-D. Set all bits to 0 for future compatibility.

RR3

Read lower 8 data bits of 12/16 bit A-D conversion. Do not read this data while a conversion is in process.

RW3

Select analog channel. Refer to Chapter 10, *ANALOG INPUT* for more information.

RR4

Read upper 4/8 data bits of 12/16 bit A-D conversion.

RW4

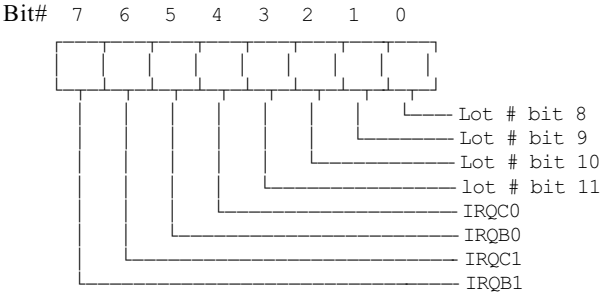
Set A-D gain, start conversion. See Chapter 10, *ANALOG INPUT* for more information.

RR5
Read lower 8 bits of lot code.

RW5
Reset and start counter 2 pulse width measurements. Writing a 0x60 resets pwm. Writing a 0x90 enables pwm.

RR6
Reads IRQ status for counters 0 & 1. The MSB of the lot code. This register is read, along with RR2, to determine the source of an interrupt.

Table 13-6 Counter IRQ status



Where:

IRQC0 = Counter 0 carry interrupt. Set if carry signal on U53, X counter, went low.

IRQB0 = Counter 0 borrow interrupt. Set if borrow signal on U53, X counter, went low.

IRQC1 = Counter 1 carry interrupt. Set if carry signal on U53, Y counter, went low.

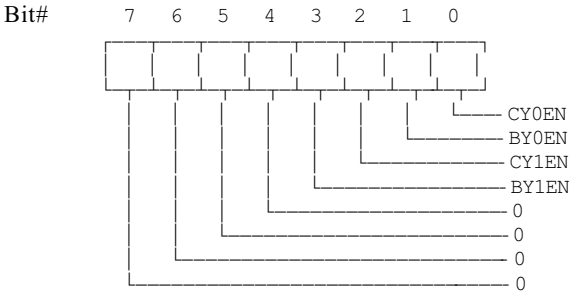
IRQB1 = Counter 0 borrow interrupt. Set if borrow signal on U53, Y counter, went low.

Lot bits 8-11 are the most significant nibble of the lot number. The lot number can be used to ID a board.

RW6
Reset and start counter 3 pulse width measurements. Writing a 0x60 resets pwm. Writing a 0x90 enables pwm.

RR9
Clear counter 0 IRQ latches. A read to this register clears carry and borrow IRQ. No meaningful data is returned.

RW9
Table 13-7 Enable counter 0 and 1 IRQ's



Where:

CY0EN = Carry, counter 0 (U53, X axis)

BY0EN = Borrow, counter 0 (U53, X axis)

CY1EN = Carry, counter 1 (U53, Y axis)

BY1EN = Borrow, counter 1 (U53, Y axis)

When a bit is set to 1, the interrupt for that line is enabled when it goes low.

Bits 4-7 are not used. However, they should be set to 0 in case of future changes.

RR10
Clear counter 1 IRQ latches. A read to this register clears carry and borrow IRQ. No meaningful data is returned.

COUNTER INTERRUPTS

All counter and A-D interrupts are 'ORed' and go to CPU INT 0.

Interrupt sources are determined by reading one or two registers. Interrupts are enabled and cleared by writing and reading to/from various registers. These registers are described below.

The 'Bits' column shows the bit of interest to generate an interrupt when either the carry or borrow line from the counter goes low. In all cases, an interrupt is enabled when a bit is set to a '1'. They are disabled on reset or by writing a '0' at any time.

All counter interrupts are latched on a 40 nS or longer pulse. During an interrupt service routine, the latch saving the interrupt must be cleared. This is done by reading the port where the interrupt was first enabled. So, a read at address 0x6000280 clears carry, borrow, and gate flags for counter 1. No meaningful data is returned on these reads.

All Counter interrupt sources share a same common output (IRQ 0 on the CPU). To determine the source, read register RR2 and, if necessary, RR6. A high indicates it caused or was one of the ones that caused an interrupt.

Interrupt source	Type	Read at register	Bit no.
Counter 0	Carry	RR6	4
Counter 0	Borrow	RR6	5
Counter 1	Carry	RR6	6
Counter 1	Borrow	RR6	7
Counter 2	Carry	RR2	1
Counter 2	Borrow	RR2	2
Counter 2	Gate	RR2	0
Counter 3	Carry	RR2	5
Counter 3	Borrow	RR2	6
Counter 3	Gate	RR2	4
A/D Done		RR2	7

PROGRAMMING NOTE:

It is possible to miss a new interrupt during the process of clearing IRQ registers. The way to avoid this is to check all counters that can generate interrupts and determine if one is imminent. This is done by reading the counts and determining if it is near a limit. There may be other methods suitable to your application. If A-D conversion is possible, set a global flag (int) before conversion to 1 to indicate this. The program can wait in the interrupt service routine until conversion is complete (> 10 micro-seconds).

You can operate in pwm mode without using interrupts. However, ALL interrupts for INT0 would have to be disabled in the CPU by forcing bits 8 and 9 to 0 in PACR1. Programming remains the same. You would have to poll the appropriate register to determine what source was active.

COUNTING/MEASUREMENT MODES

There are several possible counting and time interval measurement modes at counters 2 and 3. Counters 0 and 1 only perform the modes allowed in the LS7266 chip. They cannot measure any pulse widths.

Basic
The basic counting mode refers to those modes allowed in the LS7266 chip. These include up/down and quadrature. Refer to the LS7266 data in Appendix B for more information. The C driver/example disk show how to program and access this chip for its counting modes. Look under PWM0 - 3 and CNTR0, 1 directories.

All counters can operate in the basic mode. No CPLD registers need be modified. To generate an interrupt from the counter, modify the appropriate bit(s) in the CPLD register(s).

Counter	Register
0	RW9
1	RW9
2	RW0
3	RW1

pwm Modes
There are 4 pulse width measurement modes. These modes are programmable in registers RW0 and RW1. All pulses are triggered on an edge on A2 from counter 2 or A3 from counter 3. Pulse width resolution is 203.4505 ns

pwm #	Register data	Measurement Mode
0	0x10	Falling edge to next falling edge.
1	0x11	Rising edge to next rising edge.
2	0x12	Falling edge to next rising edge.
3	0x13	Rising edge to next falling edge.

These pwm modes are programmed using RW0 and RW1. Both are the same except for the counter. Gate, carry, and borrow interrupts are programmed by 'oring' in the appropriate bit(s) to the 'Register data' above as desired. In actual use, only two interrupts are useful. The Gate bit (GxEN) causes an interrupt when a measurement is complete. Carry (CYxEN) is used to indicate an overflow (approximately 3.3 seconds), or a very long pulse condition. Counting continues even when there is an overflow.

Operation is described below. See program examples pwm0.c through pwm3.c in directories PWM0 - PWM3 for examples.

pwm Operation
The counter is programmed in simple up/down mode and is reset to 0. The appropriate register (RW0 for counter 2, RW1 for counter 3) is programmed (including interrupts if desired). Bit 4 in RW0 and/or RW1 must be high to use pwm counting modes. Making bit 4 a 0 disables any pwm modes.

Although not required, the counter logic should be reset by writing a 0x60 to RW5 (counter 2) or RW6 (counter 3) as appropriate. To arm the logic, write a 0x90 to RW5 or RW6 as appropriate. The logic is now ready to time a pulse as soon as the appropriate edge comes along.

You can disable pulse measurement by writing a 0x60 to RW5 or RW6. This is useful should you decide to disarm the timing logic.

When RW0 or RW1 is programmed for pulse width measurements, the input signal A2 or A3 from P8 is routed to internal logic in U52. A 4.9152 Mhz clock is fed to input A on the counter instead. When the input signal at A2 or A3 changes to the appropriate state, a gate enables counting. When the signal again changes to the appropriate state, the gate to the counter goes low, disables counting, and generates an interrupt (if enabled).

COUNTER DEMO PROGRAMS

The following is a list of counting demonstration programs using the LS7266 counter chip. Program names are in the same name subdirectories, under RPC400 directory.

All demonstration program share common operating characteristics. They are all interrupt driven. Interrupts can be disabled by resetting/not setting the appropriate bit(s) in RW0, 2, and 6. The current count is displayed on COM1. You may wish to change this to COM0. If you do, you cannot effectively use the debugger (GDB) but must use HINT instead. See sections 16 and 17 for more information.

Program name	Description
CNTR1.C	Counter 0 for simple up/down Counter 1 for quadrature mode
CNTR2.C	Counter 2 for simple up/down Counter 3 for quadrature mode
PWM0.C	Counters 2 & 3 for negative edge pulse width measurement. Interrupt generated on overflow.
PWM1.C	Counters 2 and 3 for positive edge pulse width measurement. Interrupt generated on overflow.
PWM2.C	Counters 2 and 3 measure pulse width starting on negative edge and ending on positive. Interrupt generated on overflow.
PWM3.C	Counters 2 and 3 measure pulse width starting on positive edge and end on negative. Interrupt generated on overflow.

INTRODUCTION

Four optional analog output channels are available. Outputs are available as voltage (D/A) or 4-20 mA current. D/A outputs are 12 bits and have a settling time of less than 10 micro-seconds for full resolution. D/A's can supply several milli-amps of current at a voltage.

All outputs are at P4. Current loop outputs use a D/A. Consequently, voltage output is not usable when current loop output is used.

Voltage outputs can be configured to operate in one of three voltage ranges. Voltage ranges are jumpered in hardware using W2-W6. Analog outputs are on schematic page 2.

POWER CONSIDERATIONS

The RPC-400 operates with a wide range of analog input voltages. These voltages are shared with analog inputs and RS-232 output. They are marked as + 15 and -15 on P1. Specifications call for ±12 to ±16 volts. However, analog outputs may not work properly if certain minimums are not supplied.

The current loop or D/A output at 0-10V require a minimum of + 13 volts to realize the full range. The development system power supply may not deliver this voltage. Current loop power, designated as 'CLP' on P1, should be a minimum of + 13V if the load is 10V. Consideration must be made for cable length also.

If you are not using current outputs and your D/A output is not 0-10V, then you can supply + 12V or less power. The negative analog supply is essentially not a factor so long as it supplies at least -8V.

INSTALLING A CHANNEL

A MAX508 or AD7248 converter (P/N 1554) is used as the D/A. An AD694 converts voltage to current. A current loop kit is available as P/N 1678. The table below shows the corresponding IC(s) to channel.

Channel		D/A	Current
Voltage	Current		
VO0	CL0	U11	U12
VO1	CL1	U13	U14
VO2	CL2	U15	U16
VO3	CL3	U17	U18

Current loop outputs require both IC's. Voltage only outputs require only the corresponding D/A chip.

Before installing a chip, ground yourself and the board and make sure power is removed. Align pin 1 on the chip to pin 1 on the socket. Pin 1 on the IC is marked with a dot or notch.

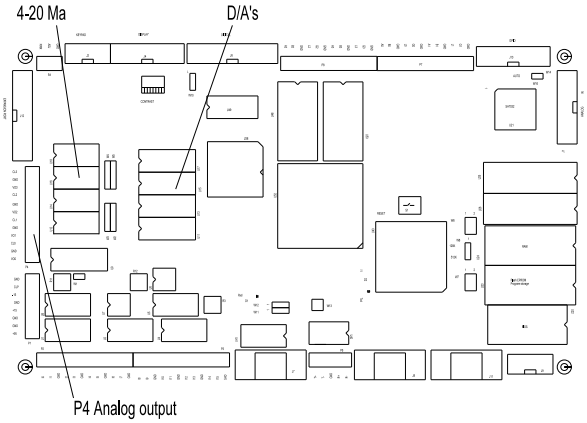


Figure 14-1 Analog Output

Voltage Output
Voltage ranges are jumpered in hardware using W2 - W5. Jumper positions for a voltage are the same for all.

Voltage Range	Jumper Position
0 - 5	4 - 5
0 - 10	1 - 2
±5V	3 - 4

Set jumper to 0 - 10 volt output (1 - 2) for a 4 - 20 mA current loop.

To output a voltage, a number from 0 to 4095 is sent in two bytes. The lower 8 bits are written first, followed by the upper 4 bits. The D/A changes value when the upper 4 bits are written to. The output is updated when the upper 4 bits are written. The demonstration disk shows how a ramp is generated. Refer to program aot1.c in subdirectory aot1.

Response Time
The time it takes to change from one voltage to another

depends upon the delta V. For a 0-10 and 10-0 V delta V, response time is about 3.5 micro-seconds.

Output Noise
Analog outputs are isolated from the digital bus through a buffer (U19). The greatest noise is when any of the analog channels is accessed. The amplitude of this noise is about 1 V P-P at 10 volt output. It is slightly less at lower output voltage. Duration of each pulse is about 200 nS, with each converter generating two pulses every change. Putting capacitors on the output could actually increase noise and ringing, depending upon the output voltage.

CURRENT LOOP

Current loop and voltage output IC's must be installed for a particular channel in order to work. Refer to the table above. The D/A must be configured for 0-10 volt output in order to work properly.

Current loop supply is on P1 and is designated as CLP. Apply + 5 to + 36 volts to this connector. Most current loops terminate into either 500 ohm (10 V output) or 250 ohm (5 V output) loads. Your supply voltage should be at least 3 volts above the load voltage.

Current output is proportional to the voltage from the D/A. An output voltage of 0 will output 4 mA while 10 volts outputs 20 mA.

Response Time
Rise and fall times vary with the drive voltage, delta voltage change, and current loop supply voltage. For a full scale change, rise time is about 70 micro-seconds and fall time is about 30 micro-seconds. A larger current loop supply voltage increases rise and fall times.

MEMORY MAP - D/A

The following are addresses used to access the D/A's. Refer to the demonstration disk for driver example. Function name is aot1.c in the aot1 subdirectory.

Channel	Address	Program range
V0/CL0	0x600300	LSB
	0x600302	MSB
V1/CL1	0x600304	LSB
	0x600306	MSB
V2/CL2	0x600308	LSB
	0x60030A	MSB
V3/CL3	0x60030C	LSB
	0x60030E	MSB

INTRODUCTION

The expansion bus at J12 provides a simple way to add digital, analog, or other types of I/O. Data and address lines are buffered from the CPU. They are not buffered from other devices considered I/O. See schematic page 4.

Port expansion read, write, and chip select timing about 100 nS low. This time is changed through CPU register WCR3 by changing the number of wait states.

Display power connector P5 goes to both J12 and the display connector J4. This is because the graphics display may be used in either connector.

See the program 'graph2.c' in the GRAPH2 directory for a programming example. This program reads and writes to the graphics display.

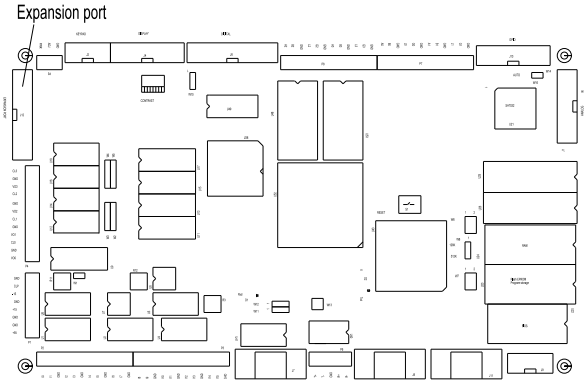


Figure 15-1 Expansion Bus

Use the following table for pin outs.

Pin	Function
1	+ 5V power
2	Ground
3	Data I/O 4
4	Ground
5	Data I/O 6
6	Data I/O 5
7	Reset (active low)
8	Write (active low)
9	Read (active low)
10	Data I/O 7
11	Data I/O 1
12	Data I/O 0
13	Data I/O 3
14	Data I/O 2
15	Port select (active low)
16	Address 1
17	+ 5V
18	Display adjust
19	Display power
20	Ground
21	Address 4
22	Address 3
23	Address 2
24	IRQ 7
25	-15V supply
26	+ 15V supply

The address range for J12 is 0x6000200 - 0x600021f. Note that address A0 is not brought out, so all addresses increments by 2.

INTRODUCTION

This section deals with several small programming issues that, individually, do not warrant an entire section.

INITIALIZATION

A number of devices are not initialized by the BIOS at reset. Some of these can cause the board to draw excess current and potentially unreliable operation. The sections below describe what to do if you are NOT using a particular function.

Operator interface J2, J4
If you are not using the display and/or keypad, then set unused lines to outputs. This is done by writing a value of 128 to the 82C55 configuration register at address 0x600186. If you are using only some of the lines for inputs, be sure to tie unused lines to ground.

COM0
COM0 is initialized by the BIOS when auto run is not enabled. Set up this port if you intend on using this port for your application.

CPU
A number of CPU registers are not altered on push button reset. This list is extensive. The programmer should either extensively review the program and verify the status of CPU registers on reset and power up and cycle power to the board to ensure best program reliability.

Using C examples

The C examples are provided to show how a feature or function works. You can use these programs and put them in your own. Be careful of modifications, however. Some things were done in a particular way because it got it working. The C language is such that you may think it is doing one thing but it is actually doing something else.

Lengths of data types

The GNU compiler uses different number of bytes, depending upon the data type. The following table shows the number for each type.

Type	Bytes
char	1
short	2
int	4
long	4
float	4
double long	8

WRITING YOUR OWN PROGRAMS

There are specific files that you probably will have to have in order to make your own program. The term "probably" is used because there are always exceptions. Review the following is a list of files. Some are mandatory and others are optional, as stated.

File	Description
vects.o	This is the interrupt vector table. At a minimum, the starting address and stack are here. Used by bios to auto-run. Source code is vects.c. Mandatory.
start.o	This changes the vector base from 0 to 0x9002000. Used for interrupts routines. Source code is start.s. Mandatory with interrupts.
inlines.o	Sets the interrupt mask. Use when enabling interrupts. Source code is inlines.s Optional.

Of course you will need to provide your own program. A good way to start is to use one of the examples as a basis then build upon it.

Programs were designed to show how to do something and to prove a particular feature works. They were not designed to be compact.

If your program gets long, you may want to break it up. If you have several programs, you can either include them as shown in the RPC400.CMD file or make up a library.

Modifying VECTS.C

VECTS.C describes the interrupt vector table. The program START.S sets the vector base to 0x9002000. Memory from 0x9002000 to 0x90023FF is used for vectors and reserve area by the CPU. In order to use interrupts and save and load programs, a version of this file must be included.

Many directories have a version of vects.c. The directories that do are a focused example that uses a specific interrupt(s). You can combine interrupts from different examples into your program simply by adding it to the table. See vects.c in the 'user' directory for an example. It contains a large number of interrupts.

Compile vects.c using gcc in your program directory as follows:

```
gcc -c -g vects.c
```

Include this file in the linker command (RPC400.CMD).

400IO LIBRARY

Programs in the 400IO library and directory have general purpose serial initialization, get, put, and print routines. Many demonstration programs include this library as a matter of course. However, including it does increase program size. You may wish to extract routines of interest and make up your own library.

LINKING PROGRAMS

Programs are linked differently depending upon the debugging method. The fastest and easiest way to link files is through a command file. Command files in the example directories have a .CMD extension.

The first line or two in the command file determine the output format. To output a linked file in a format for the debugger (gdb), the first line should be:

```
OUTPUT(rpc400.x)
```

The command 'OUTPUT' must be in all caps.

To output a linked file in a format for HINT, the first two lines should be:

```
OUTPUT(rpc400.sr)
OUTPUT_FORMAT(srec)
```

A complete link file for the GDB debugger is found in the PWM1 directory. A link command file for HINT (and output in S record format) may be found in the GRAPH1 directory. Other directories have different outputs. Make sure you review the command file to determine which one is used. The batch file will also indicate how the command file is formatted. If the program HINT COM1 is called out, then the output is in S record format. If GDB is called out, then the output is in a .X format.

Other programs and libraries may be linked in to modularize programming. Some files, such as 'vects.o' are included in the link as the program includes interrupts.

Libraries 'libc.a' and 'libgcc.a' will frequently need to be included. Pay attention to the order files are linked. Libraries are linked last while individual files are linked first. The 'rpc400.a' library is linked before the general libraries.

STRINGS AND STRING HANDLING

A number of string handling routines are provided in the GNU support and math libraries. Unfortunately, some of these libraries do not work or greatly increase the size of the program. We have provided some work arrounds. These functions are in the 400IO library.

Generally, the floating point conversions do not work well or at all. These are the string to floating point or floating pint to string converters such as `atof` or `gvcvt`. The `printf0` program in the 400IO directory shows how to convert numbers to ASCII printables.

Some conversion programs are sprinkled about in several demo programs such as the LCD440 directory. A `ltoa` converts a long to a string while `ltod` converts a number to a decimal portion with leading 0's, such as .035.

USING GNU COMPILER - A SHORT GUIDE

Writing programs for the RPC-400 is relatively straight forward. You write a program, compile it, link it, and download it. It's the options and variations that cause confusion.

The program examples provided are all stand alone. You could take them 'as is', write them to flash, and auto-execute them on power up or reset. There are

really only a few compiler and linker options you may want to consider. Debugging is covered in *SECTION 17, DEBUGGING PROGRAMS*.

The 'MAKE' program is used for a few examples. This is useful if you break up your program into many smaller modules. You can read the manual, included with the development system, or use the template in the 'CMON400' directory.

Compiler and linker options are discussed below. The major difference is how you intend to do your debugging. If you use the GDB debugger, you set certain switches. If not, you can ignore them.

Using comments
You cannot nest comments. The comment terminator '*/' cancels out all previous '/*'.

Compiler
Compiling takes the following command line form:

```
gcc -c -g -O fname.c
```

The compiler has a few command lines to consider. Refer to the *Using GNU CC* manual for more information.

-c Compile or assemble but do not link. Since all examples use the linker, this switch is necessary.

-g Produce debugging information. Needed with gdb. Produces line number info with -S option.

-O Optimize code by tying to reduce code size and execution time. Additional options 0 - 2 perform different optimization. Refer to manual for more info.

-S Stop, do not assemble. Output is in assembly and goes to a file with a .S extension. This is useful if you want to see what the CPU is doing. Use with the -g option to get line numbers in the assembly code.

The GNU CC manual is for a number of different types of CPU's. References to other type should be ignored. Just pay attention to Hitachi SH-1. Generally, it does not have a lot of options.

Linker
The linker, LD, can operate using a command file (.CMD extension) or command line. Command file is good for linking a number of files. The command line is good for short programs and for the 'MAKE' program. The command line format is used primarily in the CMON400 directory.

Look at a command file (all of them are RPC400.CMD) to view the general structure. It consists of an output format, input files and memory structure. All RPC400.CMD files are similar. Differences are in the linked files and output format. HINT uses a S record output. Look at the command file in the COMM21 directory for a .SR output.

GDB uses something with a .X output. Look at the command file in the AIN1 directory for GDB output.

To add a file for linking, simply input it as part of the list.

FLOATING POINT

The GNU C compiler does support 32- and 64-bit IEEE single- and double-precision form floating point. The compiler does have a problem with floating point constants, however.

This is solved by simply "creating" a constant. Integers (and longs) are 32 bits wide, which makes numbers of ±2 billion possible. Simply casting an integer into floating point number is sufficient.

```
float   fl;
int     i = 12207;
int     j = 10000000;

fl = (float) i / (float) j;
```

The above example puts the number 0.0012207 into the variable fl.

INTRODUCTION

There are two methods you can use to debug your program. One is to use the GDB debugger supplied with the development system. Second is to use HINT, a terminal program with program download capability.

An advantage to using HINT for debugging is you can use the programming port as part of your program. HINT also gets you closer to the hardware and software. HINT is necessary to save programs to flash (unless you make saving a part of your program).

The GDB program is very convenient for debugging C programs. You can single step through C programs and view variables. If your program gets lost, GDB is not very good about recovering from errors.

One note about using GDB and HINT in a network environment using Microsoft Windows 95. On a slow machine (25 Mhz 486) downloading took a "normal" amount of time. When the same code was downloaded on a 80 Mhz machine, downloading took considerably longer. When the 80 Mhz machine operated in DOS mode, downloading took the normal amount of time.

GDB DEBUGGER - QUICK GUIDE

The GDB debugger manual is included in the development system. This section shows you how to use the debugger with the demonstration programs.

Starting GDB
The general form is invoked at the DOS prompt:

```
gdb -x program.gsc
```

GDB must be in the /GNU/BIN directory and your path must point to it. The `.gsc` file is in ASCII text. It specifies start up parameters such as PC COM port and program to use and load. It replaces you having to type in all of the parameters to get going.

Make sure you edit the `.gsc` files for the proper COM port on your PC. Default in these demo files is COM1. If you are using COM2, the programs will not work.

Batch files in the demo programs that use the debugger invoke it when called. The following are a list of directories and demo programs that use the GDB debugger:

Other command line options are displayed by typing at the DOS prompt:

```
gdb -help
```

Running and debugging programs using GDB
If you use one of the batch files in the above directories, the program will automatically download to the card. You are now ready to run or debug your program.

A short list of commands to get you started is listed below. Some of the most frequently used commands are listed at the end of this section.

break - sets break points
continue - Starts and resumes execution
print or x - display data
list *function* - Print lines starting at *function*
list *line number* - Print lines in current source file
info line *number* - Prints starting and ending addresses of compiled code.
disassemble [*program counter*] - Displays machine instructions.
step - Run program to next line
stepi - Run line, break, and display next line to execute
help - display help commands

While the program is loading, the debugger more or less displays the progress. When you see the

```
(gdb)
```

prompt, you can type in commands. To start execution, use the "continue" command.

Program crashes
In the unlikely event your program should crash, press the reset switch on the RPC-400 board and wait a few seconds. GDB should acknowledge a "SIGTRAP".

From here you can do one of two things. Type "quit" to go back to DOS; or you can re-start the program.

If things really get messed up (usually on the GDB end), you can exit out by hitting the < Ctrl> Break keys. This tends to mess up DOS when operating from Microsoft Windows® 95 if you do this several times (you get kicked back into Windows).

When GDB receives the "SIGTRAP" error, you must reset GDB and reload the program. While in GDB, enter the following at each (gdb) prompt:

```
target remote com1
```

If your serial port is COM2, use that. Then enter:

```
load file.x
```

Where your file is the one you are debugging. If you don't remember the name of the file, type "run". GDB will prompt you to start the program then return an error saying it can't. However, the file it was trying to run is displayed. After you have loaded the file, use "continue" or "stepi" to start execution. The command "stepi" shows you the lines it will execute. Simply pressing "Enter" on the PC repeats the last command.

GDB COMMAND LINE REFERENCE

Many frequently used GDB commands are given below for your reference. All commands are entered after the (gdb) prompt.

Command	Description
target remote COM1	Informs GDB it is communicating with the card.
list main	Display C source code starting at 'main'. You can list starting at any function.
< Enter>	Repeat last command or show additional lines.
list nn	Display C source code starting at line 65
list	Continue displaying C source code lines
info line nn	Display start and end addresses for C code at line nn
disassemble 0xnnnnn	Disassembles code starting at address 0xnnnn
break 90	Break at C source code line 90
break *0xnnnnn	Break at address 0xnnnnn
set $pc-0xnnnnn	Set program counter to address 0xnnnn
reg	Display all CPU registers

Command	Description
display y	Display current value of global variable y
display array[n]	Display current value of global array 'array' at element n
x 0xnnnn	Examine memory starting at address 0xnnnn
x /3w 0xnnnn	Display 3 words of memory at address 0xnnnn
x /4b 0xnnnn	Display 4 bytes of memory at address 0xnnnn
set count= 5	Set C source code variable 'count' to 5
set *0xnnnn= xxx	Set address 0xnnnn in memory to xxx
clear	Delete all break points
info break	Prints table of all break points and watch points
watch count	Set watchpoint 'count'. When 'count' changes, program stops execution.
del 3	Delete breakpoint/watchpoint
info program	Display information about status of program
help	Display help items
continue	Start/resume program execution at address where program last stopped.
quit	Exit gdb

DEBUGGING WITH HINT

HINT is the Hitachi terminal program. Using this program allows you to use the CMON monitor commands. The main disadvantage to using HINT is you cannot debug using your C source code easily. The advantage is if you frequently use the reset button, getting restarted is a lot quicker.

You will need to use HINT to save your program to flash EPROM to autorun it.

HINT is invoked using the following syntax:

```
hint com1
```

If your serial port is COM2, then use that instead.

When power is applied or the reset button is hit, you should get the sign on message.

To load programs, output code must be in .SR format. Use the 'l' command to load programs. For example, the code in the LCD4X20 directory is in .SR format. To load the code, simply type:

```
l :rpc400.sr
```

The HINT program will take care of loading it from the disk and sending it to the card. When loading is complete, the lowest and highest address as well as the starting address is displayed.

To run a program, simply type the command 'g' followed by the address. Do not use the prefix "0x" as all addresses using the monitor are understood to be hex.

If you forget to write down the starting address after a load, you can always view it by entering the command

```
dw 9002000
```

The first two words are the starting address. It should be something like 9002500. This will vary depending upon the number and length of interrupt programs and text in your program.

ELECTRICAL SPECIFICATIONS

CPU
Hitachi SH-1 (SH7032) RISC processor

Clock
19.6608 Mhz

Memory
High speed (20 ns) SRAM - 256KB. Program execution.
Low speed (100 ns) SRAM - 32K to 512KB. Data storage.
Flash EPROM - 32K to 512KB. Program and data storage.

Monitor ROM
Socket accepts 32K or 64KB EPROM. Based on Hitachi CMON. 32KB installed.

Auto-run on power up (jumper W16 removed)

Monitor used during development for initial downloading and debugging programs. When auto-run jumper W16 is removed, monitor merely loads a program from flash into RAM and begins execution.

Analog Input
Channels:
- 24 total single ended
- 8 - from J1
- 16 - from P2 and P3

8 differential maximum from P2 and P3. May be dynamically switched between SE and differential.

Resolution:
- J1 - 10 bits
- P2 and P3 - 12- or 16- bits, depending upon model ordered.

Trigger:
Conversion at an input on J1 can start on external trigger.

Accuracy:
Adjustable. See section 10 for discussion. When averaging readings, ±4 bits.

ENOB:
- 12 bit converter - 11.3
- 16 bit converter - 13.5

Input range:
J1 - 0 to 5V or Vref input, which ever is greater.
P2 and P3 - 0 to 5V or ±5V at X1 gain. Range is reduced in proportion to gain. At X800 gain, range is 6.25 milli-volts.

P2 and P3 Gain:
Software programmable from X1 to X800. Pre-amplifier programmable for X1, X2, X4, or X8. Post amplifier programmable for X1, X10, or X100.

Input Impedance:
J1 - 100K ohm to ground
P2 and P3 - 1 Meg ohm to ground

CMRR:
-75dB minimum at 120 Hz.

Conversion time:
J1 - 6.5 micro-seconds
P2 and P3 - Less than 10 micro-seconds

Accuracy:
J1 - Depends upon + 5V supply or external reference.
P2 and P3: ±(5 counts + 0.05% of input)
Dependent upon gain.

Settling time:
Depends upon gain and voltage change. Generally less than 30 micro-seconds.

Over voltage:
±30V maximum
Other channels are affected when over-voltage.

Analog output
Channels: 4 (optional)
Type: Voltage and 4-20 mA current loop
Range: Voltage outputs jumperable for 0-5, 0-10, and ±5V.
Maximum current (voltage output): 1 mA for rated speed and accuracy
Accuracy: ±2 bits (voltage)
±0.05% (current)
Part kit: 1454 for voltage output, 1678 for current.
Current loop supply: 13 to 36 volts.

DAC output drives current loop. DAC not available when driving current loop.

Serial I/O
2 or 4 available, depending upon model chosen.
Type:
Two port model - RS-232
Four port model - RS-232 and RS-422/485
Four port models include isolated RS-422/485/232

Isolation voltage: 100 VAC maximum

Connector:
All models - programming port uses 10 pin IDC
All models - all other ports use DB-9 male. Pin out matches that of a PC.
Signals:
All models - Txd, Rxd, CTS, RTS
CTS and RTS not available on RS-422/485 port

Multi-mode counter
Channels: 4
Types:
Quadrature encoder and pulse and pulse width
Count rate:
Quadrature encoder - 1 Mhz maximum with minimal filter.
Pulse input - 20 Mhz
Pulse width - 203 nS minimum width, 3.3 seconds maximum.
Count bits:
24 (each channel)
Input voltage:
Counter 0 is ±12 volts, adjustable trigger level
Counters 1-3 are TTL levels

Interrupts
Total externally available: 5
Interrupt latency: 2 micro-seconds, typical
Pull-ups : All external pulled up to + 5V through 10K resistor.
A-D trigger for 10 bit input not counted as an interrupt.

Digital I/O
Groups and purpose:
J2 Keypad from 82C55
J4 Display from 82C55
J6 Digital I/O from 82C55
J13 Digital I/O and timing from CPU

Description
J2 10 lines primarily used for keypad. May be used for digital I/O if keypad not used. I/O from port B and C of 82C55.

J4 14 lines primarily used for display. May be used for digital I/O if display not used. I/O

from port A and B of 82C55.

J6 24 lines. Primary digital port. 8 high current lines sink up to 500 mA at 50V. When all 8 lines are on, maximum is 80 mA/line. Output disabled by substituting a dip shunt jumper for the driver IC.

J13 13 lines, 12 are from CPU port B and 1 from CPU port A. Some lines may be used for timing and interrupts.

I/O Electrical Specifications

82C55 Low voltage is 0.45V at 2.5 mA, 1V maximum at 15 mA for opto rack interface. High voltage is 2.4V min., sink or source current is 2.5 mA.
Lines are programmable as inputs or outputs in groups of 4 or 8.

CPU Lines are TTL compatible, programmable for inputs or outputs. Some are Schmidt trigger inputs. Refer to CPU manual for specific information.

Keypad input

10 lines accept a 16 - 24 position matrix keypad.

Display port

14 lines to control character and graphics displays.
Contrast adjustment available.
Mate with RPC P/N 1032, 1033, and 1034 displays.

Power Requirements

+ 5V, ±0.25 @ 400 mA (4 serial port model)
±9 to ±15 @ 50 mA. Minimum voltage may be greater if using analog output.
External supply required for current loop.

Environmental

Operating temperature range: 0°C to 70°C
Storage range: -20°C to + 70°C

Mechanical

Size: 6.0" x 10.0"
Mounting: 4 mounting holes 0.25" from edges, 1 near center. Hole diameter is 0.125".

System Memory and I/O Map

Description	Address Range (Maximum)	Memory Area	Access Width (bits)
Monitor ROM U22	000 0000 - 000 FFFF	0	8
Flash program storage U23	200 0000 - 207 FFFF	2	8
Battery backed data RAM U24	208 0000 - 20F FFFF	2	8
Digital I/O port at J6	600 0000 - 600 0006	6	8
Serial port COM2	600 0080 - 600 008E	6	8
Serial port COM3	600 0100 - 600 010E	6	8
Keypad and display port	600 0180 - 600 0186	6	8
Expansion port J12	600 0200 - 600 027F	6	8
Counter IRQ and pwm	600 0280 - 600 0288	6	8
A-D control and data	600 0290 - 600 02A0	6	8
Counter and IRQ control	600 02A8 - 600 02D0	6	8
DAC and 4 - 20 mA	600 0300 - 600 030F	6	8
Program executable RAM U25, U26	900 0000 - 90F FFFF	1	16
CPU RAM	FFF E000 - FFF FFFF	7	32

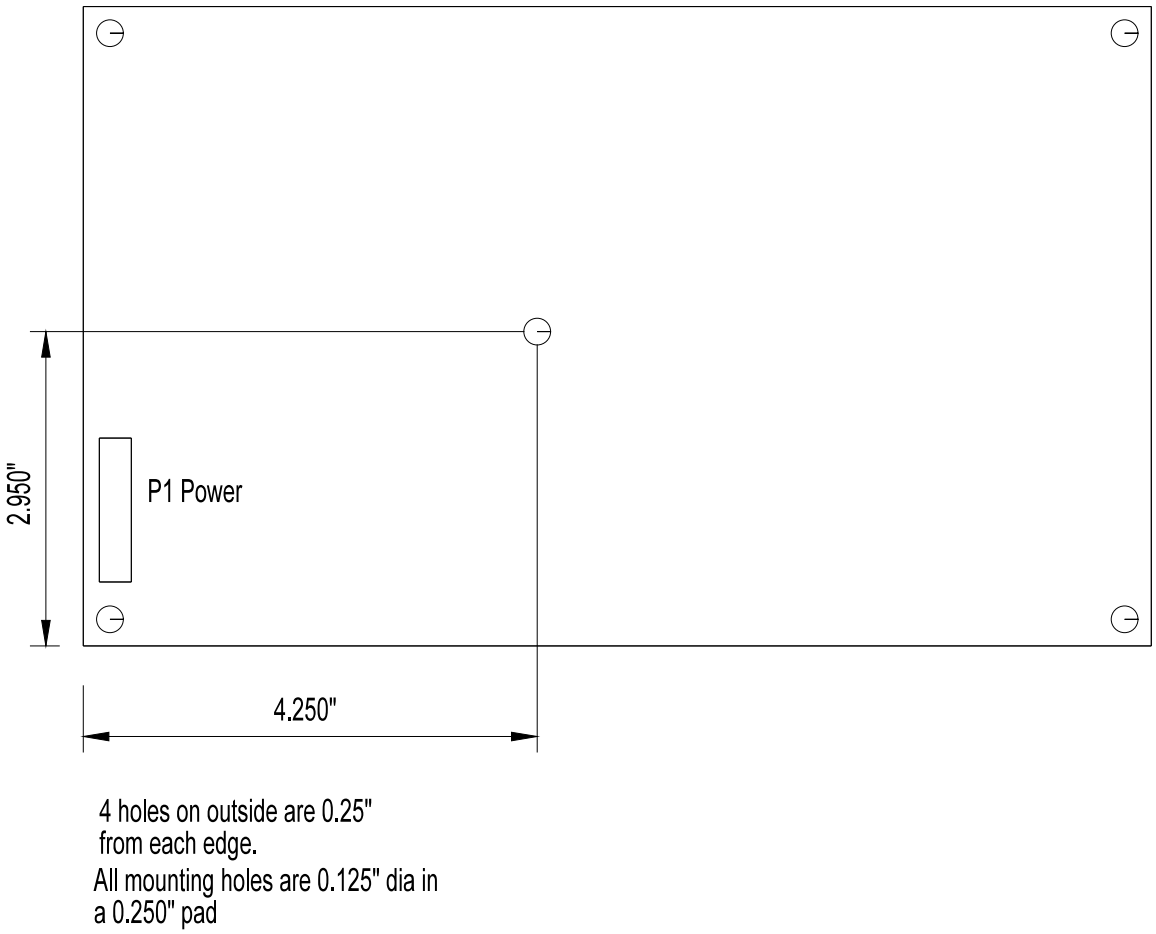


Figure 18-1 RPC-400 mounting holes