IP2022 Internet Processor™

Universal Device Networking Kit

User’s Guide

Revision History

Revision	Release Date	Summary of Changes
1.0	September 16, 2002	Update of previous *Connectivity Kit User's Guide* (IP2K-DUG-CK2UG-13)

Part #:: IP2K-DUG-UNIVDNET-10

Ubicom, Inc.
635 Clyde street
Mountain View, CA 94043
tel 650 210 1500
fax 650 210 8715
www.ubicom.com

Table of Contents

1 Quick Set Up **1**

2 Overview **5**

2.1 Minimum System Requirements 11
2.2 Installing the Software 12
2.3 Contents of the ubicom Directory 13

3 Kit description **15**

3.1 Terminology 16
3.2 Demo Board Set-Up 17
3.2.1 Demo Board Connections 17
3.2.2 Power Supply 17
3.2.3 PC/Terminal Connection 18
3.2.4 ISD/ISP Connection 18
3.2.5 Verifying Installation 18
3.3 Description of Hardware Blocks 19
3.3.1 IP2022 19
3.3.2 Reset 19
3.3.3 Clock 19
3.3.4 RTCLK 20
3.3.5 Power 20
3.3.6 ISD/ISP 20
3.3.7 Analog Unit 21
3.3.8 Run LED 21
3.3.9 SPI Serial Flash Memory 21
3.3.10 On-Board Power Supply 21
3.3.11 User LEDs 22
3.3.12 Additional DATA SRAM 22
3.3.13 Switches 22
3.3.14 Prototype Area 22
3.3.15 Oscilloscope I/O Breakout 23

3.4 Connectors. 23
3.4.1 Daughter Board Connectors (J2, J3). 23
3.4.2 Test Points 26
3.5 Board Configuration Options. 26
3.5.1 RS232 Port P1 (JP4, JP12) 28
3.5.2 RS232 Run-Time Debug Port P2. 29
3.5.3 Additional SPI Flash (JP3) 30
3.5.4 Additional Data SRAM (JP2). 30
3.5.5 IOVDD (JP10) 30
3.5.6 Clock (JP13) 30
3.5.8 Run LED (JP8) 31

4 Unity User Interface 33

4.1 Unity Windows. 35
4.2 Unity Menu Bar 35
4.3 Unity Tool Bar 36
4.4 Creating a New Project 40
4.5 Adding and Removing Files 41
4.6 Project File Tree 42

5 Configuring ipModules 45

5.1 Project Configuration GUI 46
5.2 Generating Makefiles and Header Files 48

6 Compiling 49

7 Debugging 53

7.1 Debugging Tool Bar 53
7.2 Viewing Program Execution 56
7.3 Breakpointing Program Execution 58
7.4 Viewing Registers. 62
7.5 Viewing Memory 63
7.6 Memory Map 67

8 Utilities 69

8.1 IP2KProg User Interface 71
8.1.1 Downloading a Project 73
8.2 GNU Binary Utilities 74

9 SDK Demo Projects 77

9.1 Equipment Required 77
9.1.1 PC Network Configuration 78
9.1.2 Configuring IP2022 Board IP Address 79
9.1.3 Network Tips 80
9.1.4 Demo Jumper Settings 81
9.2 Starter Demo 82
9.2.1 IP2022 Board Configuration 83
9.2.2 PC Configuration 84
9.2.3 Demo Project Configuration and Build 84
9.2.4 Using the Demo 85
9.3 Telnet Demo 86
9.3.1 IP2022 Board Configuration 87
9.3.2 Demo Project Configuration and Build 88
9.3.3 Using the Demo 88
9.4 Web Server Demo 91
9.4.1 IP2022 Board Configuration 92
9.4.2 Demo Project Configuration and Build 93
9.4.3 Using the Demo 93
9.5 Serial Gateway Demo 95
9.5.1 IP2022 Board Configuration 96
9.5.2 PC Configuration 97
9.5.3 Demo Project Configuration and Build 97
9.5.4 Using the Demo 97
9.6 SNMP Compiler ipOS Demo 99
9.6.1 IP2022 Board Configuration 100
9.6.2 PC Configuration 101
9.6.3 Demo Project Configuration and Build 101
9.6.4 Using the Demo 101
9.7 USB RNDIS Device Demo 103

9.7.1 IP2022 Demo Board Configuration 103
9.7.2 PC Configuration. 105
9.7.3 Demo Project Configuration and Build 105
9.7.4 Using the Demo 105

A Demo Board Schematics 107

A.1 IP2022 108
A.2 Reset. 109
A.3 Clock 109
A.4 Power 110
A.5 ISD/ISP. 111
A.6 RS-232 Communications. 112
A.7 Analog Unit 113
A.8 On-Board Power Supply 114
A.9 User LEDs 115
A.10 Switches 116
A.11 Osciloscope I/O Breakout 117
A.12 Prototype Area 118
A.13 Daughter Board Connectors (J2, J3). 119

B Other IP2022 Resources 121

List of Figures

Figure 2-1 Configuration Files 6
Figure 2-2 Compilation Tool Chain 7
Figure 4-1 Unity (Default View) 34
Figure 5-1 Project Configuration GUI 46
Figure 6-1 Makefile Structure. 51
Figure 7-1 Browse Tab 59
Figure 7-2 Setting a Breakpoint. 60
Figure 7-3 Stack Window 61
Figure 7-4 Registers Window. 62
Figure 7-5 Quick Watch Box 63
Figure 7-6 Watch Window 64
Figure 7-7 Memory Window 65
Figure 7-8 Debugging Memory Space 67
Figure 8-1 IP2KProg User Interface 71
Figure 9-1 Configuring Demo Board IP Address 79
Figure 9-2 Ping. 80
Figure 9-3 Ipconfig. 80
Figure 9-4 Starter Demo Board Diagram. 83
Figure 9-5 Set New Project Name 84
Figure 9-6 Choose Project Template 85
Figure 9-7 Telnet Demo Board Diagram 87
Figure 9-8 Web Server Demo Board Diagram 92
Figure 9-9 Browser Window. 94
Figure 9-10 Serial Gateway Demo Board Diagram 96
Figure 9-11 Telnet Window 98
Figure 9-12 Hyperterminal Window. 98
Figure 9-13 SNMP Compiler ipOS Demo Board Diagram 100
Figure 9-14 Browsing the SNMP Agent 102
Figure 9-15 USB RNDIS Device Demo Board Diagram 104

List of Tables

Table 2-1 Summary of Files 8
Table 2-2 System Requirements 11
Table 2-3 Summary of Directories 11
Table 2-4 Summary of ubicom Directories 13
Table 3-1 SERDES1 (J2) Pin Assignments 24
Table 3-2 SERDES2 (J3) Pin Assignments 25
Table 3-3 Test Points 26
Table 3-4 Demo Board Jumpers 27

Preface

This manual introduces the tools used for software development on the IP2022. For detailed information about programming the IP2022, see the *IP2022 Programmer's Reference Manual*.

The tools are based on the GNUPro tool chain supported by Red Hat. For detailed information about the tools, see the GNUPro documentation.

Related Documentation

Main documentation for the IP2022, available from Ubicom:

- *IP2022 Data Sheet*, available from Ubicom.
- *IP2022 Programmers Reference Manual*.

Reference manuals for the tool chain, available from Ubicom:

- *Getting Started with GNUPro Toolkit*
- *GNUPro Toolkit User's Guide for IP2022*
- *GNUPro Compiler Tools*
- *GNUPro Debugger Tools*
- *GNUPro Development Tools*
- *GNUPro Auxilliary Development Tools*
- *GNUPro Libraries*

Notational Conventions

In this document, the notation “->” is used to refer to a command selected from a menu. For example, the Save command on the File menu is File -> Save.

The Programs menu is accessed by clicking on the Start button (lower left corner of screen), then clicking on Programs. After installing the Ubicom software, the Programs menu will contain a Ubicom entry which is used to access the software tools. In this document, references to the Ubicom menu actually mean commands selected from the Start -> Programs -> Ubicom menu.

File Naming Conventions

Both MS-DOS and Unix file naming conventions are used in this document. An MS-DOS file name uses backslashes as separators, such as C:\Ubicom\sdk\projects\starter\Makefile.

Unix file names are used for Configuration Tool parameter values, names in **`make`** files, and the SDK directory tree. A Unix file name uses forward slashes as separators, such as /cygdrive/c/Ubicom/sdk/projects/starter/Makefile.

The Unix operating system is case sensitive, e.g. the names "makefile" and "Makefile" would refer to two different files. Because the software tools run under Windows/MS-DOS, however, all file names are interpreted as non-case-sensitive without regard to which file naming convention is used.

Unix file names do not have embedded space characters, so names like "Program Files" cannot be used as names for files or directories in the path to a file.

Chapter Summary

Chapter 1 is a quick procedure to set up the Demo Board, install the CD-ROM software, and compile, download, and run the **`starter`** project on the Demo Board. The **`starter`** project is used as an example throughout this book. The following chapters present more detailed information about each of these steps.

Chapter 2 is an overview of the Unity integrated development environment (IDE) and the **`starter`** project. The CD-ROM software and installation procedure are described in this chapter.

Chapter 3 describes the Demo Board hardware and set-up procedure. The **`starter`** project is preloaded on the Demo Board, and it begins execution after power is applied.

Chapter 4 introduces the Unity user interface and the structure of a Unity project.

Chapter 5 examines the **`starter`** project configuration and walks through generating the **`config.h`** and **`config.mk`** files.

Chapter 6 discusses the events which occur when a project is compiled.

Chapter 7 walks through debugging a project on the Demo Board.

Chapter 8 describes utility programs for working with **`.elf`** files.

Chapter 9 documents the example projects that are included with the SDK.

Appendix A contains schematic diagrams of the various units of the Demo Board.

10 Quick Set Up

The following steps comprise a quick set up procedure for the CD-ROM software and Demo Board.

1. *Register Your Kit* — at Ubicom's Technical Support Portal (www.ubicom.com). This provides access to latest documentation and software available for this kit.
2. *Install Software from the CD-ROM* — uninstall any previous installation of the GNUPro tools (ip2ktools), Unity, and Software Development Kit (SDK). Then run the installation programs for each of these software distributions:

   Run **`\Install\Ubicom_4.2.exe`** (name may vary in the future).
3. *Connect Demo Board* — connect the parallel cable to the host PC*. At the other end of this cable, connect the adapter cable from the parallel cable to the Demo Board. The red strip on the connector to the Demo Board should be closest to the header labelled DGND. Connect the serial cable be-

*. In some instances, the default configuration of the parallel port on the host PC may be incompatible with the programming dongle and drivers of the IP2022 tool chain. If you incur communication problems over the ISD/ISP cable, try switching your parallel port to an alternate configuration setting. This is done in the host PC BIOS, typically accessible via an "F" key or the "DEL" key at power-up. Also, refer to Ubicom's Technical Support Portal at www.ubicom.com for latest recommendations on compatible parallel port configurations.

tween the host PC and the Demo Board serial connector labeled Main P1. Then, connect the power supply cable to the Demo Board and plug the power supply into an AC outlet.

4. *Create a Project* — create a directory called **`C:\Ubicom\SDK_Demo`**. Open Unity by selecting Programs -> Ubicom -> Unity from the Windows Start menu. Then, select Project -> New from the Unity menu bar. For the project name, enter **`C:\Ubicom\SDK_Demo\Starter.c_c`**. For the project type, select SDK and click the OK button. Select **`starter`**, then click the OK button. Compile the project by selecting Build -> Compile from the Unity menu bar.
5. *Download Project* — enter device programming mode by selecting the Build -> Start Programmer command from the Unity menu bar. In the new window which appears, click the Program button, then click the Close button.
6. *Verify Operation* — the LED bank on the Demo Board displays a binary counter. Echoing on the serial communication port can be verified by launching a serial terminal emulator with the following settings:

   9600 baud
   8-bit ASCII
   No parity
   1 stop bit
   No flow control
   Local echo off

   Characters typed in the terminal emulator will be echoed back by the Demo Board.

2.0 Overview

Ubicom's approach to developing embedded Internet applications combines an extensive library of ipModule™ software with Unity, a powerful integrated development environment (IDE). User-friendly tools simplify incorporating these tested modules into applications, resulting in the fastest path from product concept to market entry.

The philosophy behind this approach is to provide a high-level interface to building blocks such as communication packages and peripheral interfaces. By reducing the time and effort dealing with the nuts-and-bolts of register and bit-level operations, the system designer is free to concentrate on product differentiation and increasing value-added for the customer.

The files which comprise an application are called a *project*. There are five phases in project development with Unity:

- *Configuring the Project* — ipModules have options which can be customized for a specific application. For example, a UART ipModule has options for baud rate and port pin assignments. The Configuration Tool, under control of Unity, controls the selection of ipModules that are included in a project and the settings for their options.
- *Creating and Editing Source Files* — Unity includes a source-file editor for C and assembly language files.

- *Compiling the Project* — Unity calls the GNUPro tool chain to compile, assemble, and link the project files, to produce an executable file in ELF format.
- *Programming the IP2022* — Unity calls the ProgrammerTool to download executable code to the IP2022 device through the ISD/ISP interface.
- *Debugging* — the Unity GUI invokes a source-level debugger for powerful C/assembly language debugging.

The files used for project configuration are shown in Figure 2-1. Unity keeps information about the project in a file with the **`.c_c`** extension, such as the names of the source files which compose the project, debugger settings, display fonts, etc.

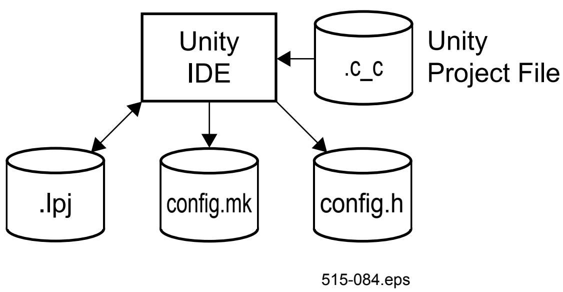


Figure 2-1 Configuration Files

Unity keeps configuration information in a file with the **`.lpj`** extension. Whenever a new configuration is created or an existing configuration is changed, Unity can generate new **`config.mk`** and **`config.h`** files. These files contain macro definitions that control ipModule selection and option settings during compilation.

The files used by the GNUPro tool chain are shown in Figure 2-2.

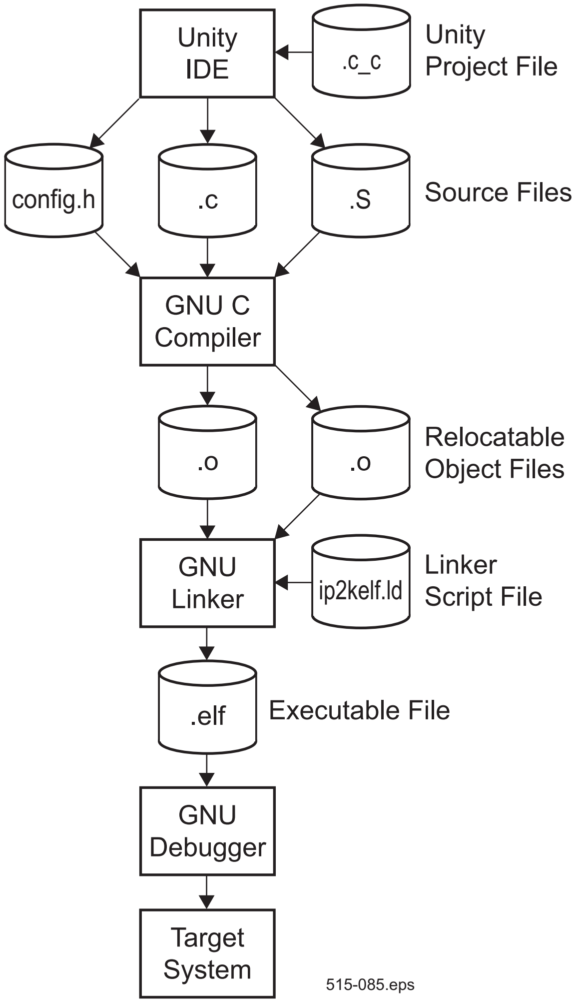


Figure 2-2 Compilation Tool Chain

The **`config.h`** header file is generated by the Configuration Tool. Files with the **`.c`** and **`.S`** extensions are C and assembly language source files, respectively. For each **`.c`** and **`.S`** source file listed in the **`.c_c`** project file, the GNU C compiler is called to generate a relocatable object file with the **`.o`** extension.

The GNU linker reads the **`.o`** object files and a linker script file, and produces an executable file with the **`.elf`** extension. Most users will not need to modify the default linker script file **`ip2kelf.ld`** provided by Ubicom. A powerful source-level debugger downloads the **`.elf`** executable file to the target system and controls its operation.

The files used by Unity and the GNUPro tool chain are summarized in Table 2-1.

Table 2-1 Summary of Files

File Type	Description
`.c`	*C source file* — source file(s) for the application.
`.c_c`	*Unity project file* — used by Unity to store project-specific information (location of source files, etc.).
`.elf`	*Binary file* — executable code in the ELF file format.

Table 2-1 Summary of Files

File Type	Description
`.lpj`	*Configuration Tool project file* — used by Configuration Tool to store project-specific information (ipModule selection, I/O port pin assignments, etc.).
`.o`	*Object file* — compiled object code.
`.S`	*Assembly-language source file* — pure assembly language file, as opposed to in-line assembly in a C source file.
`config.h`	*C header file automatically generated by Configuration Tool* — macro definitions used by the C compiler.
`config.mk`	*Makefile automatically generated by Configuration Tool* — macro definitions used by the **`make`** utility.
`Makefile`	*Project makefile* — the top-level makefile in the project directory called by Unity.
`Makefile.inc`	*File automatically generated by Unity* — specifies rules for compiling the files referenced in Makefile.gen.
`Makefile.gen`	*File automatically generated by Unity* — has list of all Project files have to be compiled
`ip2kelf.ld`	*Linker script file* — most users will not need to modify the default **`ip2kelf.ld`** file provided by Ubicom.

A typical development sequence is:

1. *Create new project using one of the SDK template projects*. All template project files will be copied into new project directory automatically.Unity keeps information about the project, such as the names and locations of its source files, in a file with the **`.c_c`** extension.
2. *Use Unity->Configuration Tool to* select the ipModules used in a project and customize their features. The Configuration Tool keeps information about the project configuration in a **`.lpj`** file. It uses this information to generate the **`config.mk`** and **`config.h`** files.
3. *Write Source Files in C or Assembly Language.* Unity has a simple, intuitive text editor for source files. C source file names have a **`.c`** extension, and assembly language file names have a **`.S`** extension.
4. *Compile the Project*. An executable **`.elf`** file is produced. Intermediate files such as object files (**`.o`** extension) are generated in the project directory.
5. *Debug the Project*. The debugger downloads and controls program execution on the target.

2.1 Minimum System Requirements

The hardware and software requirements for running the Unity IDE are described inTable 2-2:

Table 2-2 System Requirements

Feature	Requirements
CPU	Pentium 2 or equivalent 300 MHz or greater
Memory	64 Mbytes
Free Disk Space	300 Mbytes
Operating System	Windows 98/NT/2000/XP
Peripherals	*Required*—parallel port (IEEE 1284-1994 compliant ECP mode, or STANDARD or OUTPUT ONLY mode) *Recommended*—serial port, Ethernet interface

The installation CD-ROM has the structure described in Table 2-3.

Table 2-3 Summary of Directories

Name	**Description**
General Documentation	Key silicon and hardware documents
Install	Installation programs
Software Development Tools	Source code and manuals for GNUPro IP2022 tool chain

2.2 Installing the Software

Although the software can be installed anywhere, in the beginning you should keep the default directory structure to avoid errors caused by relocating the directories. Names used for directories and source files must not have embedded spaces (e.g. "Program Files").

1. *Remove Previous Installation*. If an earlier installation used a different installation directory for the tools, it may be necessary to remove the Ubicom menu to prevent Windows from searching for the tools in their old location. Any files compiled with an earlier release of the tools should be removed and recompiled from their sources.
2. *Install IP2022 Tools, SDK, Unity and Documentation.* Run **`Install\Ubicom_4.2.exe`** (name may vary). Allow all of the default selections to be used. Unity automatically updates the system path with "**`C:\ubicom\ip2ktools\H-i686-pc-cygwin32\bin`** `path`.
3. *Restart the computer.* This is necessary for the changes to take effect on system.

2.3 Contents of the `ubicom` Directory

After installation, the **`ubicom`** directory has the subdirectories, described in Table 2-4.

Table 2-4 Summary of `ubicom` Directories

Name	Description
Help	Documentation files for IP2022 Development Boards and GNUPro Tools.
ip2ktools	Executable files for tools and utilities.
sdk	Project templates, ipModule software, and on-line help file.
Unity	Unity executable file and on-line help files.

3.0 Kit description

The IP2022 Evaluation Kit provides a cost-effective demonstration and evaluation platform for the IP2022 Internet processor. The Evaluation Kit comes with source code and software tools on CD-ROM for developing applications using the ipModule library. The complete documentation set for the hardware and software is included on the CD-ROM.

The IP2022 Universal Device Networking Kit contains:

- IP2022 Demo Board (silk screen labelled as ”IP2022 Demo Board V3.0”)
- ipEthernet daughtercard board
- AC Adapter (12 VDC output)
- ISD/ISP Cable (between PC parallel port and Demo Board)
- DB9 Serial Cable
- Ethernet crossover (grey) and straight-through (red) cables
- CD-ROM with Software Tools and Documentation.

The Demo Board provides the hardware to demonstrate the IP2022 and ipModules. Its hardware features include:

- IP2022 Internet Processor
- 12 VDC Power Input
- Two RS-232 Serial Connections (Main and Run-Time Debug)

- Daughter Board Connectors for Ethernet, USB, and other-Interfaces
- 5V, 3.3V, 2.5V Voltage regulators with Power-On LED
- Run LED (controlled from software)
- 4.8 MHz Crystal
- 32.768 kHz Crystal for RTCLK
- Reset Button
- External 4 Mbit SPI Flash
- Analog Temperature Sensor connected to ADC I/O
- 1 Pushbutton
- 4 DIP Switches
- 8-LED Bank
- Testpoint for ADC Iinput/External Reference Voltage
- ISP/ISD 10-pin Header
- Prototyping Area

3.1 Terminology

Some terms used in discussing the Evaluation Kit are:

- *Daughter Board* — a piggyback board which connects to a 60-pin connector on the Demo Board to provide extra hardware for a particular interface, e.g. an Ethernet Daughter Board with transformer.
- *ipModule* — software module from the Ubicom library.
- *ISD/ISP* — in-system debugging/in-system programming. A standard 10-pin ISD/ISP connector provides all necessary signals for downloading code to the IP2022 and controlling and monitoring its operation.

- *SDK* — Software Development Kit
- *SerDes* — Serializer/Deserializer multiprotocol serial communications peripherals on the IP2022
- ***`starter`*** *Example* — complete small application for demonstrating the Demo Board and software tools. Also used as a template for developing new applications.

3.2 Demo Board Set-Up

3.2.1 Demo Board Connections

The board is preloaded with the **`starter`** example which begins running immediately after power is applied, if the cable connections and jumper placements are correct. To ensure proper operation, the board is provided with jumpers in place for running the **`starter`** example. The jumper settings are covered in Section 3.4.1. The cable connections to be made are:

- Power Supply
- PC/Terminal serial connection
- ISD/ISP parallel port cable

3.2.2 Power Supply

The supplied wall-mount AC adapter provides 12 VDC at 800 mA. If the supplied AC adapter is not used, the applied voltage should be within the range of 8 to 15 VDC. The center contact of the power connector is positive voltage with respect to the sleeve.

3.2.3 PC/Terminal Connection

Any terminal or terminal program may be used if it allows for the following settings:

- 9600 baud
- 8 bit ASCII
- No parity
- 1 stop bit
- No flow control
- Local echo off

3.2.4 ISD/ISP Connection

A parallel port cable (DB25 male to DB25 female) and a short converter cable (DB25 male to 10-pin female header) are used for in-system debugging and programming. The parallel port cable is connected between the parallel port connector on a PC and the DB25 male connector on the converter cable. The 10-pin connector on the converter cable is connected to the Demo Board. The connector mates to a group of nine header pins on the Demo Board. Pin 1 of the connector is indicated by a red stripe. The connector is installed with Pin 1 toward the right edge of the board.

3.2.5 Verifying Installation

After the cables are connected, power can be applied to the board. The power LED on the Demo Board should light up, and the LED bank should begin strobing a single lit LED from left to right.

3.3 Description of Hardware Blocks

3.3.1 IP2022

The IP2022 Internet processor is packaged in an 80-pin PQFP. It contains a CPU, 64 Kbytes of program flash memory, 16 Kbytes of program RAM, 4 Kbytes of data RAM, and many hardware peripherals. The fast RISC CPU (up to 120 MIPS) allows emulating many more peripherals using ipModule software.

3.3.2 Reset

There are five possible sources of IP2022 reset:

- *Power-On Reset*—automatically triggered after power is applied.
- *External Reset*—manually triggered from a pushbutton.
- *Brown-Out Voltage Level*—triggered if Vdd drops below the brown-out voltage level.
- *Watchdog Timer*—if enabled, software must periodically execute a **`cwdt`** instruction to avoid reset triggered by overflow of the Watchdog Timer.
- *ISD/ISP interface*—reset issued from the debugger (Unity).

3.3.3 Clock

A 4.8 MHz crystal is connected between pins OSC1 and OSC2 of the IP2022. OSC1 input can be disconnected from the 4.8 MHz crystal and connected to the can-oscillator socket U9 by switching jumper JP13 into position with pins 2-3 closed.

3.3.4 RTCLK

A 32.768 kHz crystal is connected between pins RTCLK1 and RTCLK2 of the IP2022.

3.3.5 Power

The IP2022 core logic is powered from the +2.5V digital power rail. The IP2022 pins, except Port G, are driven from the IOVDD power rail, which is selectable between 3.3V and 2.5V. Port G is driven from the 2.5V analog power rail, as are the ADC and Analog Comparator. A separate analog power rail is provided to isolate the analog section of the IP2022 from noise on the digital power rail.

3.3.6 ISD/ISP

The ISD/ISP connector provides a debugging and programming interface to a host PC through a standard 10-pin connector. Pull-up resistors on input signals allow the ISD/ISP interface to be disconnected while the board is powered, however the interface should be closed (disconnected) from the debugger before removing the electrical connection, to avoid generating spurious debugger commands.

3.3.7 Analog Unit

Only the ADC channel 3 (RG3) is connected to testpoint TP1. However, ADC channel 0 through 6 can also be accessed from the prototype area or through the daughter board connectors (SERDES1 and SERDES2).The ADC may use the internal IP2022 voltage reference or an external reference on the RG3 port pin. An analogue temperature sensor connected to ADC channel 2 and may be used for temperature readings to the IP2022 .

3.3.8 Run LED

Software can indicate code execution on the Run LED connected to port RA1, when jumper JP8 is closed.

3.3.9 SPI Serial Flash Memory

An additional serial SPI Flash memory U5 provides 512K bytes of memory, which can be used as storage by the ipFile software module. The IP2022 can communicate with the FLASH via the ipStorage software module.

3.3.10 On-Board Power Supply

An on-board power supply unit provides the following voltages from the 12 VDC input:

- +5V at 500 mA
- +3.3V at 500 mA
- +2.5V at 500 mA

The POWER LED D11 indicates when power is on.

3.3.11 User LEDs

The LED bank is connected to Port B through a tri-statable buffer U9.

3.3.12 Additional DATA SRAM

The IP2022 supports addressing of additional data SRAM. The board has a 128k x8 SRAM IC U1(ISSLV1024). Low address bits of the SRAM are connected to the IP2022 through the latch U3 (74AC573).

3.3.13 Switches

SW1 is a general purpose switch bank. SW1 can be used to put 4-bit binary code on RA3..RA0 port pins.

3.3.14 Prototype Area

All I/O pins are brought to the prototype area to allow design expansion. Care must be exercised since many of the I/O pins are used elsewhere on the board.The prototype area has +5V, +3.3V, and GND rails. The prototype area is 21 by 18 plated-through holes.

3.3.15 Oscilloscope I/O Breakout

All IP2002 ports (RA, RB, RC, RD, RE, RF) are brought out to J6, J7, and J8 (100 mils spacing) for probing and prototyping.

3.4 Connectors

There are five connectors on the Demo Board:

- DC power J12
- Serial RS232 Main (DB9) P1
- Serial RS232 Run-Time Debug (DB9) P2
- SERDES1 (60-pin) J2
- SERDES2 (60-pin) J3
- ISD/ISP (10-pin) J4

3.4.1 Daughter Board Connectors (J2, J3)

Two 60-pin interface connectors are provided on the IP2022 Demo Board for the purpose of interfacing to daughter boards. SerDes 1 (Port RE) is connected to J2 and SerDes 2 (Port RF) is connected to J3. Connectors J2 and J3 have slightly different connections for the RE and RF ports, which makes it possible to swap daughter boards between connectors.

Note: Pin 1 of connectors J2 and J3 is marked by a small groove in the plastic. A similar groove on the daughter board connector determines correct orientation of the daughter board. The correct alignment of each daughter board is such that it extends away from the Demo Board, and does not overlap it.

Table 3-1 SERDES1 (J2) Pin Assignments

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	GND	16	RB7	31	RD6	46	RG5
2	+5V	17	RC0	32	RD7	47	RG6
3	+3.3V	18	RC1	33	RE0	48	RG7
4	+2.5V	19	RC2	34	RE1	49	LAT0
5	RA0	20	RC3	35	RE2	50	LAT1
6	RA1	21	RC4	36	RE3	51	LAT2
7	RA2	22	RC5	37	RE4	52	LAT3
8	RA3	23	RC6	38	RE5	53	LAT4
9	RB0	24	RC7	39	RE6	54	LAT5
10	RB1	25	RD0	40	RE7	55	LAT6
11	RB2	26	RD1	41	RG0	56	LAT7
12	RB3	27	RD2	42	RG1	57	+3.3V
13	RB4	28	RD3	43	RG2	58	+2.5V
14	RB5	29	RD4	44	RG3	59	+5V
15	RB6	30	RD5	45	RG4	60	GND

Table 3-2 SERDES2 (J3) Pin Assignments

Pin	Signal	Pin	Signal	Pin	Signal	Pin	Signal
1	GND	16	RB7	31	RD6	46	RG7
2	+5V	17	RC0	32	RD7	47	RG4
3	+3.3V	18	RC1	33	RF4	48	RG5
4	+2.5V	19	RC2	34	RF5	49	LAT0
5	RA0	20	RC3	35	RF6	50	LAT1
6	RA1	21	RC4	36	RF7	51	LAT2
7	RA2	22	RC5	37	RF0	52	LAT3
8	RA3	23	RC6	38	RF1	53	LAT4
9	RB0	24	RC7	39	RF2	54	LAT5
10	RB1	25	RD0	40	RF3	55	LAT6
11	RB2	26	RD1	41	RG0	56	LAT7
12	RB3	27	RD2	42	RG1	57	+3.3V
13	RB4	28	RD3	43	RG2	58	+2.5V
14	RB5	29	RD4	44	RG3	59	+5V
15	RB6	30	RD5	45	RG6	60	GND

3.4.2 Test Points

The IP2022 Demo Board has several test points as listed in Table 3-3.

Table 3-3 Test Points

Test Point	Description
TP1	ADC input3 / External reference voltage
TP2	Analog ground for ADC signals
TP3,TP4,TP5	Power supply ground

3.5 Board Configuration Options

Jumpers are used to connect common hardware blocks to the IP2022. SerDes 1 and 2 can be connected to a variety of devices. Some jumpers are used to enable groups of signals to IP2022 pins. Care should be exercised so that multiple devices are not connected to the same SerDes unit. To mimimize the number of configurable jumpers, the board has tri-statable buffers U6 and U8, which enable/disable connection of the LED bank and RS232 port P1 signals to IP2022 I/O pins. The jumpers are described in Table 3-4. When a jumper is shown as "unconnected when shipped", it means that the Demo Board is shipped with the jumper over one pin of the header. This is used merely to hold the jumper.

Table 3-4 Demo Board Jumpers

Block	Description	Jumper	IP2022 Signal	Connected When Shipped?
Serial Ports: P1	PC_DTRCD1 PC_RXD1, PC_TXD1, PC_RTS1,	JP12	RF3, RF7, RF1, RF0,	Yes
Port P1	Main LoopBack	JP5		No
Port P1	Modem/PC	JP4		No
Serial Ports: P2	Data LoopBack (LpB)	JP6	-	No
SPIFlash	Chip Select signal CS	JP3	RA0	No
IOVDD	2.5V / 3.3V	JP10	IOVDD	3.3V(2-3)
ISP/ISD	CLK from Tool	JP1	OSC1	No
Clock	XTAL/CAN	JP13	OSC1	XTAL(1-2)
LEDs	RUN LED	JP8	RA1	No
LEDs	8 LED BANK	JP7	RB0–RB7	Yes
SRAM & Latch	SRAM 128Kx 8	JP2	RD0–RD7	No

Table 3-4 Demo Board Jumpers (continued)

Block	Description	Jumper	IP2022 Signal	Connected When Shipped?
Power Supply Unit	5V 3.3V 2.5DV 2.5AV	JP14 JP15 JP16 JP17		Yes Yes Yes Yes
Configuration Switches	Push-Button	JP9	RB2	No
Configuration Switches	4-Bit DIP	JP11	RA0–RA3	No

Jumpers J14, J15, J16, and J17 should be closed unless there is a need to use an external source for one of the IP2022 power supply voltages.

3.5.1 RS232 Port P1 (JP4, JP12)

Port P1 can be connected to SERDES2 pins only. If jumper JP12 is open, SERDES2 pins RF0, RF1, RF3, RF7 may be used for general I/O or for one of the serial protocols.

For PC communications:

1. Use the PC DB9 connector.
2. Install jumper JP4 and jumper JP12.
3. Use a straight-through serial cable.

For Modem communications:

1. Open jumper JP4.
2. Use a null modem serial cable.

Jumper JP5 is used only for checking the connection between the PC and the Demo Board — installing this jumper causes characters to be looped (echoed) back to the PC and can thus be used to determine if the PC's serial port and cable are correctly set up.

3.5.2 RS232 Run-Time Debug Port P2

Some applications where both SERDES units are in use may not allow software UART modules to send debug information to the PC, because having one will upset the determinism requirement of other more important virtual peripherals, such as Ethernet or USB. To allow debug information exchange for such applications the board has U2, an SX20 microcontroller. The SX20 interfaces to the IP2022 via a synchronous interface, **not** a UART interface. The SX20 takes care of streaming the data to the PC using a 115Kbit/sec UART.

Jumper JP6 is used only for checking the connection between the PC and the Demo Board — installing this jumper causes characters to be looped (echoed) back to the PC and can thus be used to determine if the PC's serial port and cable are correctly set up.

3.5.3 Additional SPI Flash (JP3)

U5 is a serial SPI Flash with 4Mbit capacity connected to the IP2022 by 4 pins: SO, SI, SCLK, and CS. To use the serial flash, jumper JP3 should be closed. To read and write U5, the ipStorage software module uses pins RA0–RA3.

3.5.4 Additional Data SRAM (JP2)

To enable SRAM, close JP2. To access SRAM, the software module uses all pins on Port RD, RC and five lines from port RB: RB3–RB7.

3.5.5 IOVDD (JP10)

IOVDD can be selected with jumper JP10 to be 2.5V or 3.3V.

3.5.6 Clock (JP13)

Jumper JP13 has two positions. The lower position (closest to XTAL) connects the 4.8 MHz crystal to OSC1. The upper postition connects OSC1 to the can-oscillator socket U9

3.5.7 4-Bit DIP Switch (JP11)

SW1 may be used to put 4 bit binary code on pins RA0–RA3. Signals from SW1 are connected to IP2022 port RA through the tri-stateable buffer U6. To enable SW1 signals to pins RA0–RA3 install jumper JP11.

3.5.8 Run LED (JP8)

The D10 LED labeled RUN is connected through JP8 to pin RA1. Active level is low. To enable the RUN LED, install the jumper.

4.0 Unity User Interface

Unity implements a powerful graphical user interface (GUI) that trims the rough edges from the GNU tool chain. Even though standard utilities such as **`make`** and **`gdb`** are running at some level of the software development environment, Unity presents a simple and intuitive front-end to these tools that does not require reading hundreds of pages of documentation before getting started. A project can be written, compiled, and debugged without ever looking at a makefile or a linker script file.

The main software development interface which accounts for most of the time spent working in Unity is the integrated source-code editor/debugger, however there are two other modes which are used for short periods:

- *Project Configuration* — used once when the project is started, and used again whenever there is a change in the hardware configuration or ipModule usage. This mode is discussed in Chapter 5.
- *Device Programming* — used to download new programming to the target. This mode is discussed in Chapter 8.

Figure 4-1 shows a view of the default appearance of the Unity GUI. Windows used for project management and source file editing usually occupy the left pane of the Unity GUI, and windows related to debugging occupy the lower right pane. However, Unity allows considerable flexibility in organizing the windows, including the ability to drag a window out of the Unity GUI and onto the desktop.

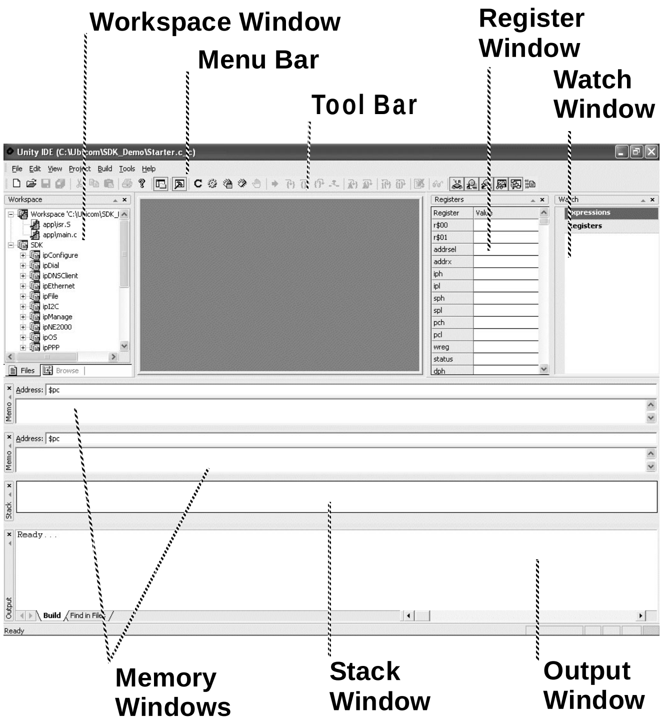


Figure 4-1 Unity (Default View)

4.1 Unity Windows

The default view has up to seven open windows:

- *Workspace Double Window* — lists the source files within the currentproject. Clicking on a file name opens the file in a new window.
- *Memory Windows* — consists of two windows that display regions of the program flash, program RAM, and data memory space.
- *Stack Window* — shows the nesting of subroutines on the software call/return stack.
- *Register Window* — displays the contents of the registers mapped in the IP2022 data space x00...FF. The contents of the registers can be edited in this window.
- *Watch Window* — shows the current value of expressions and organized registers.
- *Output Window* — shows compilation output, such as error messages.

On small screens, the Watch and Output windows might not appear in the default view unless the window is maximized.

4.2 Unity Menu Bar

The menu bar is used to access the Unity commands:

- *File Menu* — New, Open, Close, Save, and Print commands for source files. These commands do not affect Unity project files.

- *Edit Menu* — Cut, Copy, Paste, and Find commands.
- *View Menu* — commands for showing or hiding any of the Unity windows and other parts of the GUI.
- *Project Menu* — commands for creating a new project file or opening an existing project file. Also has commands for generating an HTML version of the source files and accessing a simple version control mechanism.
- *Build Menu* — commands for controlling compilation, entering debugging mode, and launching the IP2KProg utility.
- *Tools Menu* — commands for setting compilation defaults (optimization level, etc.) and entering project configuration mode. Also used to change font in source file windows (default size is 12 point).
- *Help Menu* — commands to access on-line documentation for the SDK distribution and the GNUPro tools.

4.3 Unity Tool Bar

The Tool bar has synonyms for the most frequently used commands. The File menu synonyms are:

 *^f*

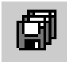
- *New Source File* — create a new source file. Equivalent to the File -> New command.

 *^o*

- *Open Source File* — open an existing source file. Equivalent to the File -> Open command.

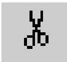
 *^s*

- *Save Source File* — save current source file. Equivalent to the File -> Save command.

 ^S

- *Save All Source Files* — save all modified source files. Equivalent to the File -> Save All command. This command saves both ipModule source files and project-specific source files.

The Edit menu synonyms are:

 ^x

- *Cut Selection* — delete selected text and save on the clipboard. Equivalent to the Edit -> Cut command.

 ^c

- *Copy Selection* — save selected text on the clipboard. Equivalent to the Edit -> Copy command.

 ^v

- *Paste Clipboard* — insert contents of the clipboard at the current selection point. Equivalent to the Edit -> Paste command.

 ^p

- *Print Selection* — print active source file. Equivalent to the File -> Print Setup command.

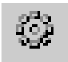
A Help menu synonym is:

- *About* — display copyright notice and version number. Equivalent to the Help -> About Unity command.

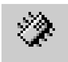
The Build menu synonyms are:

 F7

- *Compile Project* — invoke the GNUPro tool chain to compile the source files and create an executable file (**`.elf`** extension). Equivalent to the Build -> Compile command or the F7 function key. Only files which have changed or depend on files which have changed are recompiled. To force Unity to recompile every file, select the Build -> Clean command to delete all temporary files and directories used in the compilation, such as object files.

 F5

- *Debug Project* — starts a debugging session. Equivalent to the Build -> Debug command or the F5 function key. A suitable target such as the Demo Board must be available to start a debugging session.

 F4

- *Start Programmer* — launches the *IP2KProg* utility and immediately starts to program the IP2022. Equivalent to the Build -> Start Programmer command or the F4 function key.

- *Start Programmer GUI* — launches the IP2KProg utility and opens up a GUI, providing numerous programming options for the IP2022

 F9

- *Toggle Breakpoint* — sets or clears breakpoint at the insertion point in the current open source file. Equivalent to the Build -> Toggle Breakpoint command or the F9 function key.

The View menu synonyms are:

- *Toggle Workspace Window*— show or hide the Workspace window. Equivalent to the View -> Workspace command. The Workspace window displays a list of the source files in the current project. Double clicking on a file name opens that file for editing in a new window.

- *Toggle Output Window* — show or hide the Output window. Equivalent to the View -> Output command. The Output window displays the output from compiling a project, including error messages. The last line of the output will say either “Build succeeded” or “Build failed”.

- *Toggle Registers Window* — show or hide the Registers window. Equivalent to the View -> Registers command. The Registers window is used to view and edit the contents of the CPU and peripheral registers.

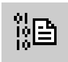
- *Toggle Memory 1 Window* — show or hide the Memory 1 window. Equivalent to the View -> Memory 1 command. The Memory 1 window is used to view the contents of an 80-byte region in program flash, program RAM, or data memory.

- *Toggle Memory 2 Window* — provided to view a second region of memory. Equivalent to the View -> Memory 2 command.

- *Toggle Watch Window* — show or hide the Watch window. Equivalent to the View -> Watch command. The Watch window shows the current values of user-specified expressions.

- *Toggle Stack Window* — show or hide the Stack window. Equivalent to the View -> Stack command. The Stack window lists the subroutines on the software stack.

One icon is not a synonym for any menu command:



- *Disassemble Window* — show source code interleaved with the corresponding assembly language instructions. To see any output in this window, start a debugging session.

- *Quick Watch Window* — provides the choice of either adding a variable to the watch window or updating/evaluating an expression.

^F2

- *Set Bookmark* — Set a bookmark at the current source line.

F2

- *Cycle Bookmark* — Cycle between bookmarks in the selected source window.

There are additional tool bar icons which are only enabled during a debugging session. These icons are discussed in Chapter 7.

4.4 Creating a New Project

New projects are started by copying one of the existing project templates. The **`starter`** example is a minimal Unity project which runs a binary counter on the bank of eight LEDs on the Demo Board and echoes characters received on the PC serial port back to the host PC. For applications using complex ipModule packages such as the Internet protocol stack, working demonstration programs are provided for use as templates.

To create a new project from the **`starter`** template:

1. *Enter Unity* — from the Windows program list, select the Ubicom -> Unity command. This opens the Unity IDE.

2. *From Main menu select Project->New*. A dialog box appears for entering the path name for the project file. Click on the “• ••“ button to browse. Use the standard windows dialog to create a new project directory and project file or to select an existing one.

   Do not open SDK template projects from Ubicom\SDK\projects\ directory. These projects are used by Unity as templates, during creation of new projects.

   Any legal full path name (with no embedded spaces) may be used. In this example, the project file name: **`C:\sdk_demo\starter\starter.c_c`**. Leave the project type selection as SDK. (Choosing General would create the new project file immediately, without copying any of the template or default files into the project.) Click the OK button.
3. *Select New Project Template* — in the list that is presented, click on **`starter`**. The other names in the list are demonstration programs for complex ipModule packages such as Ethernet communications. Click the OK button. Unity returns to the default view, but now the **`starter`** project is displayed in the Workspace window. At this point, the project file **`C:\sdk_demo\starter\starter.c_c`**has been created, and several files and directories have been copied to this directory.

4.5 Adding and Removing Files

To add a file to a project, select the Project -> Add file(s) command or press the Insert key when the workspace window is selected (active). Unity will automatically add references to the new file in the **app\Makefile.gen** file.

Note that only source files located in the project's **\app** directory will be compiled; e.g.
c:\sdk_demo\starter\app\main.c and
c:\sdk_demo\starter\app\isr.S.

To remove a file from a project, click the right mouse button over the file name in the Workspace window and select the Remove command from the pop-up menu, or press the Delete key and click the Yes button when Unity asks to confirm removing the file.

4.6 Project File Tree

This is the structure of the **C:\sdk_demo\starter** file tree, before compiling the project or after using a Build -> Clean command:

- **starter** — example project using the Demo Board
  - **app** — source files for the project
    - **main.c** — C source file
    - **isr.S** — ISR subroutines assembly file
    - **Makefile.inc** — source directory top makefile
    - **Makefile.gen** — list of project source files must be compiled
  - **config** — configuration files
    - **config.h** — macro definitions for C compiler

- **`config.mk`** — macro definitions for **`make`** utility
- **`starter.lpj`** — project configuration file

`Makefile` — top-level project makefile

`starter.c_c` — project file

5.0 Configuring ipModules

From the user's perspective, ipModule software consists of *packages* and *configuration points*. A package is a set of related functions, declarations, etc. that is managed as a group, such as the ipOS operating system or the TCP/IP Internet protocol stack. A configuration point is a customizable option within a package, such as the baud rate of a UART. A user-friendly graphical user interface (GUI) is used to select packages and customize configuration points.

The package selections and configuration point settings are saved in a *project configuration file* (**`.lpj`** extension). This information is used to generate the **`config.mk`** and **`config.h`** macro definition files, which control the inclusion of packages into a project during compilation.

5.1 Project Configuration GUI

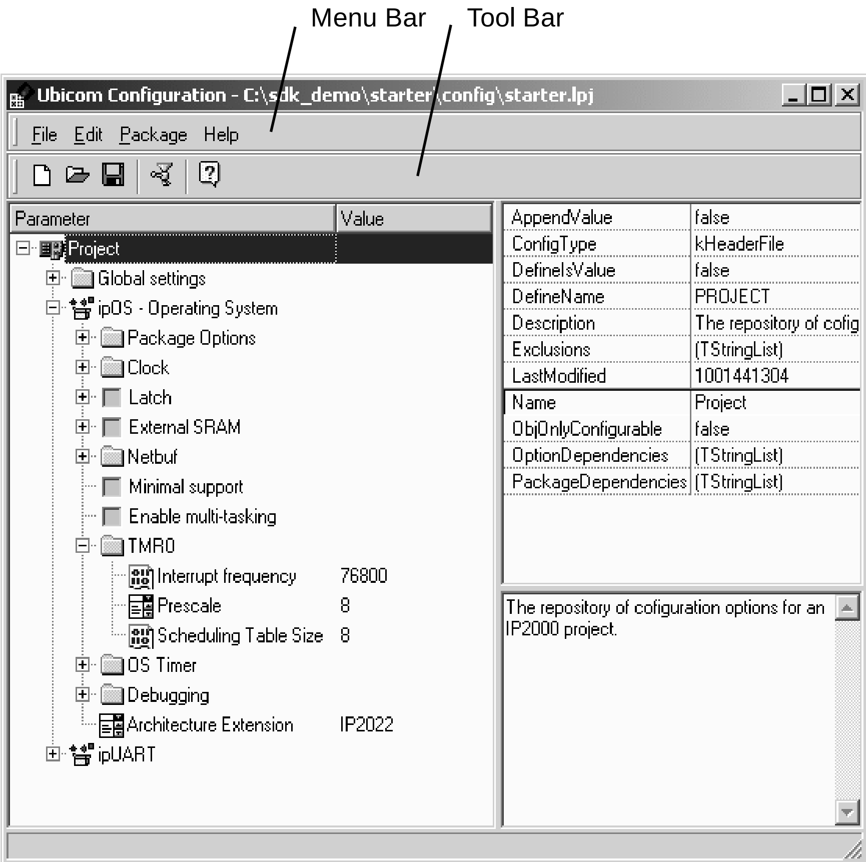


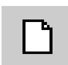
Figure 5-1 Project Configuration GUI

There are three panes in the window: the left pane shows the packages and configuration points in a project, the upper right pane shows symbolic definitions for configuration points, and the lower right pane shows comments. The right panes are protected against modification unless the Package -> Designer Mode command is used to disable protection. Most package users will only need to make changes that affect the left pane, such as toggling check boxes and entering constants into value fields.

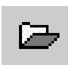
The menu bar at the top of the window may be used to access all of the project configuration commands. Frequently used commands are more conveniently accessed from the tool bar. The four menus on the menu bar are:

- *File* — opens, closes, and saves project files.
- *Edit* — removes a package from the project file (Edit -> Delete Node command).
- *Package* — adds and deletes packages from the project file, generates **`config.mk`** and **`config.h`** files, and specifies the directory for searching for packages.
- *Help* — provides access to on-line documentation.

The tool bar icons are synonyms for the most commonly used menu commands:



- *Create New Project* — create a new project file. Equivalent to the File -> New command.



- *Open Existing Project* — open an existing project file. Equivalent to the File -> Open command.

- *Save Current Project* — save changes to the project file. Equivalent to the File -> Save command.

- *Generate* — generate the **`config.mk`** and **`config.h`** files. Equivalent to the Package -> Generate command.

- *Help* — access to on-line documentation for Configuration Tool. Equivalent to the Help -> Help Contents command.

5.2 Generating Makefiles and Header Files

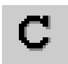
The following steps demonstrate using Configuration Tool to generate the **`config.h`** and **`config.mk`** files for the **`starter`** project.

1. *View Project Configuration* — select the Tools -> Configure command. Packages are added with the Package -> Add Package command. A package is removed by clicking on the package name and selecting the Edit -> Delete Node command.

2. *Generate Configuration Files* — either select the Package -> Generate command or click the Generate icon. If the files were generated, "Configuration generated successfully" will appear below the left pane. If there was a problem (e.g. a file cannot be found in the search path), a dialog box reports "Unable to create file".

6.0 Compiling

Compiling is a batch process which can be performed at any time, except during a debugging session. Compiling produces an executable file (**`.elf`** extension), which is run by downloading the file to a target such as the Demo Board.



The current project is compiled by clicking the Compile icon or hitting the F7 function key. *Changes to any open project source files are saved immediately, without requesting confirmation from the user.* During a compilation, the project source files are compiled with the GNU C compiler **`gcc`**, and the resulting object files are linked with the GNU linker **`ld`** to produce a **`.elf`** file. The command lines which invoke the GNUPro tools are visible in the Output window, but most users may ignore them. The important information displayed in the Output window are error messages that occur during compilation, which indicate the source file name and line number at which the error is indicated. The last line in the Output window will say either “Build succeeded” or “Build failed”.

The **`make`** utility is used to manage the compilation and linking of files, both to automate the process and to avoid unnecessary recompilation of source files. For example, if a project contains five source files but only one file is modified, it may be possible to generate a new executable file by recompiling only one source file and linking the resulting object file with four previously generated object files. If the modified source file is a header file included in the other four source files, however, it will be necessary to recompile all of the source files. The **`make`** utility keeps track of file dependencies and invokes the compiler and linker as needed to generate the files requested by the user.

To force **`make`** to recompile all files, select the Build -> Clean command to remove all temporary files, such as object files. Then, compile the project in the usual way.

A *makefile* is a command file for the **`make`** utility. **`Makefile`** in the project directory is the top-level makefile of the project. When compilation is triggered, Unity calls **`make`** to execute **`Makefile`**, as shown in Figure 6-1. **`Makefile`** calls two other makefiles, the **`config.mk`** file generated by Unity from the project configuration and the global makefile **`Makefile.inc`** in the **`sdk/scripts`** directory (one of the directories created by the SDK installation). The global makefile calls the **`Makefile.inc`** in the source files directory **`app`** and the **`Makefile.inc`** for each ipModule included in the project. All of the makefiles used to compile a project are either generated by Unity or supplied by Ubicom, so most users will not need to write or edit any makefiles.

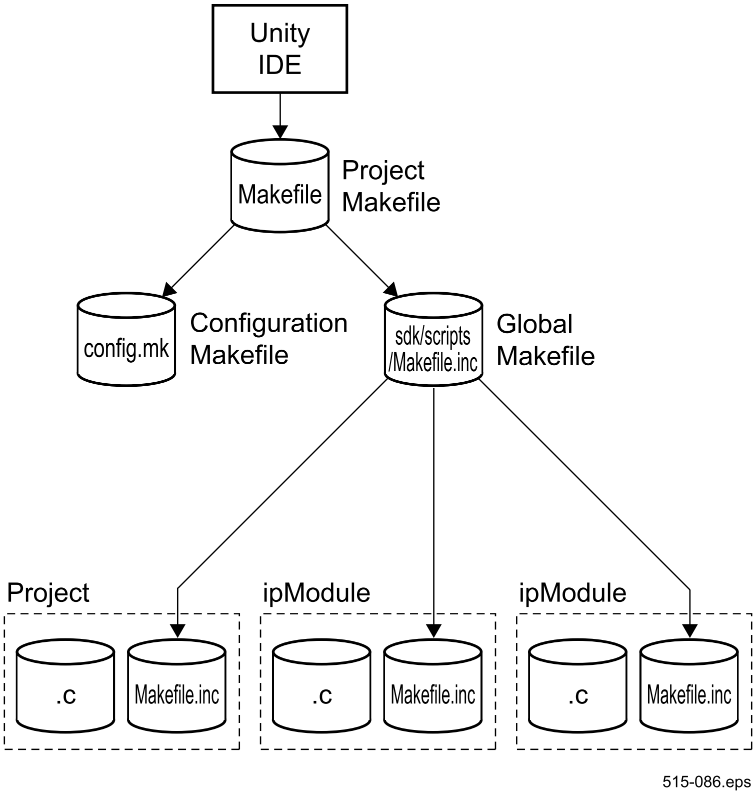
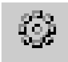


Figure 6-1 Makefile Structure

7.0 Debugging



A debugging session can be started if an executable file and a target are available. Click the Debug Project icon or press the F5 function key to start a debugging session. During a debugging session, the current project cannot be compiled, and another project cannot be opened.

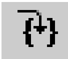
To exit from the debugging session while a program is running on the target, click the Stop icon once to stop the program and a second time to exit the debugger. If the target is already stopped, only one click is needed to exit the debugging session.

7.1 Debugging Tool Bar

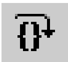
Several tool bar icons are only useful during a debugging session:

- *Toggle Breakpoint*—sets or clears breakpoint at the insertion point in the current open source file. Equivalent to the Build -> Toggle Breakpoint command or the F9 function key.

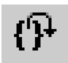
- *Continue*—start or continue program execution. Equivalent to the Build -> Continue command or the F6 function key.



- *Step*—advance program execution by one source code line. Equivalent to the Build -> Step command or the F11 function key.

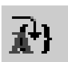


- *Next*—advance program execution by one source code line of the current function. Calls to other functions are executed without stopping until they return to the current function. Equivalent to the Build -> Next command or the F10 function key.



- *Up*—finish execution of the current function and stop after returning to the calling function. Equivalent to the Build -> Continue command or Shift plus the F11 function key.

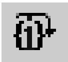
- *Reset*—reset target. Equivalent to the Build -> Reset command.



- *Automated Step*—like the Step icon, but with automatic repeat a little faster than once per second. Click once to begin repeat mode, and click a second time to end it.

- *Automated Next*—like the Next icon, but with automatic repeat a little faster than once per second. Click once to begin repeat mode, and click a second time to end it.

- *Assembler Step*—advance program execution by one instruction. Click the Disassemble Window icon to view instructions.



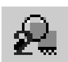
- *Assembler Next*—advance program execution by one instruction. Calls to other functions are executed without pausing until they return to the current function.

- *Stop*—if a program is running on the target in a debugging session, stop program execution. Otherwise, terminate debugging session. A debugging session must be terminated to recompile the project. Equivalent to the Build -> Stop command.

- *Quick Watch*—add expression to Watch window.

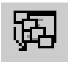
- *Toggle Registers Window*—show or hide the Registers window. Equivalent to the View -> Registers command. The Registers window is used to view and edit the contents of the CPU and peripheral registers.

- *Toggle Memory 1 Window*—show or hide the Memory 1 window. Equivalent to the View -> Memory 1 command. The Memory 1 window is used to view the contents of an 80-byte region in program flash, program RAM, or data memory.



- *Toggle Memory 2 Window*—provided to view a second region of memory. Equivalent to the View -> Memory 2 command.

- *Toggle Watch Window*—show or hide the Watch window. Equivalent to the View -> Watch command. The Watch window shows the current values of user-specified expressions.

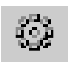


- *Toggle Stack Window*—show or hide the Stack window. Equivalent to the View -> Stack command. The Stack window lists the subroutines on the hardware call/return stack.

- *Disassemble Window*—show source code interleaved with the corresponding assembly language instructions. No output is shown in this window until one of the Step icons is used.

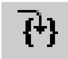
7.2 Viewing Program Execution

For the following procedure, it is assumed that a target and an executable file for the current project are available.

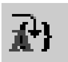


1. *Enter the Debugger*—click the Debug Project icon or press the F5 function key. Unity will connect to the target system. In the default configuration, Unity then downloads and runs the executable file on the target. To change the default behavior of Unity, select the Tools -> Options command, then select the Debugger tab. A binary counter begins incrementing on the bank of eight LEDs on the Demo Board.

2. *Stop the Program*—click the Stop icon to halt execution. Unity opens a source file window for the line which was about to be executed, which is highlighted in color in the center of the source file window.



3. *Advance Program Execution*—click the Step icon several times to advance program execution line-by-line. Because the **`starter`** application calls functions in several source files, the debugger frequently jumps among these files while stepping through code.



   Click the Automated Step icon to watch program execution continue at a speed slightly faster than one line per second. Code in the source file **`main.c`** frequently calls code in **`uart_vp.c`** and **`timer.c`**, the UART and timer ipModules, so the view jumps among these source file windows. Exit this mode by clicking the Automated Step icon a second time.

   Click the Automated Next icon. Next differs from Step in that functions are executed without pausing until they return to the calling function. In this mode, functions return to the main loop, and the main loop stays in the front window. Exit this mode by clicking the Automated Next icon a second time.

4. *Open Disassemble Window*—click the Disassemble Window icon to open a new window for displaying source lines interleaved with their corresponding instructions.
5. *Advance Program Execution*—the Disassemble window does not display any output until program execution is advanced, either by incremental advances with a command like Step or Next or by hitting the Continue icon followed by hitting the Stop icon or encountering a breakpoint.

7.3 Breakpointing Program Execution

For the following procedure, it is assumed that a debugging session has been entered, as described in Section 7.2. A breakpoint will be set in the **`led_callback`** routine which changes the pattern of lights on the LED bank of the Demo Board. This routine is called by a one-shot timer when it expires. To ensure that the call is repeated, one of the functions performed by the routine is to reattach itself to the list of timers serviced by the ipOS operating system.

1. *Reset the Target*—click the Reset icon to stop program execution on the target and reset the program counter.

2. *Browse for the **`led_callback`** Function*—click the Browse tab in the Workspace window. If the Browse tab is empty, click the Stop icon to exit debugging mode, click the Compile icon to recompile the project, click the Debug Project icon to re-enter the debugging session, and click the Reset icon to stop execution. Immediately after recompiling, the Browse tab displays the functions of the **`starter`** project, as shown in Figure 7-1.

Figure 7-1 Browse Tab

Double-click on **`led_callback`** to open the source file **`main.c`**. The first line of the **`led_callback`** function will be highlighted in the source file window.

3. *Set a Breakpoint*—click on the line that says **`(*led)++;`**. Click the Toggle Breakpoint of icon to set a breakpoint. A red circle appears in the gray bar along the left edge of the source file window to indicate a breakpoint has been set, as shown in Figure 7-2. Unity supports Multiple breakpoints in program flash memory. To enable or disable multiple break points Use menu->Tools->Options->Debugger tab page. Any number of breakpoints can be set in program RAM. Select the View -> Breakpoint List command to examine the breakpoints which have been set.

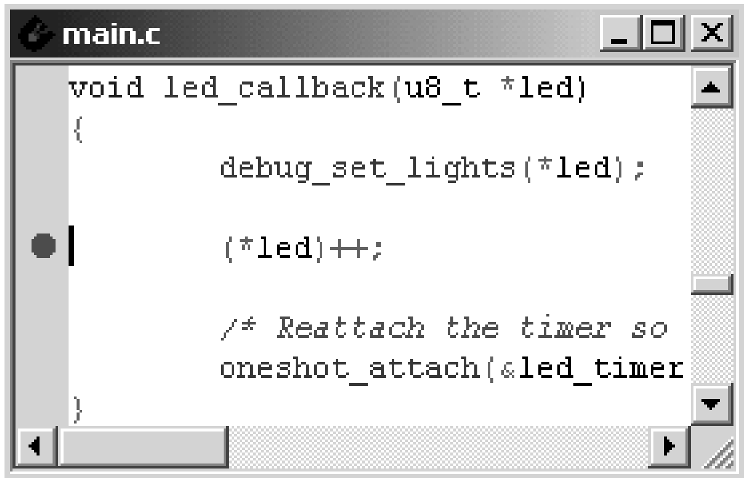


Figure 7-2 Setting a Breakpoint

4. *Continue Program Execution*—click the Continue icon to begin program execution. Immediately, the program will stop at the breakpoint. Continue clicking the Continue icon. For each click, the pattern on the LED bank is incremented.
5. *Examine Stack Window*—the Stack window is a read-only display of the subroutines which have been pushed on the hardware stack. With the **`starter`** example stopped in **`led_callback`**, the stack window appears as shown in Figure 7-3.

Figure 7-3 Stack Window

`main` is the top-level function of the **`starter`** example, so it is the deepest function on the stack. **`timer_poll`** is called in the main loop of **`main`**, to update any timers that ipOS is maintaining. **`timer_poll`** calls **`oneshot_tick`** to update any one-shot timers. If the timer for incrementing the pattern of lights on the LEDs has expired, **`oneshot_tick`** calls **`led_callback`**.

7.4 Viewing Registers

To view the registers of the **`starter`** example as it executes:

1. *Scroll to **`rbin`** and **`rbout`***—scroll down the Registers window until the **`rbin`** and **`rbout`** registers are displayed, as shown in Figure 7-4. **`rbout`** is the output register for the port which drives the LED bank, and **`rbin`** is the input register for that port.

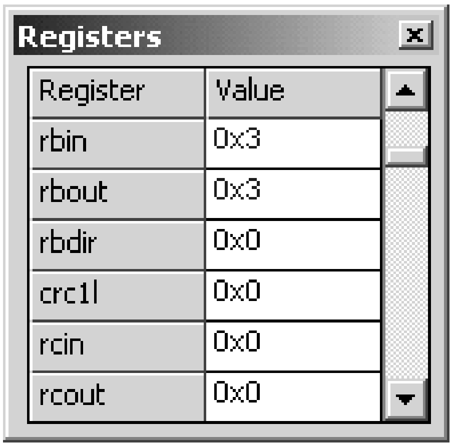


Figure 7-4 Registers Window

2. *Continue Execution*—click the Continue icon to run the program until the breakpoint is hit again. When a register value changes between steps, it is highlighted with color in the Registers window. The **`rbout`** register is incremented and

read back into the **`rbin`** register on every click of the Continue icon, so the values in these registers are highlighted after each click.

7.5 Viewing Memory

The expression **`*led`** is a pointer to data driven on the LED bank. The Watch window can be used to display both the pointer and the data.

1. *Specify Watch Expressions*—click the Quick Watch icon to bring up a box for specifying a watch expression.

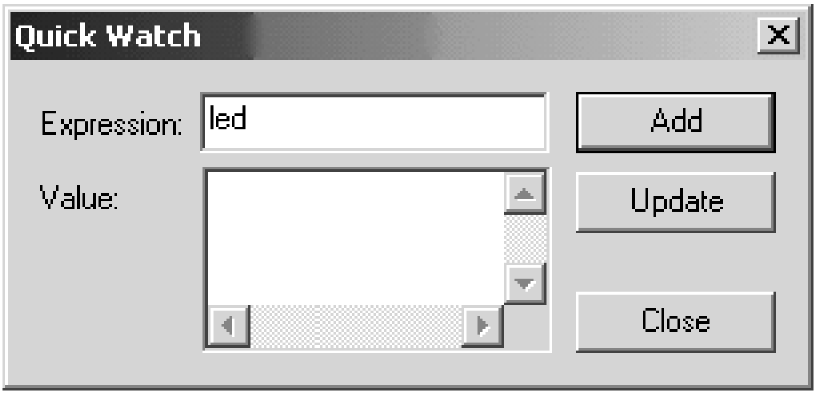


Figure 7-5 Quick Watch Box

Enter **`led`** and click the Add button. Then, click the Quick Watch icon again, enter **`*led`**, and click the Add button. At this point, both **`led`** and **`*led`** have been added to the Watch window, as shown in Figure 7-6.

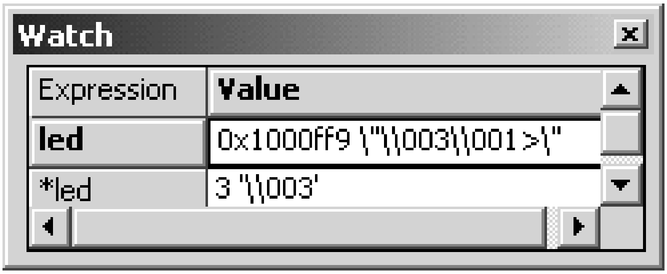


Figure 7-6 Watch Window

The value in **`led`** is 0x1000FF9, which is the address of the data that **`starter`** loads in the **`rbout`** register. The addresses used by the debugger have a different mapping than those used in source files, as discussed in Section 7.6. The value in **`*led`** shows the data itself. Click the Continue icon to increment the data.

2. *Examine Memory*—click the address box in one of the Memory windows, and change it to 0x01000FB0. A Memory window presents a fixed display of 80 characters, and it will not display any characters if one of them is in a reserved section of the memory space. Because a reserved section starts at 0x01001000, the address 0x01000FF9 cannot be used to view memory. However, the reserved section can be avoided by entering 0x01000FB0 into the address box, as shown in Figure 7-7. The data at address 0x01000FF9 is 03 in the second data column on the last line.

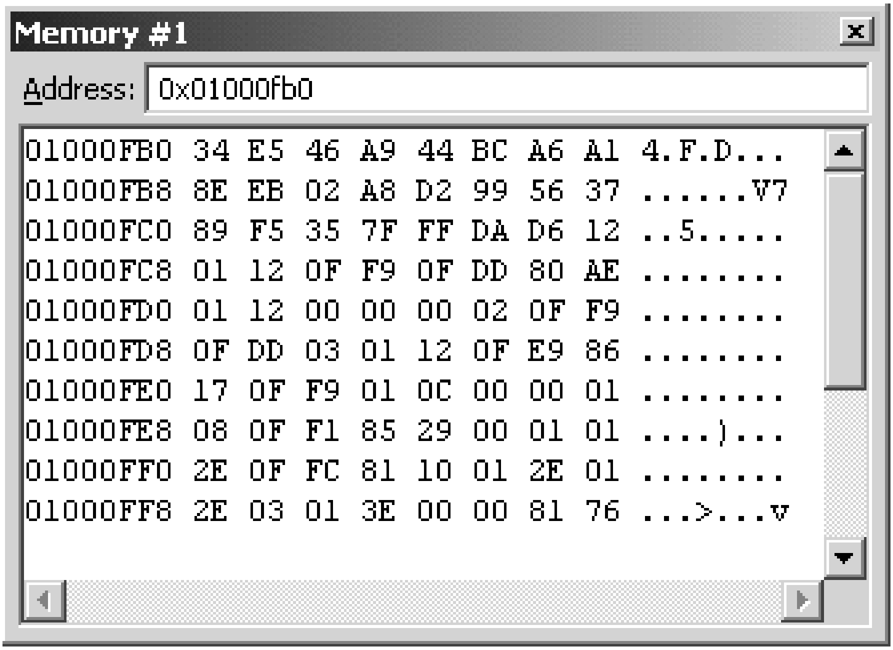


Figure 7-7 Memory Window

ASCII character equivalents of the data are shown in the column on the right.

Note that local variables cannot be inspected (shown) in a watch window when a breakpoint has been hit somewhere outside the function or block where there those variables are defined. In this case Unity displays "error" instead of a variable value.

Note also that some local variables (depending on a particular application’s or function’s complexity) cannot be shown in a watch window because of compiler optimization. This is not a bug; it is normal behavior that can be observed in many other development systems that use compilers with a high level of optimization. In order to view such variables, you may chose declare them temporarily as globals or give them the attribute volatile or static.

7.6 Memory Map

A map of the memory space as viewed through the debugger is shown in Figure 7-8. This address mapping is only used in the **`.elf`** files generated by the linker and analyzed in the debugger. Do not use these addresses in C source files.

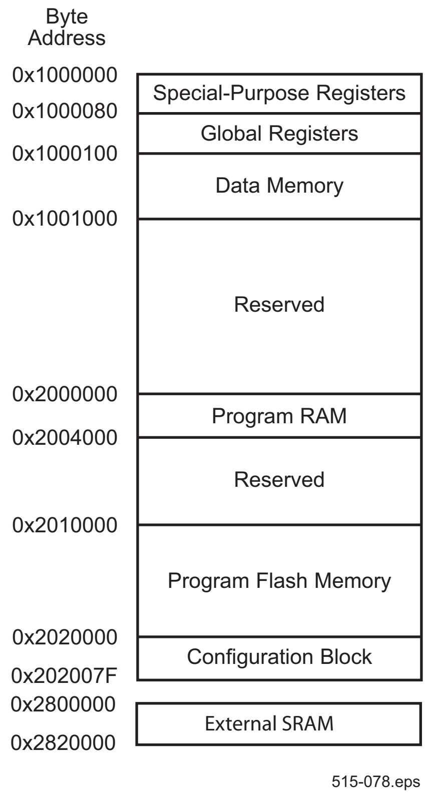


Figure 7-8 Debugging Memory Space

8.0 Utilities

An executable file (`.elf` extension) can be downloaded to a target using the `IP2KProg` utility. `IP2KProg` can be used to:

- Read the executable sections from a `.elf` file to an internal buffer.
- Read the executable code from the flash memory of a programmed IP2022 device to an internal buffer.
- Save the buffer contents into a binary file.
- Edit the buffer in hexadecimal format.
- Download the software to an IP2022 device flash memory.

After downloading, `IP2KProg` resets the IP2022, and the software begins running.

From within Unity, `IP2KProg` is launched by clicking the Start Programmer icon, selecting the Build -> Start Programmer command, or pressing the F4 function key. `IP2KProg` loads the executable file of the current project automatically. An executable file must be available, otherwise Unity will not launch `IP2KProg`. Only one copy of `IP2KProg` is allowed in PC memory; all subsequent clicks on the Program button will just activate the existing `IP2KProg` window. Upon initial activation, `IP2KProg` automatically reloads the previously loaded file. Although `IP2KProg` does not interfere with the debugger as long as it is not doing any program/read operations, it is recommended to close **ip2kprog** while in a debug session.

`IP2KProg` can also be run as a standalone program by selecting the ubicom -> **Ip2kprog** command. In this case, **`IP2KProg`** is launched with a blank memory buffer.

Do not download a blank buffer to the Demo Board.

Be careful to avoid overwriting configuration block settings in a way that prevents the IP2022 device from functioning. In particular, the 4.8 Mhz crystal oscillator is the main clock source available on the Demo Board. If the XTAL bit disables the OSC oscillator, the Demo Board will not be operational and any further attempts to program the IP2022 device will be unsuccessful until an external clock source is provided from the external U9 CAN oscillator that is optional for the board.

8.1 IP2KProg User Interface

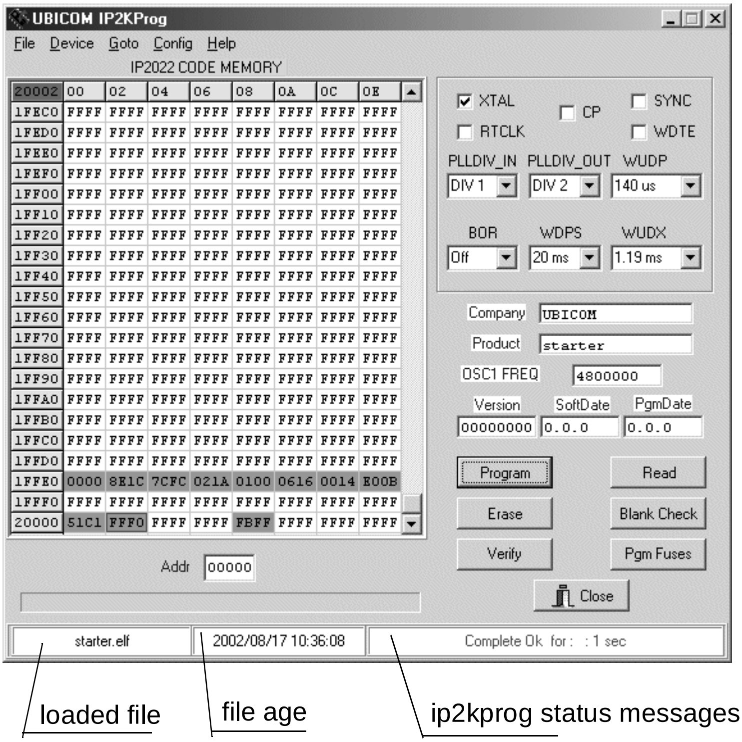


Figure 8-1 IP2KProg User Interface

`IP2KProg` keeps a memory buffer of the IP2022 program memory and configuration block. A hexadecimal listing of this memory buffer is shown on the left. Addresses are listed as word addresses. The data in the memory buffer can be edited by typing over any previous values.

The configuration block settings are shown to the right of the program memory space. *Be sure to check the XTAL box to enable the OSC oscillator before downloading to the Demo Board.*

The memory buffer can be downloaded to both the program memory and the configuration block by clicking the Program button. Clicking the Program button performs the following operations:

- Erases all FLASH pages and Configuration block
- Programs FLASH pages and Configuration from buffer
- Verifies Flash and Configuration block with Buffer

There is no need to use Erase and Verify buttons under normal conditions.

Clicking the PgmFuses button does following:

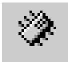
- Erases the Configuration Block only
- Programs the Configuration Block from Buffer

The memory buffer can be loaded by selecting the File -> Load command and browsing for a **`.elf`** file. The buffer also can be loaded from a programmed IP2022 device by clicking the Read button. A list of the regions of memory loaded from the **`.elf`** file can be viewed by selecting the Device -> Flash_allocation command.

To navigate around the memory buffer, use the scroll bar. Use the Goto menu to jump to the beginning of any memory block or the configuration block.

8.1.1 Downloading a Project

To download the **starter** example:



1. *Launch **IP2KProg*** — click the Start Programmer icon. When launched from Unity, the executable file **starter.elf** is automatically loaded into the memory buffer.
   To run **IP2KProg** as a standalone program, select the Ubicom -> Ip2kprog command to launch **IP2KProg**. Then, select the File -> Load command to browse for the **starter.elf** file.
   Select the Goto -> Flash_0 command to navigate to the beginning of the flash memory at word address 0x08000. The memory locations loaded from the **starter.elf** file are highlighted in green.
2. *Download the Executable File* — click the Program button to download the contents of the memory buffer to the IP2022 device. After downloading, **IP2KProg** resets the target and the software begins running. The pattern of lights on the LED bank begins incrementing. If a serial cable is connected, a serial communications program such as HyperTerminal can be used to demonstrate that characters are being echoed by the Demo Board.
3. *Verify the Download* — click the Verify button to check that the download was successful. To provoke a mismatch during verification, edit a memory location and click the Verify button again. A mismatch report appears in a new window. Press a keyboard key to close the mismatch report window.
4. *Exit **IP2KProg*** — click the Close button.

8.2 GNU Binary Utilities

The following utilities are available from an MS-DOS command prompt window:

- *ip2k-elf-objdump* — with the **`-D`** argument, produces a disassembly listing. With the **`-x`** argument, prints section information (size, load address, etc.).
- *ip2k-elf-readelf* — with the **`-a`** argument, lists **`.elf`** file header information, section header information, and the symbol table.
- *ip2k-elf-size* — lists the size of each section.
- *ip2k-elf-nm* — lists the symbol table.
- *ip2k-elf-strings* — lists any ASCII strings found in the file.

To open an MS-DOS command prompt window, select the MS-DOS Command Prompt command from the Start menu. A new window will open for submitting MS-DOS commands. The **`ip2k-elf-objdump`** and **`ip2k-elf-readelf`** commands require an argument. For a brief summary of the available arguments, try the command without an argument. For detailed information about the available arguments, see the binutils section of the *GNUPro Toolkit—GNUPro Utilities* manual (pages 317 to 368).

Some of these utilities produce output that scrolls off the window. The output can be run through the MS-DOS **`more`** utility to print only as much output as can be seen in the window, as shown in the following command line.

```
ip2k-elf-objdump -D starter.elf | more
```

Press the space bar to advance the output by one window of information.

The output can also be directed to a file so that it can be viewed in an editor. For example, the following command line places the output in the file **`tmp.txt`**.

```
ip2k-elf-objdump -D starter.elf > tmp.txt
```

9.0 SDK Demo Projects

This manual is intended for users who have already exercised the IP2022 development tools, used the IP2022 Primer, and have built the starter project using the IP2022 Primer. The user should have experience in changing the PC's IP settings. The following demos are presented in this chapter:

- Starter
- Telnet
- Webserver
- Serial Gateway
- SNMP Compiler ipOS
- USB RNDIS Device

9.1 Equipment Required

The following is required to run all the demos:

- IP2022 Demo Board
- IP2022 Native Ethernet Daughter Board
- Parallel Port ISD/ISP Cable
- DC Power Supply
- Serial Straight-Through Cable
- CAT 5 Ethernet Crossover or Straight-Through Cable

The foregoing items are included in the IP2022 Universal Device Networking Kit. In addition, a number of the demo projects either require or can be used with other daughtercards, available separately. The USB RNDIS Device demo requires a USB daughtercard (part # IP2K-FSP-USBDEV00) for use on the IP2022 Demo Board. While most ethernet-based projects are typically used with the IP2022's on-chip MAC/PHY ethernet capability via the included ipEthernet native daughtercard, external ethernet controller daughter cards are available for 10/100 (part # IP2K-FSP-ASIXDC) and 10baseT (part # IP2K-FSP-RT8019AS) ethernet.

9.1.1 PC Network Configuration

All Ethernet-based demos can be run either with the Demo Board connected directly to the PC or with the Demo Board connected to the same LAN as the PC through a hub or switch. To connect the Demo Board directly to the PC, use the grey crossover CAT5 Ethernet cable. To connect the Demo Board to a LAN, use the red straight-through CAT5 Ethernet cable.

The Ethernet-based demos come preconfigured with IP address 192.168.1.24 and subnet mask 255.255.255.0. If the Demo Board is directly connected to the PC, you only need to ensure that the PC has another IP address on the same subnet (192.168.1.2, for example). When the demos are connected to a hub or switch, the PC IP address and settings can be assigned by DHCP. If the Demo Board's preconfigured IP address 192.168.1.24 is on the same subnet as the PC and does not conflict with another device, the demos can be used without modification; otherwise, you need to configure the IP2022 Demo Board with a free IP address as described in Section 9.1.2.

9.1.2 Configuring IP2022 Board IP Address

IP2022 demo projects that use the ipEthernet software module may require an IP address change in order to work when connected through a hub to a LAN. IP addresses can be changed from the Configuration Program. Figure 9-1 shows how to change the Demo Board's IP address to 192.168.100.128, just as an example; you must choose an IP address that works with your LAN. After the IP address is changed, the configuration should be saved and regenerated.

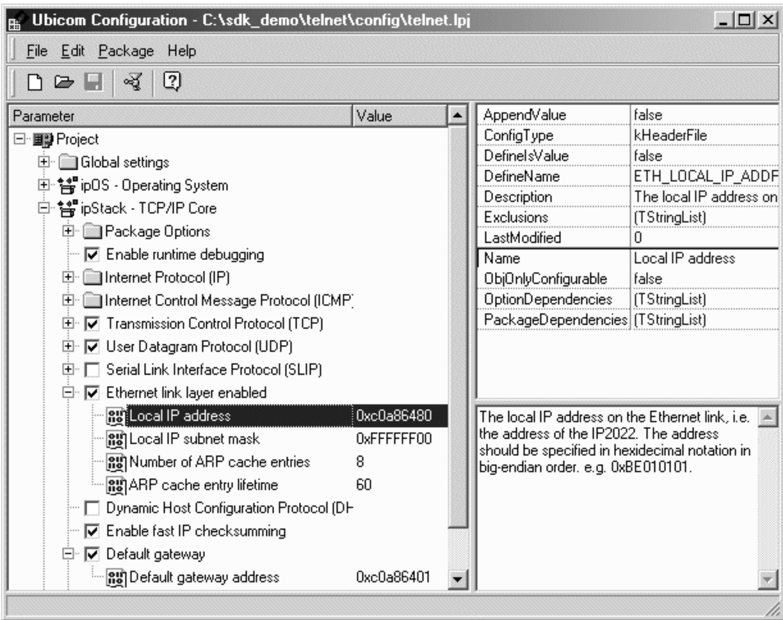


Figure 9-1 Configuring Demo Board IP Address

9.1.3 Network Tips

Use the ping utility to check that the PC can see the IP2022 Demo Board when a network demo project is running. Use the ipconfig utility to check that your PC has the correct IP settings.

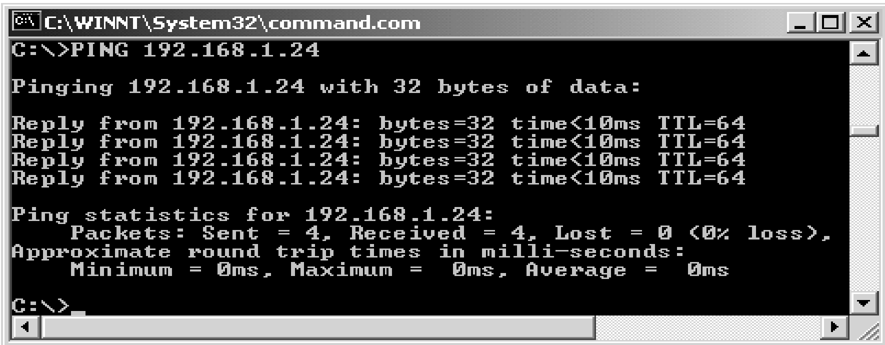


Figure 9-2 Ping

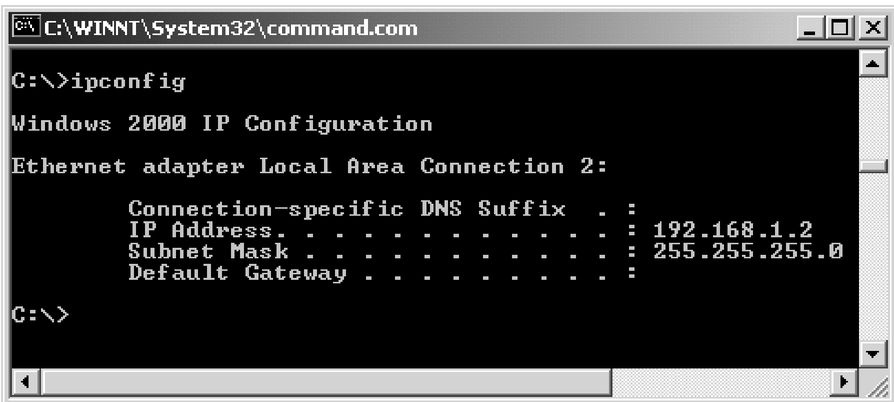


Figure 9-3 Ipconfig

9.1.4 Demo Jumper Settings

The Demo Board can run all the demo projects with the factory jumper settings:

- JP14, JP15, JP16, JP17 (power supply)
- JP10 (IOVDD to 3.3V)
- JP13 (OSC)
- JP7 (LEDs On)
- JP12 (Uart En)
- All others removed.

9.2 Starter Demo

The starter demo project is a fundamental SDK project. It includes the basic building blocks for a general-purpose SDK project. This demo shows the ipOS operating system one-shot timer implementation and the serial UART driver implementation. The one-shot timer is used to increment a binary counter on the LED bank. The UART is used to echo characters sent to the IP2022 back to the PC terminal program.

SDK ipModules include:

- ipOS
- ipUART

Project files include:

- isr.S
- main.c

9.2.1 IP2022 Board Configuration

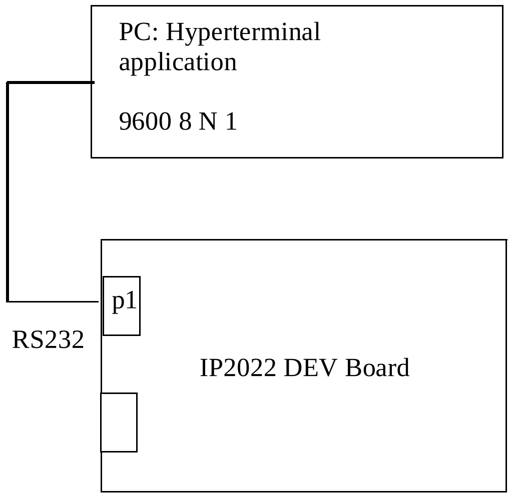


Figure 9-4 Starter Demo Board Diagram

Jumper Configuration: as defined in Section 9.1.4.

Cables:

- Straight-thru serial (from "TO PC" to PC's serial communications port)
- Power
- Parallel ISD/ISP

Daughter Boards:

- None

9.2.2 PC Configuration

Configure and launch a Terminal program. The Terminal program should have the followings settings:

- 9600 baud
- 8 bit ASCII
- No parity
- 1 stop bit
- No flow control
- Local echo off

9.2.3 Demo Project Configuration and Build

- Launch Unity.
- Create a new project. Project->new

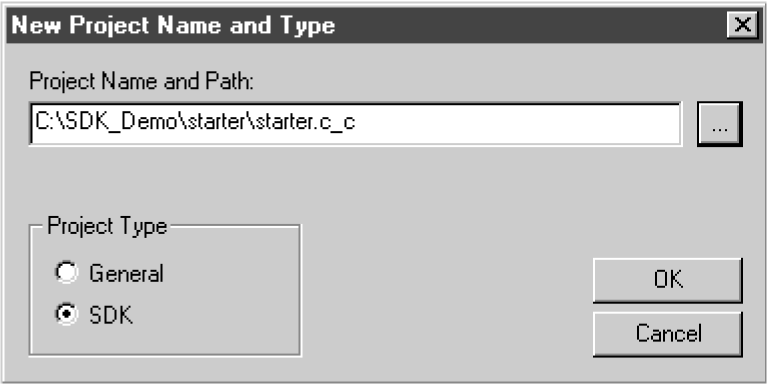


Figure 9-5 Set New Project Name

- Choose the project template.

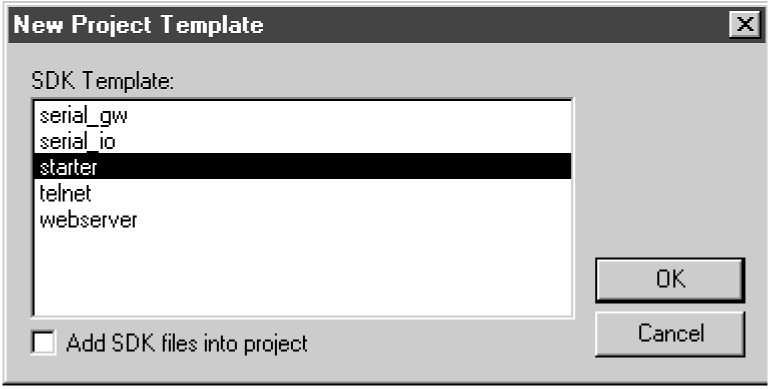


Figure 9-6 Choose Project Template

- Compile the project. Build->Compile
- Download the project. Build->Debug
- Run the project. Click the "Continue" button.

9.2.4 Using the Demo

Once the project starts running, the LED bank displays a binary counter. Characters typed in the terminal program are echoed back by the IP2022 UART.

9.3 Telnet Demo

This project shows an IP2022 based monitor application, accessed through a Telnet session. The Telnet session demonstrates the ipStack and ipEthernet. The monitor application provides information about the ipOS and the IP2022's network connection.

SDK ipModules include:

- ipOS
- ipIO
- ipStack
- ipEthernet

Project files include:

- isr.S
- main.c
- telnet.c
- udp_ping.c

9.3.1 IP2022 Board Configuration

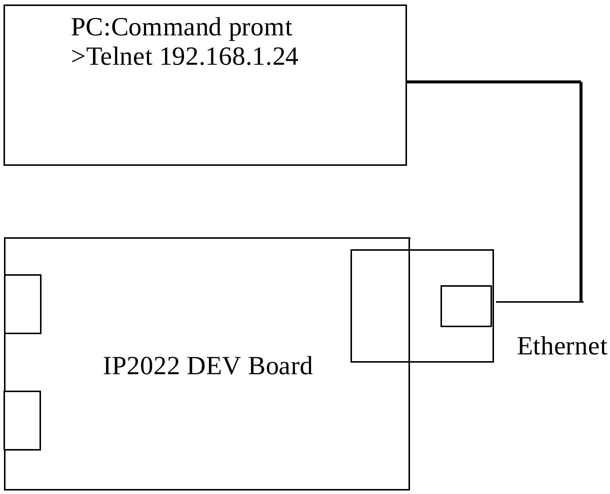


Figure 9-7 Telnet Demo Board Diagram

Jumper Configuration: as defined in Section 9.1.4.

Cables:

- CAT 5 Ethernet Crossover
- Power
- Parallel ISD/ISP

Daughter Boards:

- IP2022 Native Ethernet Daughter Board (inserted in J3, SERDES2)

9.3.2 Demo Project Configuration and Build

- Launch Unity.
- Create a new project. Project->new
- Select the telnet template.
- Compile the project. Build->Compile
- Download the project. Build->Debug
- Run the project. Click the "Continue" button.

9.3.3 Using the Demo

From the Command Prompt or a DOS window, type telnet 192.168.1.24. Type h to display the IP2022 Telnet Monitor options.

Type a for a display of "Allocated heap (memory) chains"

This displays a list of memory blocks allocated in the heap. It shows the address, size, associated ipModule package and the type of memory block.

addr = Starting address in data ram
size = Size in bytes
pkg = Associated ipModule package. Refer to heap.h in SDK
type = Refers to the type of memory block. Refer to heap.h in SDK

Type f for "Free heap (memory) chains"

This displays a list of free memory blocks in the heap. It shows the total free, total heap size, address of block and size of block.

Free memory = Total free memory
of total heap =Total heap size
addr = Address in data RAM
size = Size of free block

Type h for "Monitor options"

Type l for "Load averages"

Not active in the default demo project.

Type n for "Netpage information"

Netpage free = Number of free netpages.
Low water mark = Lowest number of free netpages. As shown in the example below, 47 netpages has been the lowest number of netpages free. Zero indicates the program is out of memory for netpages.

Type o for "One-shot timers"

addr = Address in data RAM
ticks left = Number of ipOS ticks left until expiration.

Type q to "Quit telnet session"

Type s for "TCP sockets"

This displays all active TCP sockets and the listening for new TCP socket requests.

addr = Address in data RAM
local = IP address of IP2022 (in HEX)

remote = IP address of connected device, PC's address (in HEX)
state = TCP connection state. Refer to tcp.h in SDK

Type u for "UDP sockets"

This displays all active UDP sockets and the listening for new UDP socket requests.

addr = Address in data RAM
local = IP address of IP2022 (in HEX)
remote = IP address of connected device, PC's address (in HEX)
state = UDP connection state. Refer to udp.h in SDK

9.4 Web Server Demo

This demo shows the IP2022 being used as a Web server. It has static HTTP resources, including graphics and HTML pages. These pages are stored in the internal flash. Dynamic Web page creation is based on query submission from HTTP client (ipOS Status and IP2022 Game sub-pages). The ipOS status monitoring application outputs Web pages showing similar information as the Telnet demo. The LED Game demonstrates dynamic control of IP2022 external I/O through HTTP queries.

SDK ipModules include:

- ipOS
- ipIO
- ipStack
- ipWeb
- ipFile
- ipEthernet

Project files include:

- cgi.c
- isr.S
- main.c

9.4.1 IP2022 Board Configuration

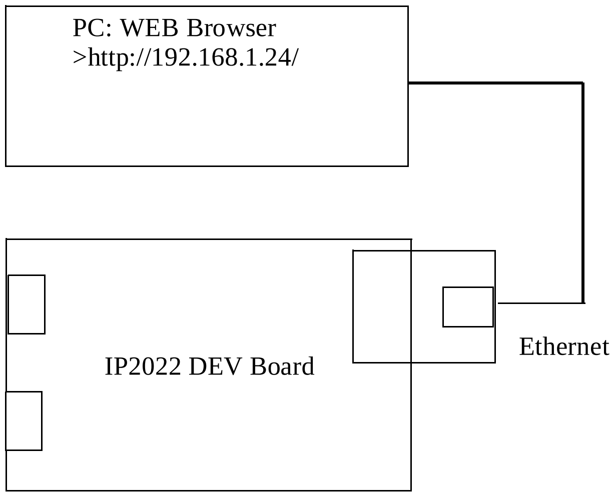


Figure 9-8 Web Server Demo Board Diagram

Jumper Configuration: as defined in Section 9.1.4.

Cables:

- CAT 5 Ethernet Crossover
- Power
- Parallel ISD/ISP

Daughter Boards:

- IP2022 Native Ethernet Daughter Board (inserted in J2, SERDES1)

9.4.2 Demo Project Configuration and Build

- Launch Unity.
- Create a new project. Project->new.
- Select the webserver template.
- Compile the project. Build->Compile.
- Download the project. Build->Debug.
- Run the project. Click the "Continue" button.

9.4.3 Using the Demo

- Open your Internet browser and enter **`http://192.168.1.24/index.html`** in the address bar.
- Click on ipOS Status and a new page is displayed with the same type of functionality as the Telnet demo.
- Click on LED Games and a new page is displayed. This page allows the user to play some games on the IP2022 Demo Board's LED bank. The LED game works only if, in project configuration under "Debugging" mode, the "Use Debug LED" checkbox is checked.

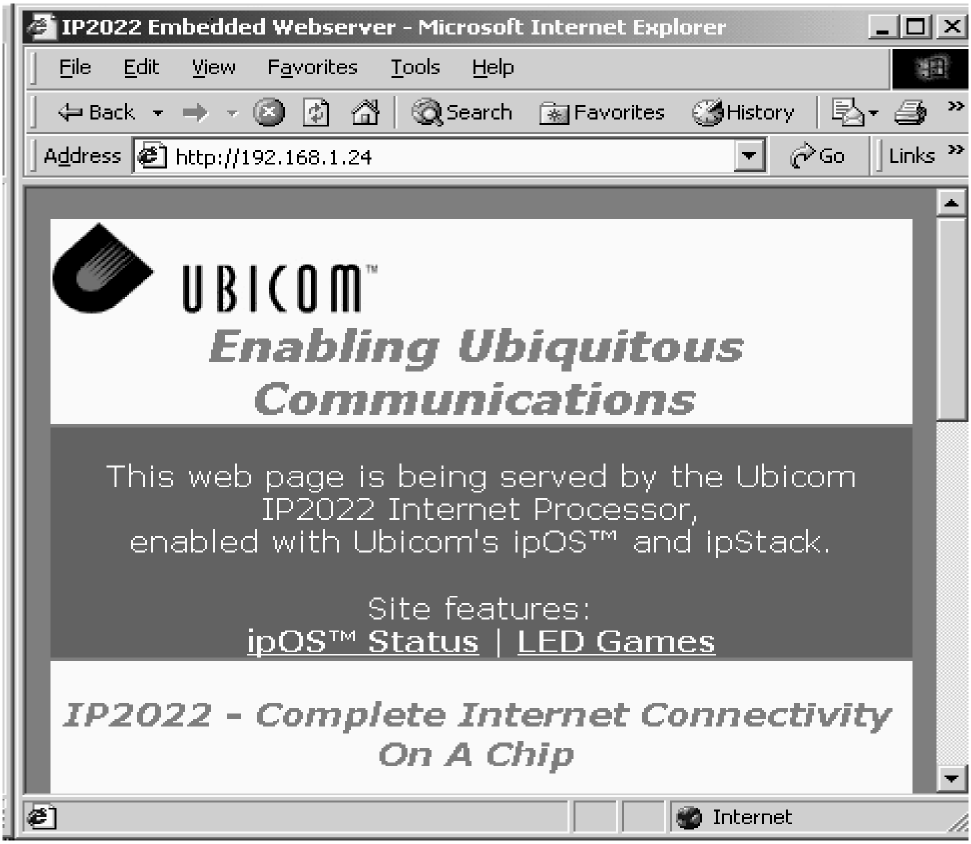


Figure 9-9 Browser Window

9.5 Serial Gateway Demo

This demo project shows a RS232 UART to Ethernet Bridge. Characters typed in the PC terminal program are sent to the IP2022 UART. The IP2022 UART passes the characters to the IP2022 Telnet Application. The IP2022 Telnet Application sends the characters down the ipStack and out Ethernet. The PC receives the Telnet Packet and the PC Telnet program displays the characters. Vice versa occurs for Telnet typed characters.

SDK ipModules include:

- ipOS
- ipIO
- ipStack
- ipEthernet

Project files include:

- bridge.c
- isr.S
- main.c
- telnet.c

9.5.1 IP2022 Board Configuration

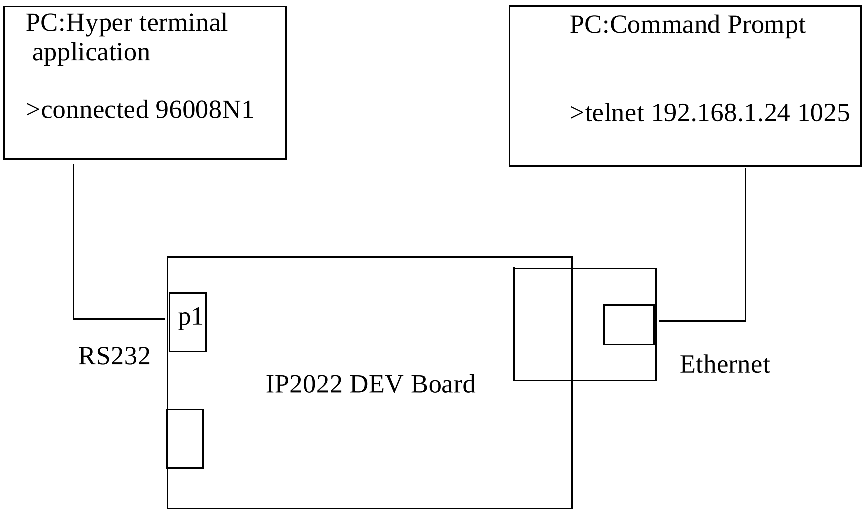


Figure 9-10 Serial Gateway Demo Board Diagram

Jumper Configurations: as defined in Section 9.1.4.

Cables:

- Straight-thru serial (from "TO PC" to PC's serial communications port)
- CAT 5 Ethernet Crossover
- Power
- Parallel ISD/ISP

Daughter Boards:

- IP2022 Native Ethernet Daughter Board (inserted in J2, SERDES1)

9.5.2 PC Configuration

Configure and launch a terminal program. The terminal program should have the followings settings:

- 9600 baud
- 8 bit ASCII
- No parity
- 1 stop bit
- No flow control
- Local echo off

9.5.3 Demo Project Configuration and Build

- Launch Unity.
- Create a new project. Project->new
- Select the serial_io template
- Compile the project. Build->Compile
- Download the project. Build->Debug
- Run the project. Click the "Continue" button.

9.5.4 Using the Demo

Once the project is running, launch a DOS window and type

`telnet 192.168.1.24 1025.`

Type characters in the Telnet window to see them displayed in the Terminal program.

Type characters in the terminal program to see them displayed in the Telnet window.

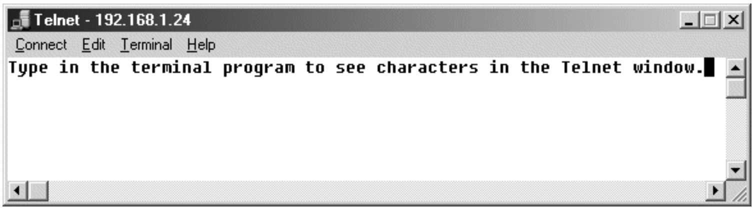


Figure 9-11 Telnet Window

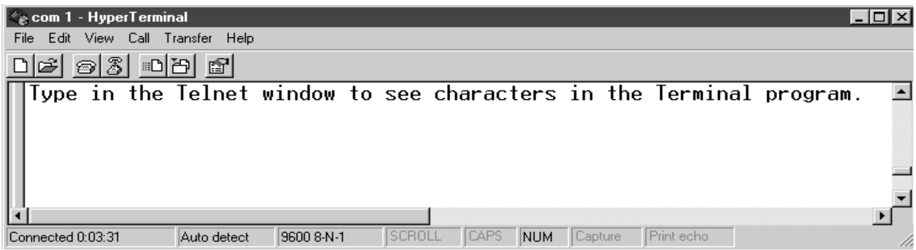


Figure 9-12 Hyperterminal Window

9.6 SNMP Compiler ipOS Demo

The ipManage package of the SDK provides support for implementing SNMP (version 1) agents. The package contains code for agent operations as defined in RFC1157 (A Simple Network Management Protocol) and the encoding of information as defined in RFC1155 (Structure and Identification of Management Information for TCP/IP-based Internets). It also contains a MIB compiler, smidump, which can be used to generate most of the code required for an agent.

This example project shows how the ipManage package can be used to implement an SNMP agent that manages the values of certain data objects of the ipOS operating system.

9.6.1 IP2022 Board Configuration

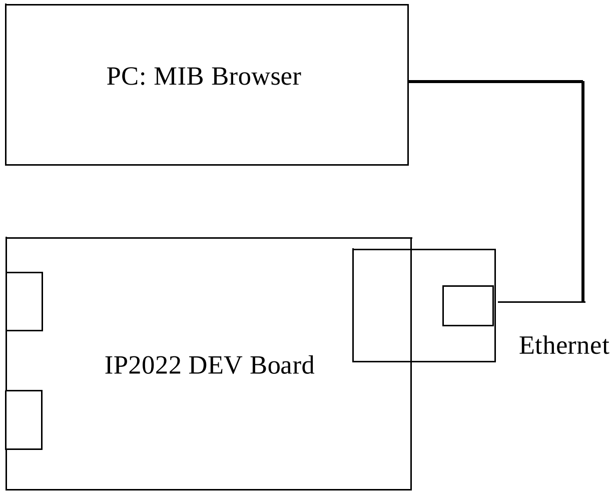


Figure 9-13 SNMP Compiler ipOS Demo Board Diagram

Jumper Configuration: as defined in Section 9.1.4.

Cables:

- CAT 5 Ethernet Crossover
- Power
- Parallel ISD/ISP

9.6.2 PC Configuration

To view the SNMP agent running on the IP2022 Demo Board, you need to have an SNMP application running on the PC. A variety of SNMP applications are available; none is included with this project. Here, for illustration purposes, we use an SNMP MIB Browser from Koshna Software Technologies (www.koshna.com).

9.6.3 Demo Project Configuration and Build

- Launch Unity.
- Create a new project. Project->new
- Select the snmp_compiler_ipos template
- Compile the project. Build->Compile
- Download the project. Build->Debug
- Run the project. Click the "Continue" button.

9.6.4 Using the Demo

With the MIB browser, load the MIB file **`RFC1155-SMI`**, which is located at **`\Ubicom\sdk\tools\mibs\ietf\.`** Then load the project MIB file **`IPOS-MIB`** which is located in the directory **`sdk_demo\snmp_compiler_ipos\app\`**. Enter the configured IP address of the Demo Board in the address box of the toolbar. Now you can perform SNMP operations on ipOS objects which are controlled by the SNMP agent running on the Demo Board. Figure 9-14 shows the result of a GET operation on the iposFreeHeap object.

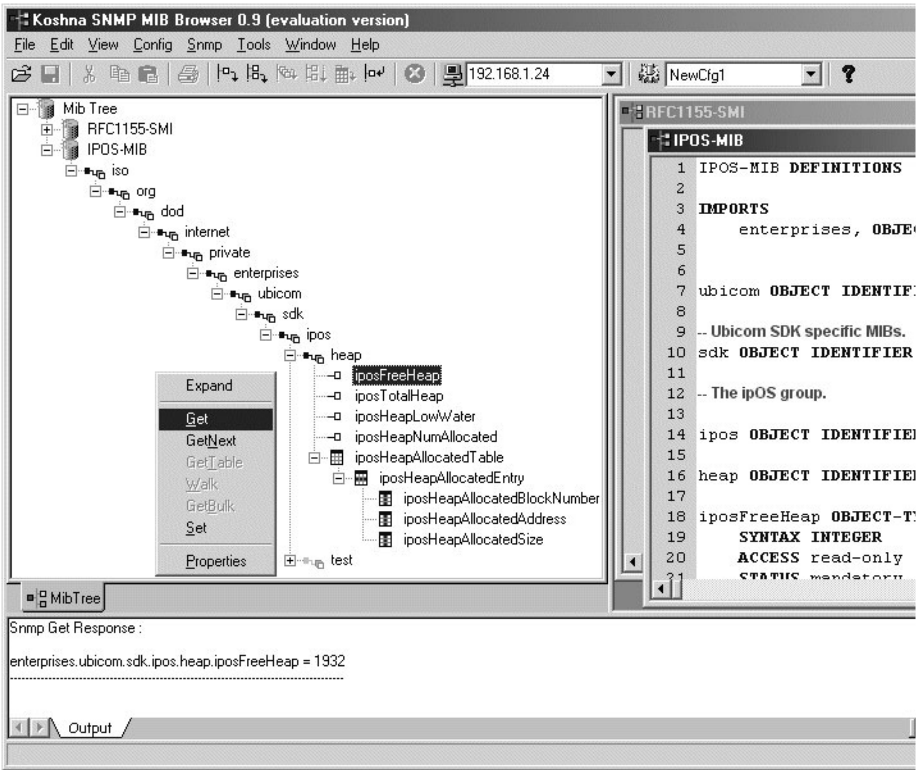


Figure 9-14 Browsing the SNMP Agent

9.7 USB RNDIS Device Demo

The ipUSB package of the SDK provides a USB Serial Interface Engine (SIE) implementation and a complete USB Remote NDIS stack, including an Application Programming Interfaces (API) and services. This project uses ipUSB to implement a USB to Ethernet bridge application -- the Demo Board communicates with a Windows PC through Window’s USB RNDIS driver on one port and with an Ethernet LAN on the other port. The project demonstrates how easily a device manufacturer can incorporate USB Remote NDIS support in a peripheral device by employing the IP2022.

This demo requires the use of an IP2022 Development Board and Native 10bT Ethernet Daughter Card (both provided in the Connectivity Kit), and a USB Device Daughter Card (available separately).

9.7.1 IP2022 Demo Board Configuration

Plug the Native 10bT Ethernet Daughter Card into J2 of the an IP2022 Development Board (SERDES channel 1). Connect the Ethernet Daughter Card to your LAN with astraight-through CAT5 Ethernet cable.

Plug the USB Device Daughter Card into J3 of the an IP2022 Development Board (SERDES channel 2). One standard USB cable is required to connect the USB Daughter Card to the PC.

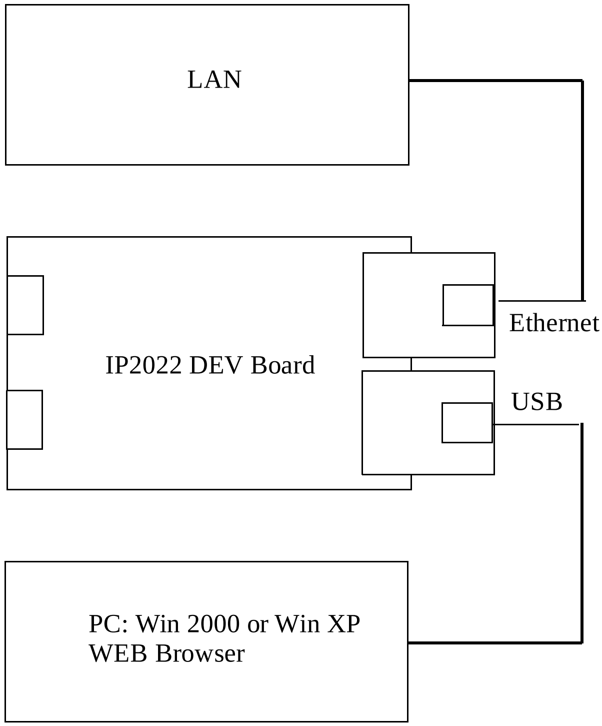


Figure 9-15 USB RNDIS Device Demo Board Diagram

Demo Board Jumper Configuration: as defined in Section 9.1.4.

USB Daughter Board Configuration:

- Install jumper J3
- Position SW1 to OFF-OFF-ON (C1 and C2 toward USB Connector, C3 toward #3)

9.7.2 PC Configuration

The application will get its IP from DHCP by default; however, if no DHCP server is available, select Start -> Control Panel -> Network and assign a static IP.

9.7.3 Demo Project Configuration and Build

- Launch Unity.
- Create a new project. Project->new
- Select the usb_device_rndis template
- Compile the project. Build->Compile
- Download the project. Build->Debug
- Run the project. Click the "Continue" button.

9.7.4 Using the Demo

Connect the USB cable between the PC and the USB Device Daughter Card. Windows will ask for a driver for the "Ubicom USB Remote NDIS Network Device"; this driver is located in the Driver directory of this SDK project. The driver is designed to work with Windows 2000 or Windows XP.

You should now be connected to the network; simply use the network as normal. You can use Windows' Network Neighborhood and you can browse the web.

Several cautions for a healthy demo:

1. Windows 2000/XP is preferred.

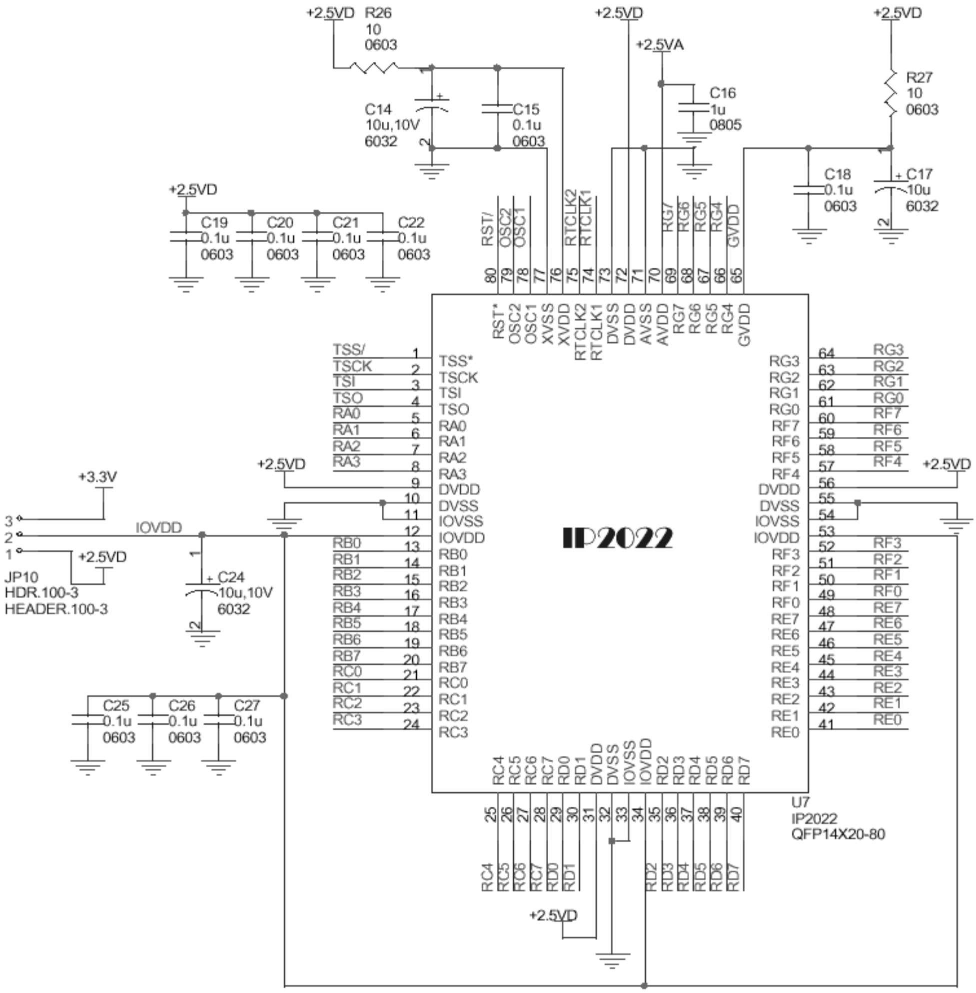
2. Disable your current network adapter. You don't need to, but it will be preferred over the USB port.
3. If page download gets stuck, refresh.
4. If the USB device has locked up, no more traffic or no status reads, unplug it, then plug it in again.
5. If the PC doesn't recognize the USB device when it's plugged in, follow this sequence in the order shown:
   – Unplug the USB cable.
   – IMPORTANT! wait for the "unsafe removal of device" dialog box to come up. It will LOCK UP if you do not wait.
   – Reset the IP2022 board. DO NOT reset the IP2022 board until after the USB cable is removed.
   – Plug the USB cable back in.
6. Unplug and replug to demonstrate PnP, this also resets and clears potential accumulations.
7. Occasionally, the Microsoft Windows driver may lock up.

A.0

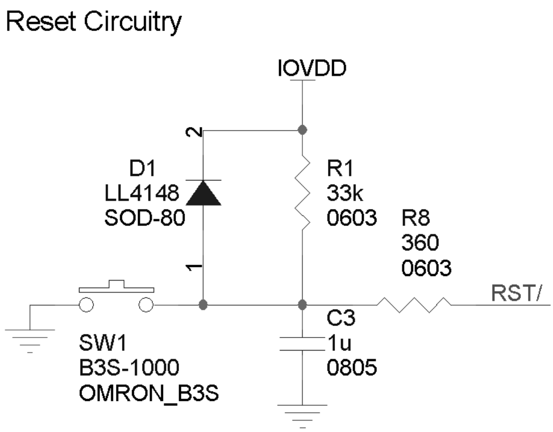
Demo Board Schematics

This appendix shows the schematic diagrams for the Demo Board. Resistors R37 and R39 and capacitors C28 through C31 appear in the crystal oscillator circuits, however these components are not required, and their locations on the Demo Board are not populated.

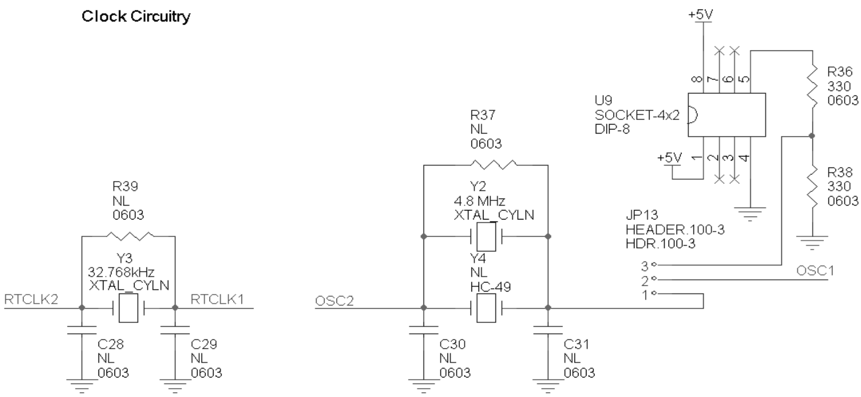
A.1 IP2022


+2.5VD
R26
10
0603
C14
10u,10V
6032
C15
0.1u
0603
+2.5VD
+2.5VA
C16
1u
0805
+2.5VD
R27
10
0603
C18
0.1u
0603
C17
10u
6032
+2.5VD
C19
0.1u
0603
C20
0.1u
0603
C21
0.1u
0603
C22
0.1u
0603
RST/
OSC2
OSC1
RTCLK2
RTCLK1
RG7
RG6
RG5
RG4
GVDD
RST*
XVSS
XVDD
DVSS
DVDD
AVSS
AVDD
TSS/
TSCK
TSI
TSO
RA0
RA1
RA2
RA3
TSS*
IOVSS
IOVDD
RB0
RB1
RB2
RB3
RB4
RB5
RB6
RB7
RC0
RC1
RC2
RC3
RG3
RG2
RG1
RG0
RF7
RF6
RF5
RF4
RF3
RF2
RF1
RF0
RE7
RE6
RE5
RE4
RE3
RE2
RE1
RE0
IP2022
+3.3V
IOVDD
+2.5VD
JP10
HDR.100-3
HEADER.100-3
C24
10u,10V
6032
C25
0.1u
0603
C26
0.1u
0603
C27
0.1u
0603
RC4
RC5
RC6
RC7
RD0
RD1
RD2
RD3
RD4
RD5
RD6
RD7
U7
IP2022
QFP14X20-80
+2.5VD

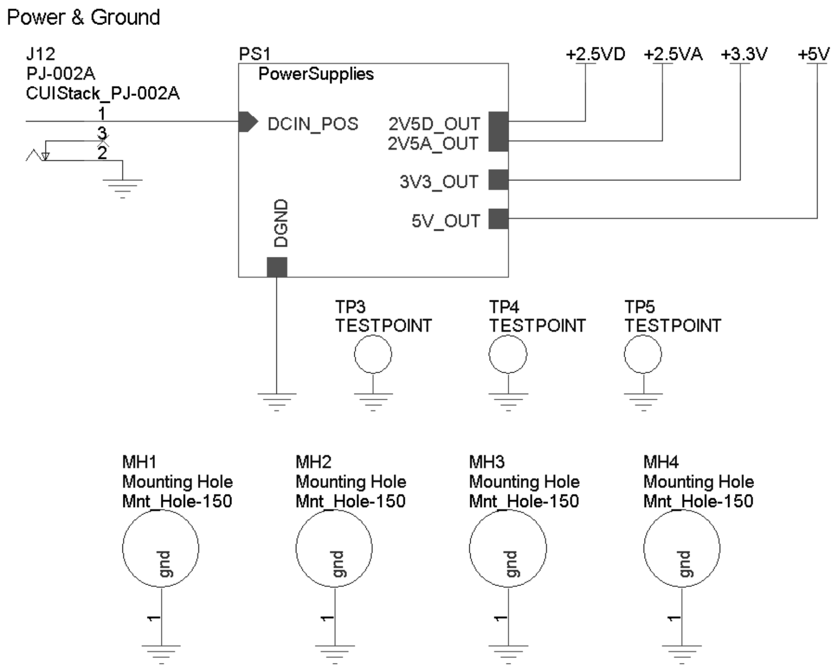
A.2 Reset



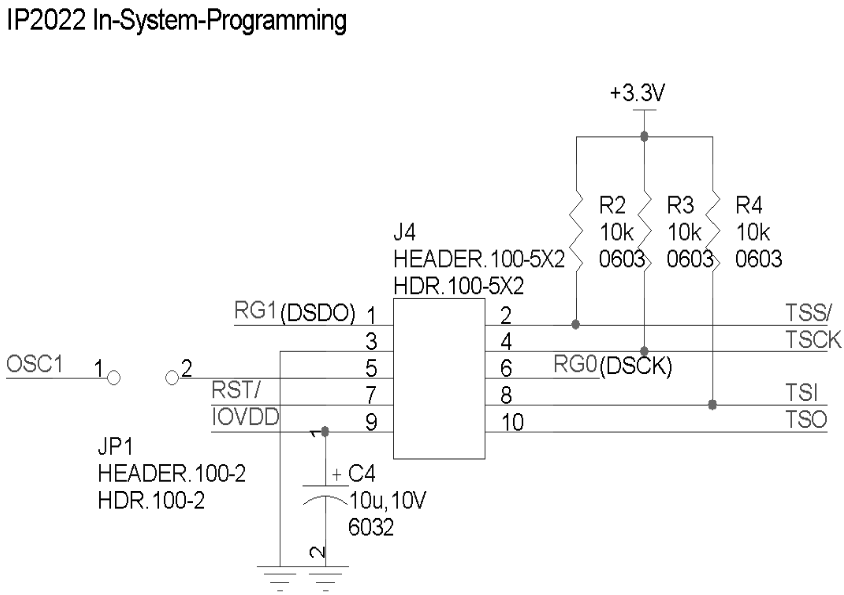
A.3 Clock



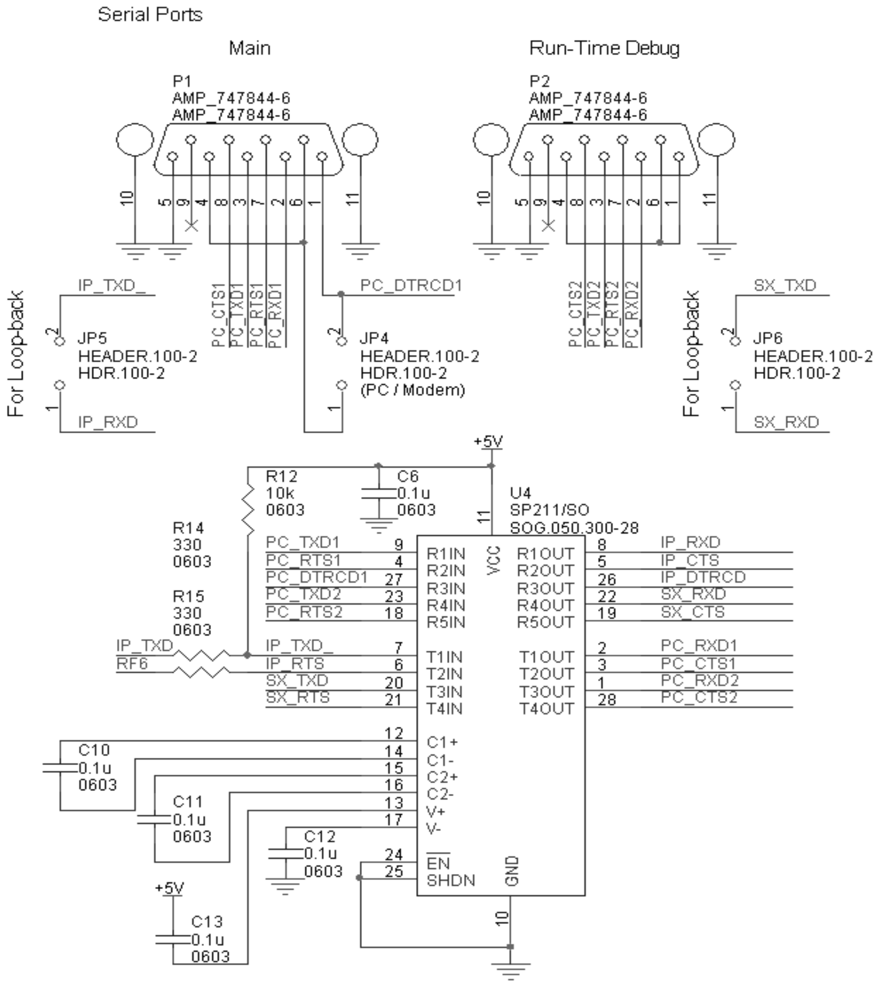
A.4 Power


Power & Ground
J12
PJ-002A
CUIStack_PJ-002A
1
3
2
PS1
PowerSupplies
DCIN_POS
2V5D_OUT
2V5A_OUT
3V3_OUT
5V_OUT
DGND
+2.5VD
+2.5VA
+3.3V
+5V
TP3
TESTPOINT
TP4
TESTPOINT
TP5
TESTPOINT
MH1
Mounting Hole
Mnt_Hole-150
MH2
Mounting Hole
Mnt_Hole-150
MH3
Mounting Hole
Mnt_Hole-150
MH4
Mounting Hole
Mnt_Hole-150
gnd
1

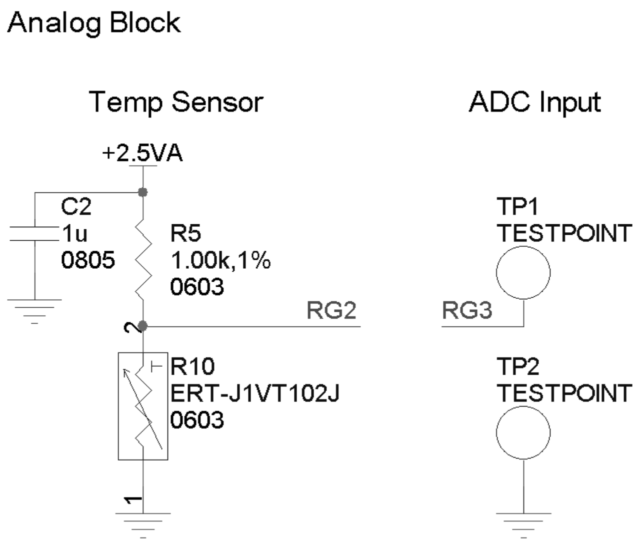
A.5 ISD/ISP


IP2022 In-System-Programming
+3.3V
R2
10k
0603
R3
10k
0603
R4
10k
0603
J4
HEADER.100-5X2
HDR.100-5X2
RG1(DSDO)
1
2
TSS/
3
4
TSCK
OSC1
1
2
5
6
RG0(DSCK)
RST/
7
8
TSI
IOVDD
9
10
TSO
JP1
HEADER.100-2
HDR.100-2
C4
10u,10V
6032

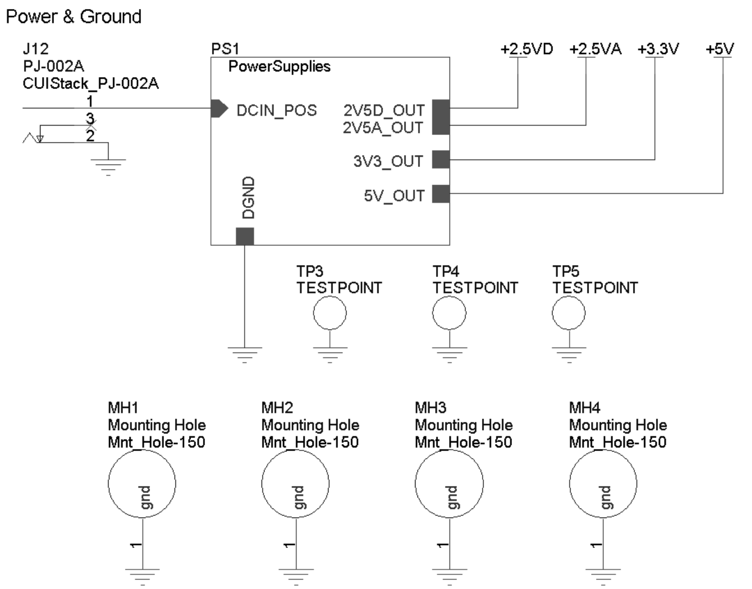
A.6 RS-232 Communications



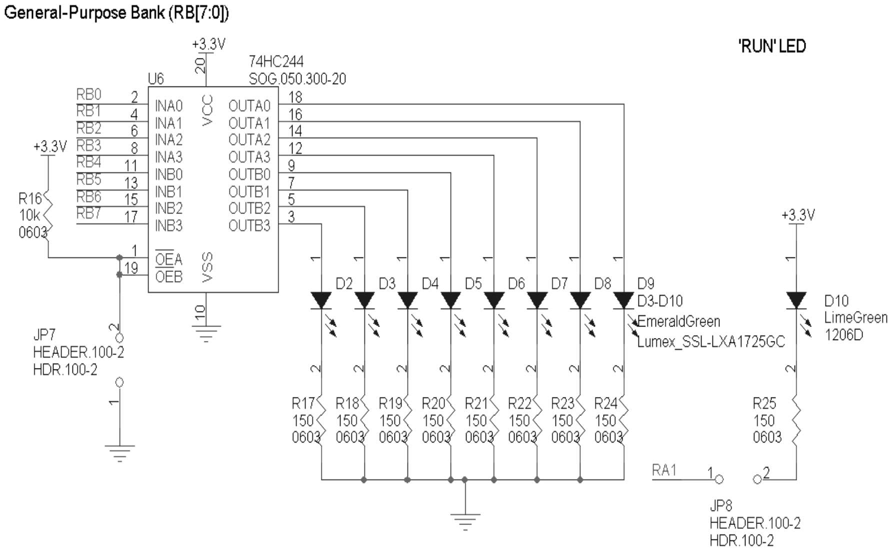
A.7 Analog Unit



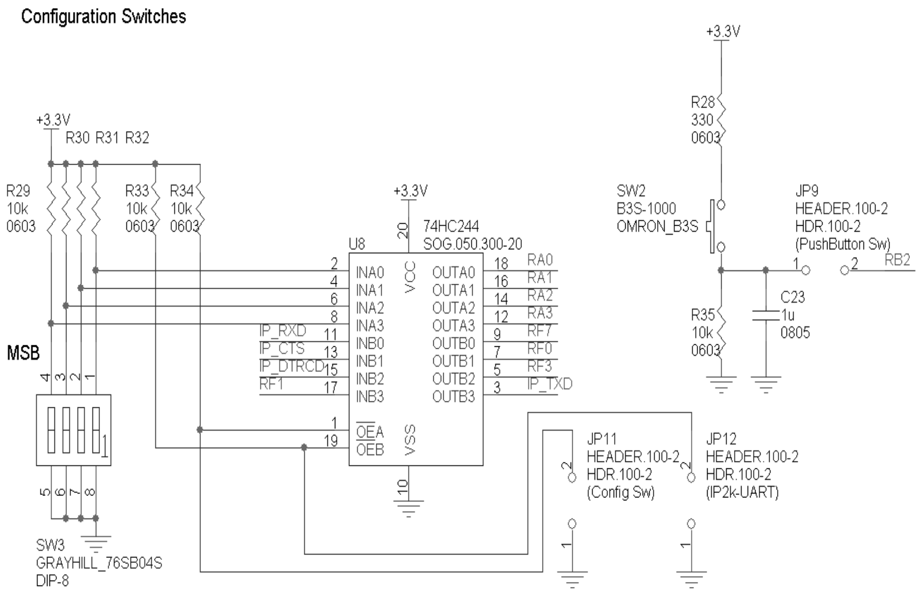
A.8 On-Board Power Supply



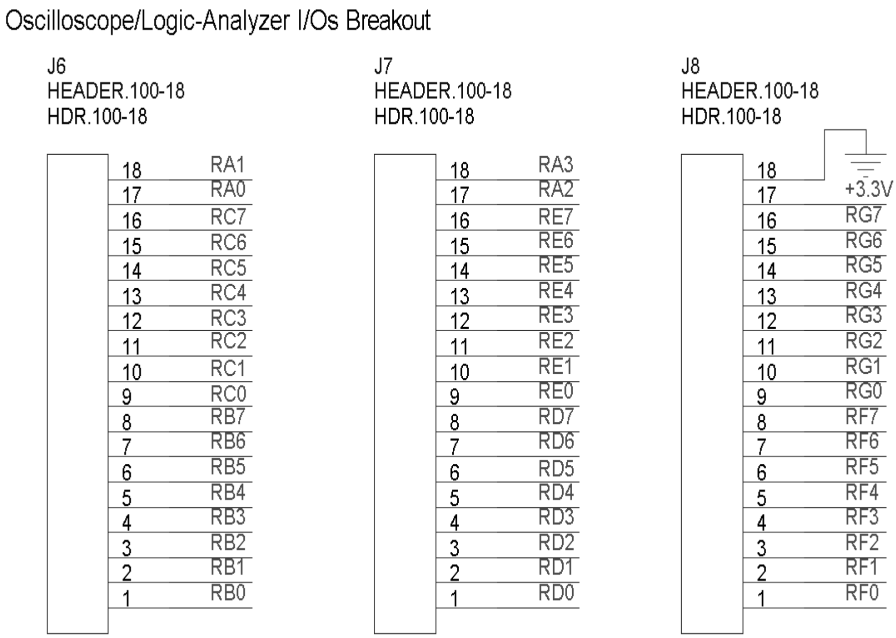
A.9 User LEDs


General-Purpose Bank (RB[7:0])
'RUN' LED
+3.3V
U6
74HC244
SOG.050.300-20
RB0
RB1
RB2
RB3
RB4
RB5
RB6
RB7
INA0
INA1
INA2
INA3
INB0
INB1
INB2
INB3
OEA
OEB
VCC
VSS
OUTA0
OUTA1
OUTA2
OUTA3
OUTB0
OUTB1
OUTB2
OUTB3
R16
10k
0603
JP7
HEADER.100-2
HDR.100-2
D2
D3
D4
D5
D6
D7
D8
D9
D3-D10
EmeraldGreen
Lumex_SSL-LXA1725GC
D10
LimeGreen
1206D
R17
R18
R19
R20
R21
R22
R23
R24
150
0603
R25
RA1
JP8
HEADER.100-2
HDR.100-2

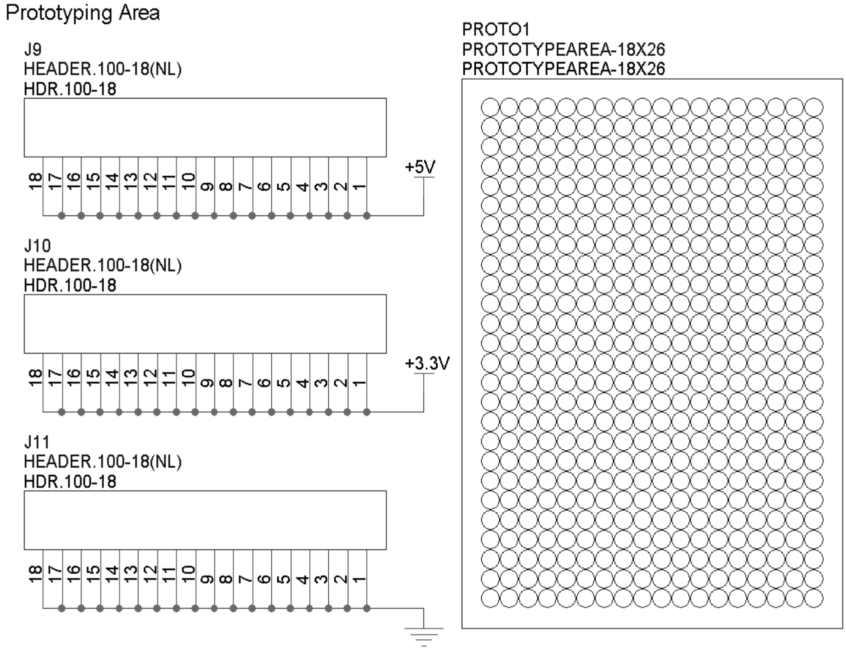
A.10 Switches



A.11 Osciloscope I/O Breakout

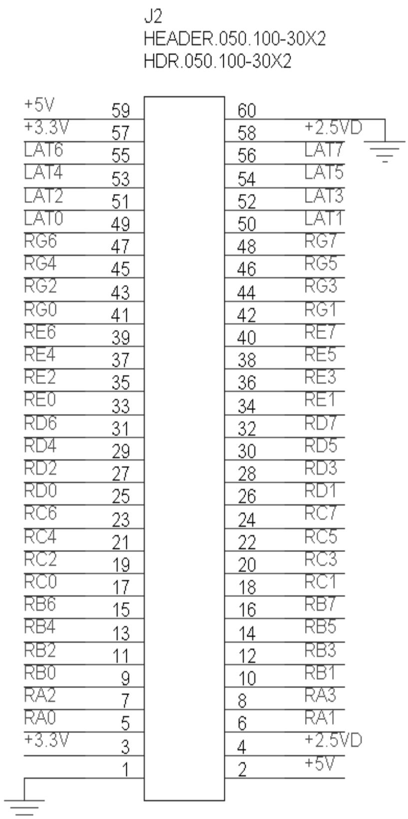


A.12 Prototype Area

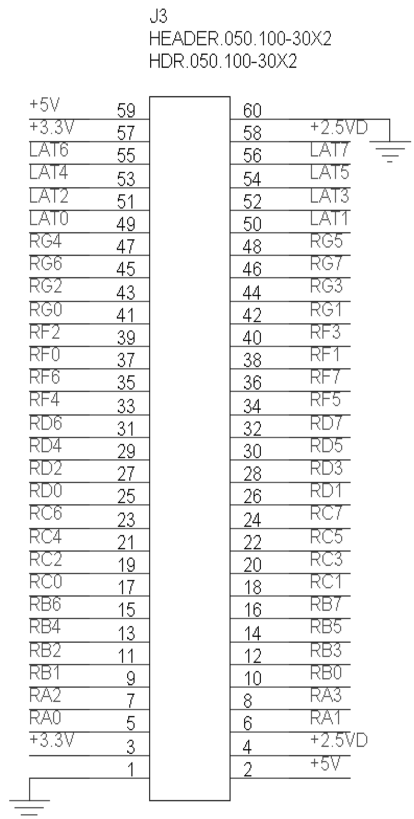


A.13 Daughter Board Connectors (J2, J3)

Daughterboard Interface Connectors



J2 provides access to Serdes 1



J3 provides access to Serdes 2

B.0 Other IP2022 Resources

- Ubicom website http://www.ubicom.com/ and Technical Support Portal https://portal.ubicom.com/
- SXlist Web site: http://www.sxlist.com/
- SX-Forum Web site: http://www.sx-forum.com/
- Frequently Asked Questions is part of the SDK Manual which can be accessed from Unity Main menu Help->SDK_Help-> ip200SDK Manual->Frequently Asked Questioms