



# **User's Primer**

**IP2022 Internet Processor™**

## Revision History

Revision	Release Date	Summary of Changes
1.0	March 15, 2001	Original issue.
1.1	April 5, 2001	Merged with demo board User's Guide.
1.2	May 20, 2001	Rewritten for the Unity IDE.

© 2001 Ubicom, Inc. All rights reserved. No warranty is provided and no liability is assumed by Ubicom with respect to the accuracy of this documentation or the merchantability or fitness of the product for a particular application. No license of any kind is conveyed by Ubicom with respect to its intellectual property or that of others. All information in this document is subject to change without notice.

Ubicom products are not authorized for use in life support systems or under conditions where failure of the product would endanger the life or safety of the user, except when prior written approval is obtained from Ubicom.

Ubicom™ and the Ubicom logo are trademarks of Ubicom, Inc.

Internet Processor™ is a trademark of Ubicom, Inc.

All other trademarks mentioned in this document are property of their respective companies.



**Ubicom, Inc.**  
**1330 Charleston Road**  
**Mountain View, CA 94043**  
tel 650 210 1500  
fax 650 210 8715  
[www.ubicom.com](http://www.ubicom.com)

# Table of Contents

<b>1</b>	<b>Quick Set Up</b>	<b>1</b>
<b>2</b>	<b>Overview</b>	<b>3</b>
2.1	Minimum System Requirements	9
2.2	Installation CD-ROM	10
2.3	Installing the Software	11
2.4	Contents of the <code>ubicom</code> Directory	12
<b>3</b>	<b>Evaluation Kit and Demo Board</b>	<b>13</b>
3.1	Terminology	14
3.2	Demo Board Set-Up	15
3.2.1	Demo Board Connections	15
3.2.2	Power Supply	15
3.2.3	PC/Terminal Connection	16
3.2.4	ISD/ISP Connection	16
3.2.5	Verifying Installation	17
3.3	Description of Hardware Blocks	17
3.3.1	IP2022	17
3.3.2	Reset	17
3.3.3	Clock	18
3.3.4	RTCLK	18
3.3.5	Power	18
3.3.6	ISD/ISP	18
3.3.7	ADC and ADC Reference	19
3.3.8	Run LED	19
3.3.9	I <sup>2</sup> C EEPROM	19
3.3.10	SPI Temperature Sensor	20
3.3.11	On-Board Power Supply	20
3.3.12	User LEDs	20
3.4	Switches	20
3.4.1	Jumpers	21



3.4.2	Connectors	23
3.4.3	Prototype Area	23
3.4.4	Daughter Board Connectors (J3, J4)	23
3.4.5	Test Points	26
3.5	Board Configuration Options	26
3.5.1	RS232 (JP15-JP23)	27
3.5.2	I <sup>2</sup> C (JP1, JP3, JP6)	27
3.5.3	SPI (JP2, JP4, JP5, JP7)	28
3.5.4	IOVDD (JP13)	28
3.5.5	Tool VDD (JP24)	28
3.5.6	Clock (JP14)	28
3.5.7	RTCLK (JP12)	28
3.5.8	4-Bit DIP Switch (JP8-JP11)	29
3.5.9	Run LED (JP26)	29
<b>4</b>	<b>Unity User Interface</b>	<b>31</b>
4.1	Unity Windows	33
4.2	Unity Menu Bar	33
4.3	Unity Tool Bar	34
4.4	Creating a New Project	38
4.5	Adding and Removing Files	39
4.6	Project File Tree	40
<b>5</b>	<b>Configuring ipModules</b>	<b>41</b>
5.1	Project Configuration GUI	42
5.2	Generating Makefiles and Header Files	44
<b>6</b>	<b>Compiling</b>	<b>45</b>
<b>7</b>	<b>Debugging</b>	<b>49</b>
7.1	Debugging Tool Bar	49
7.2	Viewing Program Execution	52
7.3	Breakpointing Program Execution	54
7.4	Viewing Registers	58
7.5	Viewing Memory	59



7.6	Memory Map	62
<b>8</b>	<b>Utilities</b>	<b>63</b>
8.1	IP2KProg User Interface	65
8.1.1	Downloading a Project	67
8.2	GNU Binary Utilities	68
<b>A</b>	<b>Assembly Language Syntax</b>	<b>71</b>
A.1	Comments, Constants, and Symbols	71
A.2	Addressing Modes	73
A.3	Sections Used in Default Linker Script	75
A.4	Summary of CPU Instructions	75
A.5	Summary of CPU and Peripheral Registers	83
A.6	List of IP2022-Specific Reserved Words	90
A.7	Directives	91
A.8	In-Line Assembly in C Source Files	92
<b>B</b>	<b>Demo Board Schematics</b>	<b>93</b>
B.1	IP2022	94
B.2	Reset	95
B.3	Clock	95
B.4	RTCLK	96
B.5	Power	96
B.6	ISD/ISP	97
B.7	RS-232 Communications	98
B.8	ADC and ADC Reference	99
B.9	Run LED	100
B.10	I <sup>2</sup> C EEPROM	100
B.11	SPI Temperature Sensor	101
B.12	On-Board Power Supply	101
B.13	User LEDs	102
B.14	Switches	103
B.15	Jumpers	105
B.16	Connectors	106



## Table of Contents—IP2022 Getting Started

---

B.17	Prototype Area . . . . .	107
B.18	Daughter Board Connectors (J3, J4). . . . .	109
B.19	Test Points . . . . .	109
B.20	Bypass Capacitors . . . . .	110
B.21	Mounting Holes . . . . .	110



## List of Figures

Figure 2-1	Configuration Files .....	4
Figure 2-2	Compilation Tool Chain .....	5
Figure 4-1	Unity (Default View) .....	32
Figure 5-1	Project Configuration GUI .....	42
Figure 6-1	Makefile Structure .....	47
Figure 7-1	Browse Tab .....	55
Figure 7-2	Setting a Breakpoint .....	56
Figure 7-3	Stack Window .....	57
Figure 7-4	Registers Window .....	58
Figure 7-5	Quick Watch Box .....	59
Figure 7-6	Watch Window .....	60
Figure 7-7	Memory Window .....	61
Figure 7-8	Debugging Memory Space .....	62
Figure 8-1	IP2KProg User Interface .....	65



## List of Figures—IP2022 Getting Started

---



## List of Tables

Table 2-1	Summary of Files.....	6
Table 2-2	System Requirements.....	9
Table 2-3	Summary of Directories.....	10
Table 2-4	Summary of ubicom Directories.....	12
Table 3-1	Demo Board Jumpers.....	21
Table 3-2	SERDES1 (J3) Pin Assignments.....	24
Table 3-3	SERDES2 (J4) Pin Assignments.....	25
Table 3-4	Test Points.....	26
Table A-1	Special Characters.....	72
Table A-2	Addressing Modes.....	74
Table A-3	Key to Abbreviations and Symbols.....	75
Table A-4	Logical Instructions.....	77
Table A-5	Arithmetic and Shift Instructions.....	77
Table A-6	Bit Operation Instructions.....	81
Table A-7	Data Movement Instructions.....	81
Table A-8	Program Control Instructions.....	82
Table A-9	System Control Instructions.....	82
Table A-10	Register Addresses and Reset State.....	83



## List of Figures—IP2022 Getting Started

---



## Preface

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

- *IP2022 Data Sheet*, available from Ubicom.
- *IP2022 User's Manual*, available from Ubicom.
- *IP2022 Quick Reference Guide*, available from Ubicom.

Reference manuals for the tool chain:

- *GNUPro Toolkit—GNUPro Utilities*, available from Red Hat.
- *GNUPro Toolkit—GNUPro Compiler Tools*, available from Red Hat.
- *GNUPro Toolkit—Debugging with GDB*, available from Red Hat.

### Notational Convention

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 Start menu 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.





# 1.0

## Quick Set Up

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

1. *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 `IP2022 Development Tools\Install\IP2000 GNUPro Tools Setup.exe` (name may vary).

Run `Software Development Kit_SDK\Install\IP2000 Unity Setup.exe` (name may vary).

Run `IP2022 Development Tools\Install\IP2000 SDK Setup.exe` (name may vary).

2. *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 between the host PC and the Demo Board serial connector labelled To PC. Then, connect the power supply cable to the Demo Board and plug the power supply into an AC outlet.



3. *Create a Project*—create a directory called **C:\Ubicom\_Projects**. Open Unity by selecting Programs -> Ubicom -> Unity from the Windows Start menu. Then, select the Project -> New command from the Unity menu bar. For the project name, enter **C:\Ubicom\_Projects\Project1.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 the Build -> Compile command from the Unity menu bar.
4. *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.
5. *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.

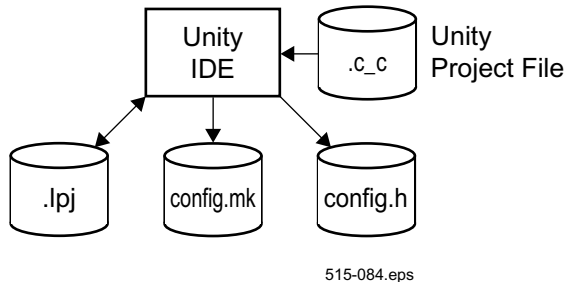
The files which comprise an application are called a *project*. There are five phases in project development with Unity:

- *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 project configuration 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 downloads executable code to the IP2022 device through the ISD/ISP interface.
- *Debugging*—the source-level editor is part of the GUI for a powerful C/assembly language debugger.

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.

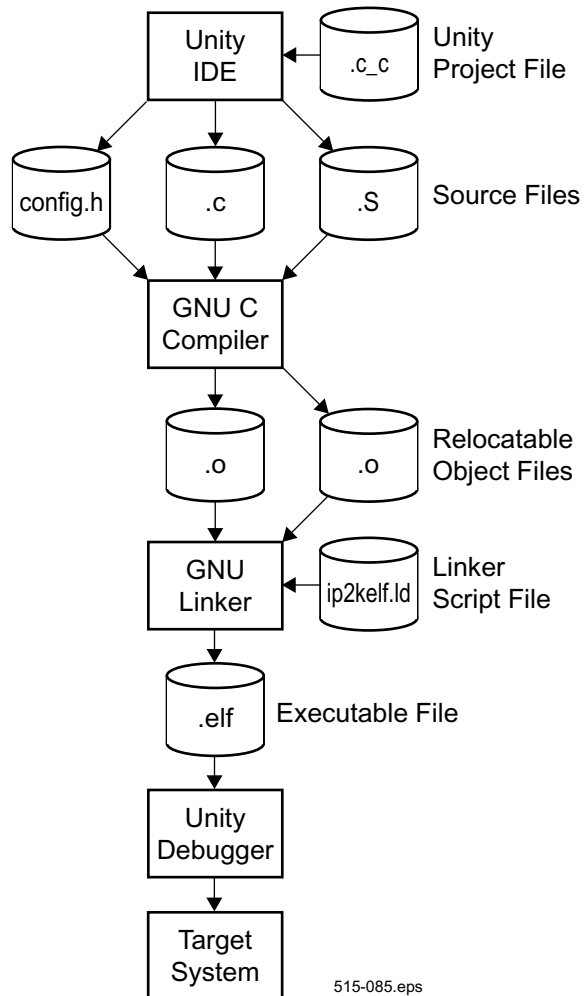


Figure 2-2 Compilation Tool Chain



As mentioned before, the `config.h` header file is generated automatically by Unity from the project configuration. 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 it produces an executable file with the `.elf` extension. Most users will not need to modify the default linker script file `ip2kelf.ld` provided by Uvicom. 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
<code>.c</code>	<i>C source file</i> —source file(s) for the application.
<code>.c_c</code>	<i>Unity project file</i> —used by Unity to store project-specific information (location of source files, etc.).
<code>.elf</code>	<i>Binary file</i> —executable code in the ELF file format.

**Table 2-1 Summary of Files** (continued)

<b>File Type</b>	<b>Description</b>
<code>.lpj</code>	<i>Configuration Tool project file</i> —used by Configuration Tool to store project-specific information (ipModule selection, I/O port pin assignments, etc.).
<code>.o</code>	<i>Object file</i> —compiled object code.
<code>.s</code>	<i>Assembly-language source file</i> —pure assembly language file, as opposed to in-line assembly in a C source file.
<code>config.h</code>	<i>Automatically generated C header file</i> —macro definitions used by the C compiler.
<code>config.mk</code>	<i>Automatically generated makefile</i> —macro definitions used by the <b>make</b> utility.
<code>ip2kelf.ld</code>	<i>Linker script file</i> —most users will not need to modify the default <code>ip2kelf.ld</code> file provided by Uvicom.
<b>Makefile</b>	<i>Project makefile</i> —the top-level makefile in the project directory called by Unity.
<b>Makefile.inc</b>	<i>Source file makefiles</i> —all source file directories have an associated <b>Makefile.inc</b> with rules for compiling the files they contain.



A typical development sequence is:

1. *Copy a Project Template*—start a new project by copying a template, to make sure all of the default files are present. 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. *Describe the Project Configuration*—a GUI-based editor is used to select the ipModules used in a project and customize their features. Unity keeps information about the project configuration in a `.lpj` file. Unity 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 in Table 2-2.

**Table 2-2 System Requirements**

<b>Feature</b>	<b>Requirements</b>
CPU	Pentium 2 or equivalent 300 MHz or greater
Memory	64 Mbytes
Free Disk Space	300 Mbytes
Operating System	Windows 98/Windows NT/Windows 2000
Peripherals	<i>Required</i> —parallel port <i>Recommended</i> —serial port, Ethernet interface



## 2.2 Installation CD-ROM

The installation CD-ROM has four directories, described in Table 2-2.

**Table 2-3 Summary of Directories**

Name	Description
IP2022 Development Tools	GNUPro tools directory. The Install subdirectory contains the installation programs for the GNUPro tools and the Unity IDE.
IP2022 Demo Board	Demo Board documentation, including schematics, layout, and bill of materials.
IP2022 Silicon Documentation	IP2022 Data Sheet and User's Manual.
Software Development Kit	The Install subdirectory contains the installation program for the ipModule software and template files.



## 2.3 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 GNUPro Tools*—run `IP2022 Development Tools\Install\IP2000 GNUPro Tools Setup.exe` (name may vary).
3. *Install SDK*—run `Software Development Kit_SDK\Install\IP2000 SDK Setup.exe` (name may vary). Allow all of the default selections to be used.
4. *Install Unity*—run `IP2022 Development Tools\Install\IP2000 Unity Setup.exe` (name may vary). Allow all of the default selections to be used. Unity will modify the `C:\autoexec.bat` file to add an `IP2K` path variable. The default path is:  
`C:\ubicom\ip2ktools\H-i686-pc-cygwin32\bin`
5. *Restart the computer*—this is necessary for the changes made in `Autoexec.bat` to take effect.



## 2.4 Contents of the `ubicom` Directory

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

**Table 2-4 Summary of `ubicom` Directories**

Name	Description
Help	Documentation files used by on-line help.
ip2ktools	Executable files for tools and utilities.
sdk	Project templates and ipModule software.
Unity	Unity executable file and assembly language project template.

# 3.0

## Evaluation Kit and Demo Board

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 evaluation kit contains:

- IP2022 Demo Board (silk screen labelled as "IP2022 DEMO BOARD V1.0")
- AC Adapter (12 VDC output)
- ISD/ISP Cable (between PC parallel port and Demo Board)
- DB9 Serial Cable
- CD-ROM with Software Tools, Documentation, and **starter Example**

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

- IP2022 Internet Processor
- 12 VDC Power Input
- RS-232 Serial Connections (for DTE to DTE with optional hardware flow control and DTE to DCE/modem)
- Daughter Board Connectors for Ethernet and USB Interfaces



- Switching Power Supply with Power-On LED
- Run LED (controlled from software)
- 4 MHz Crystal
- 32.768 kHz Crystal for RTCLK
- Reset Button
- External EEPROM
- Digital Temperature Sensor with SPI Interface
- 4 Pushbuttons
- 4 DIP Switches
- 8-LED Bank
- ADC Input
- 2.5V ADC External Reference Voltage
- ISP/ISD 10-pin Header
- Prototyping Area
- Multiple Test Points

### 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 Uicom 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

To use this interface, jumpers are required on DCE\_TXD and DCE\_RXD.

### 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, which is indicated by a red stripe, does not have a header pin. 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 100 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 V<sub>dd</sub> 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 MHz crystal is connected between pins OSC1 and OSC2 of the IP2022. The 4 MHz crystal can be disconnected from OSC1 by removing jumper JP14.

### 3.3.4 RTCLK

A 32.768 kHz crystal is connected between pins RTCLK1 and RTCLK2 of the IP2022. The 32.768 kHz crystal can be disconnected from RTCLK1 by removing jumper JP12.

### 3.3.5 Power

The IP2022 core logic is powered from the +2.5D digital power rail. The IP2022 pins are driven from the IOVDD power rail, which is selectable between 3.3V and 2.5V. The on-chip ADC is connected to the +2.5A analog power rail. 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. Pullup 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 ADC and ADC Reference**

The ADC IN and AGND test points may be used to apply a voltage (0–2.5V) to the ADC input. On the Demo Board, the ADC input is ADC channel 7 (RG7). 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. Jumper JP25 connects a 2.5V precision reference voltage to the RG3 pin.

### **3.3.8 Run LED**

Software can indicate code execution on the Run LED when jumper JP26 is inserted.

### **3.3.9 I<sup>2</sup>C EEPROM**

An external EEPROM provides 32K bytes of memory. The IP2022 communicates with the EEPROM through an I<sup>2</sup>C ipModule.



### 3.3.10 SPI Temperature Sensor

An external temperature sensor provides temperature readings to the IP2022 through SerDes 1 operating in SPI interface mode or through an SPI ipModule.

### 3.3.11 On-Board Power Supply

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

- +5V at 500 mA
- +3.3V at 500 mA
- +2.5V at 500 mA
- Unregulated >12V

The POWER LED indicates when power is on.

### 3.3.12 User LEDs

The LED bank is connected to Port B through an 8-bit DIP switch (SW1). The LEDs can be disconnected from Port B by opening the switches, e.g. for other uses of Port B.

## 3.4 Switches

Switch SW1 connects the LED bank to Port B.

Switches SW2, SW3, SW5, and SW6 are pushbutton switches which can be accessed at header J7. These switches are not



initially connected to the IP2022, but are available for any purpose.

Switch SW4 is a 4-bit DIP switch connected to Port A.

Switch SW7 is the IP2022 reset switch.

### 3.4.1 Jumpers

The jumpers are described in Table 3-1. When a jumper is shown as unconnected as shipped, it means the Demo Board is shipped with the jumper over one pin of the header. This is used as a parking place to hold the jumper.

**Table 3-1 Demo Board Jumpers**

Block	Description	Jumper	IP2022 Signal	Connected As Shipped?
RS232	DCE_TXD	JP21	RF1	Yes
	DCE_RXD	JP15	RF7	Yes
	DCE_RTS	JP20	RC1	No
	DCE_CTS	JP16	RC2	No
RS232	DTE_TXD	JP22	RE5	No
	DTE_RXD	JP17	RE3	No
	DTE_RTS	JP23	RC0	No
	DTE_CTS	JP18	RC3	No
	DTE_DCD	JP19	RC4	No



**Table 3-1 Demo Board Jumpers** (continued)

Block	Description	Jumper	IP2022 Signal	Connected As Shipped?
I <sup>2</sup> C	I2C-SDA	JP1	RE5	No
	I2C-SDA	JP3	RE3	No
	I2C-SCL	JP6	RE0	No
SPI	SPI-SCK	JP4	RE0	No
	SPI-SDO	JP2	RE3	No
	SPI-CS/	JP5	RC5	No
	SPI-SDI	JP7	RE5	No
IOVDD	2.5V or 3.3V	JP13	IOVDD	3.3V
Tool VDD	2.5V or 3.3V	JP24	None	No
Clock	OSC	JP14	OSC1	Yes
RTCLK	RTCLK	JP12	RTCLK1	No
4-Bit DIP	CFG0	JP8	RA0	No
	CFG1	JP9	RA1	No
	CFG2	JP10	RA2	No
	CFG3	JP11	RA3	No
ADC	Ext. 2.5V Vref	JP25	RG3	Yes
RUN LED	RUN	JP26	RG6	No



### 3.4.2 Connectors

There are five connectors on the Demo Board:

- DC power
- Serial RS232 (DB9)
- SERDES1 (60-pin)
- SERDES2 (60-pin)
- ISD/ISP (10-pin)

Another DB9 connector (female) can be added to the board for modem communication.

### 3.4.3 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 DGND, 3.3V, and 5V rails. The prototype area is 26 by 18 plated-through holes.

### 3.4.4 Daughter Board Connectors (J3, J4)

Two 60-pin interface connectors are provided on the IP2022 Demo Board for the purpose of interfacing to Daughter Boards. SerDes 1 is connected to J3 and SerDes 2 is connected to J4. Caution must be observed with SW1 (LED bank switch), because Port B is connected to J3 and J4.



**Table 3-2 SERDES1 (J3) Pin Assignments**

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

**Table 3-3 SERDES2 (J4) Pin Assignments**

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



### 3.4.5 Test Points

Several test points are available on the IP2022 Demo Board, as listed in Table 3-4.

**Table 3-4 Test Points**

Test Point	Description
UNREG	12V, 800 mA DC input from wall-mount adapter
+5V	5V output from switching power supply
+3.3V	+3.3V output from switching power supply
+2.5D	+2.5V output referenced to digital ground
+2.5A	+2.5V output referenced to analog ground
Vref ADC	+2.5V reference voltage for ADC
DGND	Digital ground
AGND	Analog ground
OSC	4 MHz clock signal
RTCLK	32.768 kHz clock signal

### 3.5 Board Configuration Options

The jumpers are used to connected common hardware blocks to the IP2022. The SerDes 1 and 2 can be connected to a variety of devices. Caution should be observed such that multiple devices are not connected to the same SerDes unit.



### 3.5.1 RS232 (JP15-JP23)

The DCE\_TXD and DCE\_RXD jumpers are connected to SerDes 2 of the IP2022. The DTE\_TXD and DTE\_RXD jumpers are connected to SerDes 1 of the IP2022. RTS and CTS are connected to Port C for flow control.

For PC communications:

1. Use the PC DB9 connector.
2. Install the jumpers for DCE\_TXD and DCE\_RXD.
3. If hardware flow control is required, install the jumpers for DCE\_RTS and DCE\_CTS.

For Modem communications:

1. Use Modem DB9 connector.
2. Install the jumpers for DTE\_TXD and DTE\_RXD.
3. Install the jumpers for DTE\_RTS, DTE\_CTS, and DTE\_DCD, according to the requirements for the modem that is connected.

### 3.5.2 I<sup>2</sup>C (JP1, JP3, JP6)

All jumpers must be in place for I<sup>2</sup>C connection. These jumpers connect the EEPROM (slave) to the SerDes 1 of the IP2022 (master).



### 3.5.3 SPI (JP2, JP4, JP5, JP7)

All SPI jumpers must be in place for SPI interface. These jumpers connect the temperature sensor (slave) to the SerDes 1 of the IP2022 (master).

### 3.5.4 IOVDD (JP13)

IOVDD can be selected to be 2.5V or 3.3V.

### 3.5.5 Tool VDD (JP24)

These jumpers are not used with the supplied ISP/ISD cable. Debuggers and programmers have the option of using the board's 2.5V or 3.3V supplies, however be sure the Demo Board power and the debugger or programmer power are compatible before installing this jumper.

### 3.5.6 Clock (JP14)

Jumper connects the 4 MHz crystal to OSC1.

### 3.5.7 RTCLK (JP12)

Jumper connects the 32.768 kHz crystal to RTCLK1.



### 3.5.8 4-Bit DIP Switch (JP8-JP11)

The jumpers labeled CFG0 to CFG3 connect the CONFIG switches to Port A in the following way:

CFG0 = RA0  
CFG1 = RA1  
CFG2 = RA2  
CFG3 = RA3.

### 3.5.9 Run LED (JP26)

The jumper labeled RUN directly below the RUN LED connects the LED to pin 6 of the Port G.





# 4.0

## Unity User Interface

Unity implements a powerful graphical user interface (GUI) that trims the rough edges off 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 upper pane of the Unity GUI, and windows related to debugging occupy the lower 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.

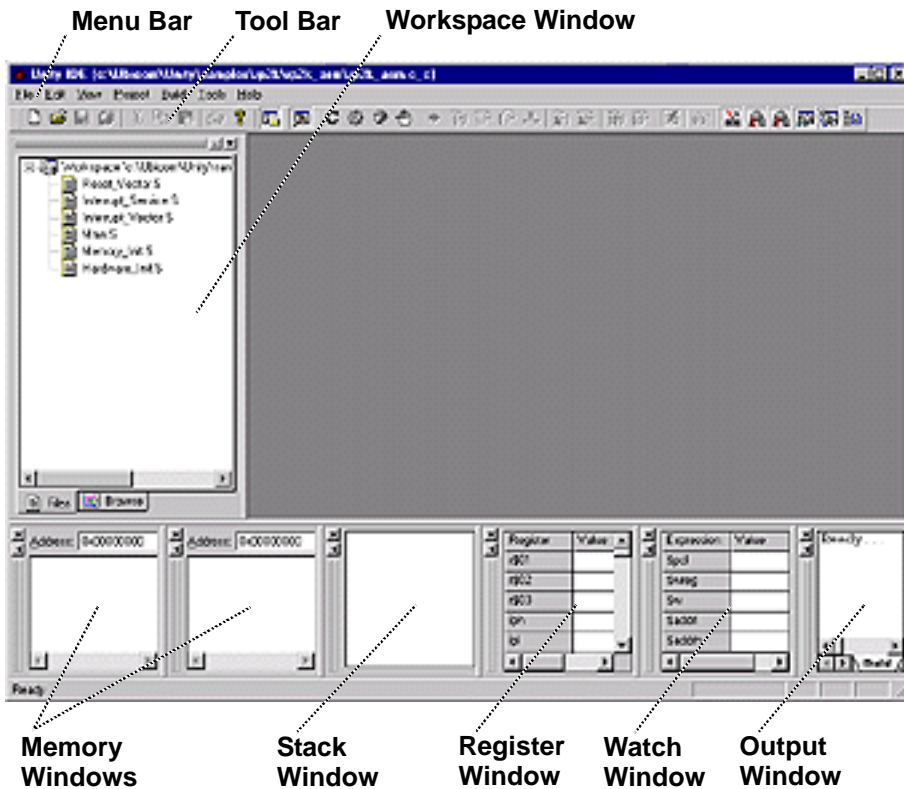


Figure 4-1 Unity (Default View)

## 4.1 Unity Windows

The default view has up to seven open windows:

- *Workspace Window*—lists the source files within the current project. Clicking on a file name opens the file in a new window.
- *Memory Windows*—display regions of the program flash, program RAM, and data memory space.
- *Stack Window*—shows the nesting of subroutines on the hardware call/return stack.
- *Register Window*—displays the contents of the registers. The contents of the registers can be edited in this window.
- *Watch Window*—shows the current value of expressions.
- *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. The default view can be restored by exiting Unity and selecting the `ubicom -> Unity -> UnityReset` command.

## 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 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:



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



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



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





- *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:



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



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



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



- *Print Selection*—print selected source file. Equivalent to the Edit -> Copy 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:



- *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.



- *Debug Project*—Start 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.



- *Start Programmer*—launch the *IP2KProg* utility. Equivalent to the Build -> Start Programmer command or the F4 function key.



- *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. 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 hardware call/return 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 and use one of the Step icons.



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*—select the `ubicom -> Unity` command. This opens the Unity IDE. The first time Unity is opened after installation, an example assembly language project will open, but it is not used in this chapter.
2. *Enter Project File Full Path Name*—click on New Project. A dialog box will appear for entering the path name for the project file. Click on **•••** to browse for the project file. Any legal full path name (with no embedded spaces) may be used. In this example, the project file name is `C:\starter\starter.c_c`. Leave the file 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. Leave the Add SDK Files Into Project box unchecked. (Checking the box causes the included ipModule software to appear in the list of project files.) 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:\starter\starter.c\_c** has been created, and several files and directories have been copied to **C:\starter**.

## 4.5 Adding and Removing Files

To add a file to a project, select the Project -> Add file(s) command or hit the Insert key, then browse for the file. Unity will automatically add references to the new file in the **app/Makefile.inc** file.

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 hit 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:\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

**Makefile.inc**—source directory makefile

**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

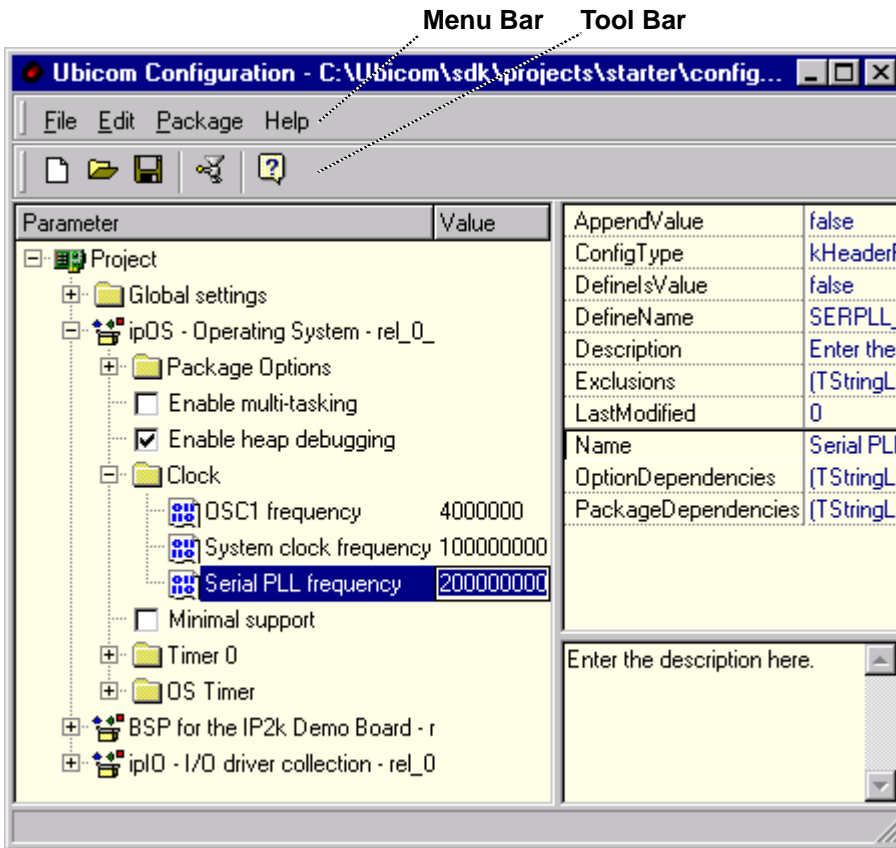
## Configuring ipModules

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”.



# Compiling 6.0

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.

## C

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

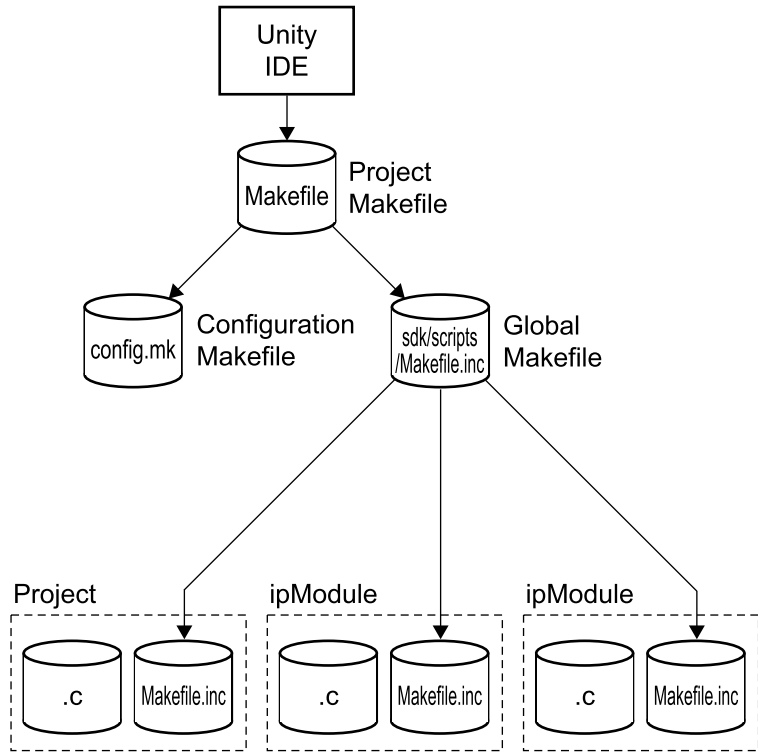


object files. If the modified source file is a header file included in the other four source files, however, it will be necessary to recompile all of the source files. The **make** utility keeps track of file dependencies and invokes the compiler and linker as needed to generate the files requested by 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.





515-086.eps

**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 hit 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 hit 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.



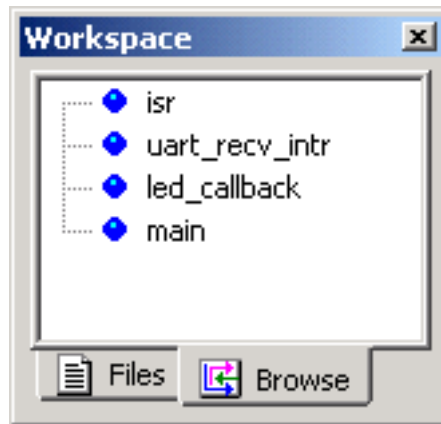
### 7.3 Breakpointing Program Execution

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 insure 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 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. Only one breakpoint can be set in program flash memory. Any number of breakpoints can be set in program RAM. Select the View -> Breakpoint List command to examine the breakpoints which have been set.



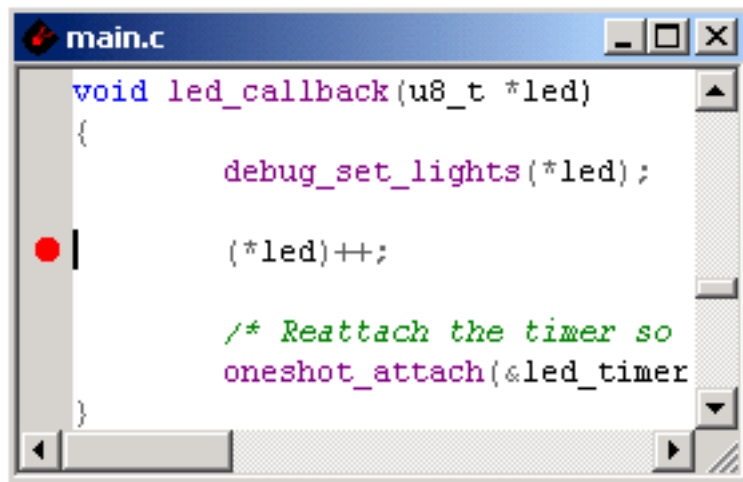
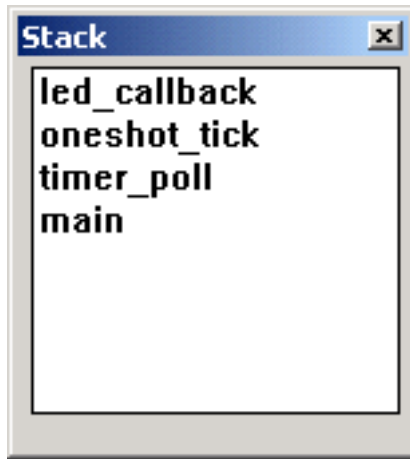


Figure 7-2 Setting a Breakpoint

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**

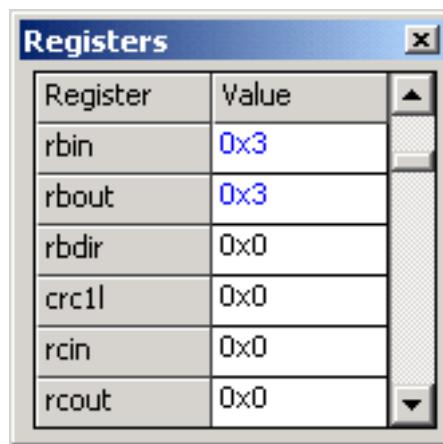
`main` is the top-level function of the `starter` example, so it is the deepest function on the stack. `timer_poll` is called in the main loop of `main`, to update any timers that ipOS is maintaining. `timer_poll` calls `oneshot_tick` to update any one-shot timers. If the timer for incrementing the pattern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



Register	Value
rbin	0x3
rbout	0x3
rbdirection	0x0
crc1	0x0
rcin	0x0
rcout	0x0

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



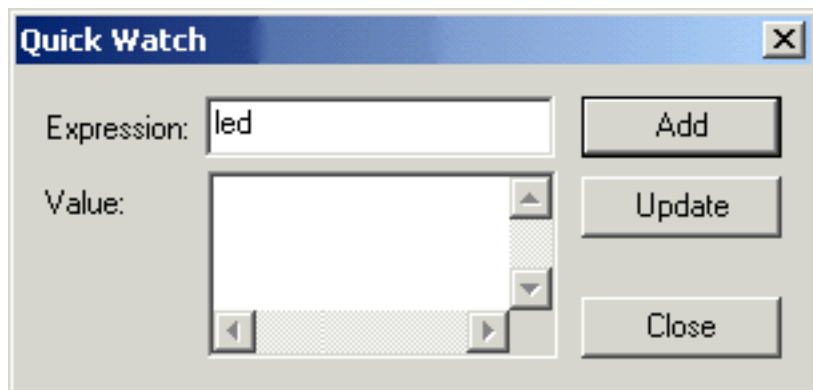


read back into the `rbin` register on every click of the Continue icon, so the values in these registers are highlighted after each click.

## 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.

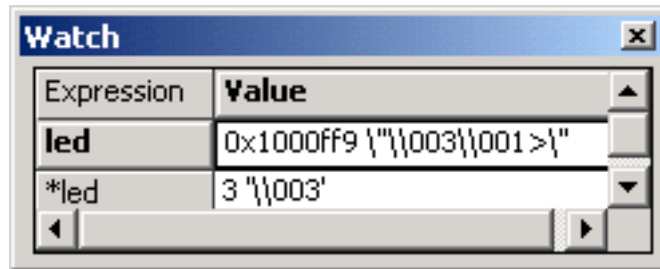
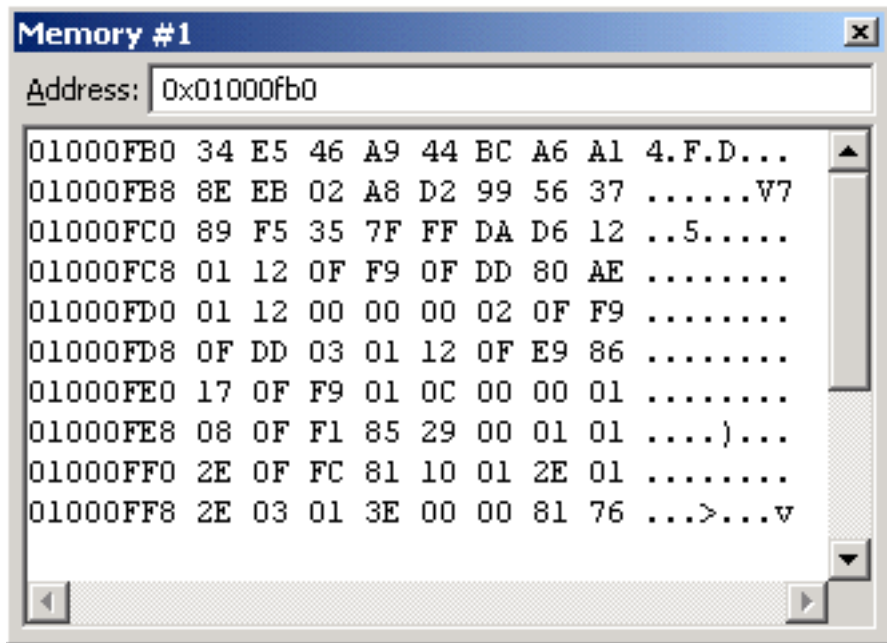


Figure 7-6 Watch Window

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.





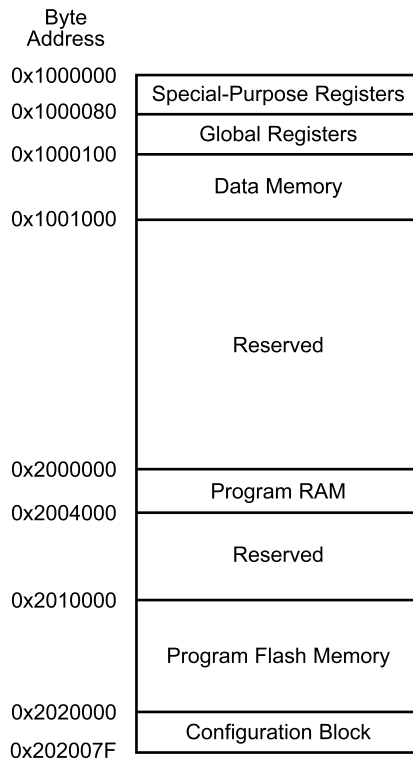
**Figure 7-7 Memory Window**

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



## 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.



515-078.eps

**Figure 7-8 Debugging Memory Space**



# Utilities 8.0

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

- Read the software for an IP2022 device from a **.elf** file
- Read the software for an IP2022 device from the flash memory of a programmed device
- Edit the software in hexadecimal
- Download the software to an IP2022 device

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 hitting the F4 function key. **IP2KProg** will load the executable file of the current project. An executable file must be available, otherwise Unity will not launch **IP2KProg**.

Although Unity will not launch **IP2KProg** during a debugging session, it is possible to enter a debugging session after launching **IP2KProg**. This is not recommended, because it causes loss of synchronization between the Demo Board and the software on the host PC.

**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 OSC oscillator is the only 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.*



## 8.1 IP2KProg User Interface

**Check The XTAL Box!**

**Do Not Touch The Erase Button!**

Figure 8-1 IP2KProg User Interface

Clicking the Erase button disables the Demo Board. *Do not touch!*



**IP2KProg** keeps a memory buffer of the 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. An erase cycle occurs before programming, so it is not necessary to use the Erase button. *Clicking the Erase button disables the Demo Board.* The memory buffer can be downloaded to the configuration block alone by clicking the PgmFuses button.

The memory buffer can be reloaded 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 may be viewed by selecting the Device -> Flash\_allocation command.

To navigate around the memory buffer, 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. Hit 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
```



Hit 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
```





# A.0

## Assembly Language Syntax

This appendix briefly describes the assembly-language syntax and the IP2022-specific features of the GNU assembler.

For a complete description of the non-IP2022-specific features of the assembly-language syntax, see the *GNUPro Toolkit—GNUPro Utilities* manual.

### A.1 Comments, Constants, and Symbols

Comments can occur at the end of a line, after a pound sign (#) or semicolon. The semicolon is recommended, as in:

```
clrb status,c ;this is a comment
;this whole line is a comment
```

Comments can also be enclosed in C-style comment delimiters, such as:

```
/* this is a comment */
```

and:

```
/* this is
a multi-line
comment */
```

As with C, comments may not be nested.



Constants may be character constants, string constants, or numeric constants. A character constant is a single quote followed by a character, such as `'f` which is a byte with the value 102 (decimal) corresponding to its ASCII code. A string constant consists of one or more characters enclosed in double quotes, such as `"UbiCom"`. To use a character with special meaning or a character outside of the standard ASCII printing characters, a backslash (`\`) is used to indicate a representation for the character, as shown in Table A-1.

**Table A-1 Special Characters**

Representation	Value	Character
<code>\b</code>	0x08	Backspace (control-H)
<code>\f</code>	0x0C	Form Feed (control-L)
<code>\n</code>	0x0A	New Line (control-J)
<code>\r</code>	0x0D	Carriage Return (control-M)
<code>\t</code>	0x09	Horizontal Tab (control-I)
<code>\xNN</code>	0xNN	NN is the ASCII code for the character, in hex. E.g. <code>\x09</code> is equivalent to <code>\t</code> .
<code>\\</code>	0x5C	Backslash ( <code>\</code> )
<code>\"</code>	0x22	Double Quote ( <code>"</code> )

A numeric constant is an integer. By default, it is interpreted as a decimal number. To express it in binary, prefix the value with `0b`,



e.g. `0b01101001`. To express it in hex, prefix the value with `0x`, as in `0x9F`. Hex digits may be either upper or lower case.

A symbol is a name for any nameable object, such as labels and constants. A symbol consists of one or more characters from the set of letters, digits, period (`.`), and underscore (`_`). A symbol may not begin with a digit. Symbols are case-sensitive, so `abc` is distinct from `aBc`. Symbols may not be reserved words (see Section A.6).

The assembler has two special constructs for recovering the address of a symbol in data memory. `%lo8data(symbol)` returns the low byte of the address of `symbol`, and `%hi8data(symbol)` returns the high byte. These are useful for initializing pointers used in the IPH/IPL, DPH/DPL, and SPH/SPL registers. Another set of constructs, `%lo8insn(symbol)` and `%hi8insn(symbol)`, are used to recover addresses in program memory. These are useful for initializing pointers used in the ADDRH/ADDRL register.

## A.2 Addressing Modes

The addressing modes are shown in Table A-2. For a more detailed explanation of these modes, see Chapter 3 of the *IP2022 User's Manual*. (The discussion in Chapter 3 covers the addressing modes accessed with the "fr" operand field of the instruction word, so it does not include immediate mode because separate opcodes are used for those instructions that offer an immediate mode operand.)



**Table A-2 Addressing Modes**

Addressing Mode	Assembly Language Examples	Description
Immediate	<code>mov w, #0xff</code>	Immediate operand is the literal value 0xFF (i.e. 255 decimal).
Direct	<code>mov w, 0xff</code> <code>mov 0xff, w</code>	Direct operand is the global register at address 0xFF. Direct addressing can only be used for addresses between 0x01 and 0xFF (i.e. special-purpose registers and global registers).
Indirect	<code>mov w, (ip)</code> <code>mov (ip), w</code>	Indirect operand is the location in data memory addressed by the contents of the IPH/IPL register.
Indirect with Offset, Data Pointer	<code>mov w, 8(dp)</code> <code>mov 8(dp), w</code>	Indirect operand is the location in data memory addressed by the contents of the DPH/DPL register plus an offset of 8. The offset is restricted to a range of 0 to 255. The operand address must be $\geq 0x20$ .
Indirect with Offset, Stack Pointer	<code>mov w, 8(sp)</code> <code>mov 8(sp), w</code>	Indirect operand is the location in data memory addressed by the contents of the SPH/SPL register plus an offset of 8. The offset is restricted to a range of 0 to 255. The operand address must be $\geq 0x20$ .





### A.3 Sections Used in Default Linker Script

The linker script file `ip2ke1f.ld` defines the following sections:

- `.gpr`—general-purpose registers (addresses 0x80 to 0xFF)
- `.data`—preallocated, preinitialized data memory
- `.text`—flash memory
- `.pram`—program space allocated in program RAM
- `.pram_data`—data space allocated in program RAM
- `.strings`—strings stored in flash memory
- `.reset`—reset vector
- `.config`—configuration block
- `.bss`—storage for globals. Initialized to zero.

### A.4 Summary of CPU Instructions

**Table A-3 Key to Abbreviations and Symbols**

Symbol	Description
W	Working register
fr	File register field (an operand specified using direct addressing, indirect addressing, or indirect-with-offset addressing)
PCL	Virtual register for direct PC modification (direct address 0x09)
STATUS	STATUS register (direct address 0x0B)
IPH	Indirect Pointer High - Upper half of pointer for indirect addressing (direct address 0x04)
IPL	Indirect Pointer Low - Lower half of pointer for indirect addressing (direct address 0x05)



**Table A-3 Key to Abbreviations and Symbols**

Symbol	Description
DPH	Upper half of data pointer for indirect-with-offset addressing (direct address 0x0C)
DPL	Lower half of data pointer for indirect-with-offset addressing (direct address 0x0D)
SPH	Upper half of stack pointer for indirect-with-offset addressing (direct address 0x06)
SPL	Lower half of stack pointer for indirect-with-offset addressing (direct address 0x07)
C	Carry bit in the STATUS register (bit 0)
DC	Digit Carry bit in the STATUS register (bit 1)
Z	Zero bit in the STATUS register (bit 2)
BO	Brown-out bit in the STATUS register (bit 3)
WD	Watchdog Timeout bit in the STATUS register (bit 4)
PA2:PA0	Page bits in the STATUS register (bits 7:5)
WDT	Watchdog Timer counter and prescaler
,	File register/bit selector separator (e.g. <code>clrb status, z</code> )
lit8	8-bit immediate operand (i.e. literal) in assembly language instruction
addr13	13-bit address in assembly language instruction
(address)	Contents of memory referenced by address
	Logical OR
	Concatenation
^	Logical exclusive OR
&	Logical AND
!=	inequality



Table A-4 Logical Instructions

Assembler Syntax	Pseudocode Definition	Description	Flags Affected
<code>and fr,w</code>	$fr = fr \& W$	AND fr,W into fr	Z
<code>and w,fr</code>	$fr = W \& fr$	AND W,fr into W	Z
<code>and w,#lit8</code>	$W = W \& lit8$	AND W,literal into W	Z
<code>not fr</code>	$fr = \overline{fr}$	Complement fr into fr	Z
<code>not w,fr</code>	$W = \overline{fr}$	Complement fr into W	Z
<code>or fr,w</code>	$fr = fr   W$	OR fr,W into fr	Z
<code>or w,fr</code>	$W = W   fr$	OR W,fr into W	Z
<code>or w,#lit8</code>	$W = W   lit8$	OR W,literal into W	Z
<code>xor fr,w</code>	$fr = fr \wedge W$	XOR fr,W into fr	Z
<code>xor w,fr</code>	$W = W \wedge fr$	XOR W,fr into W	Z
<code>xor w,#lit8</code>	$W = W \wedge lit8$	XOR W,literal into W	Z

Table A-5 Arithmetic and Shift Instructions

Assembler Syntax	Pseudocode Definition	Description	Flags Affected
<code>add fr,w</code>	$fr = fr + W$	Add fr,W into fr	C, DC, Z
<code>add w,fr</code>	$W = W + fr$	Add W,fr into W	C, DC, Z
<code>add w,#lit8</code>	$W = W + lit8$	Add W,literal into W	C, DC, Z
<code>addc fr,w</code>	$fr = C + fr + W$	Add carry,fr,W into fr	C, DC, Z
<code>addc w,fr</code>	$W = C + W + fr$	Add carry,W,fr into W	C, DC, Z
<code>clr fr</code>	$fr = 0$	Clear fr	Z



**Table A-5 Arithmetic and Shift Instructions** (continued)

<b>Assembler Syntax</b>	<b>Pseudocode Definition</b>	<b>Description</b>	<b>Flags Affected</b>
<code>cmp w,fr</code>	$fr - W$	Compare W,fr then update STATUS	C, DC, Z
<code>cmp w,#lit8</code>	$lit8 - W$	Compare W,literal then update STATUS	C, DC, Z
<code>cse w,fr</code>	if $(fr - W) = 0$ then skip	Compare W,fr then skip if equal	None
<code>cse w,#lit8</code>	if $(lit8 - W) = 0$ then skip	Compare W,literal then skip if equal	None
<code>csne w,fr</code>	if $(fr - W) \neq 0$ then skip	Compare W,fr then skip if not equal	None
<code>csne w,#lit8</code>	if $(lit8 - W) \neq 0$ then skip	Compare W,literal then skip if not equal	None
<code>cwdt</code>	$WDT = 0$	Clear Watchdog Timer	None
<code>dec fr</code>	$fr = fr - 1$	Decrement fr into fr	Z
<code>dec w,fr</code>	$W = fr - 1$	Decrement fr into W	Z
<code>decsnz fr</code>	$fr = fr - 1$ if $fr \neq 0$ then skip	Decrement fr into fr then skip if not zero (STATUS not updated)	None
<code>decsnz w,fr</code>	$W = fr - 1$ if $fr \neq 0$ then skip	Decrement fr into W then skip if not zero (STATUS not updated)	None
<code>decsz fr</code>	$fr = fr - 1$ if $fr = 0$ then skip	Decrement fr into fr then skip if zero (STATUS not updated)	None



**Table A-5 Arithmetic and Shift Instructions** (continued)

<b>Assembler Syntax</b>	<b>Pseudocode Definition</b>	<b>Description</b>	<b>Flags Affected</b>
<code>dec sz w, fr</code>	$W = fr - 1$ if $fr = 0$ then skip	Decrement $fr$ into $W$ then skip if zero (STATUS not updated)	None
<code>inc fr</code>	$fr = fr + 1$	Increment $fr$ into $fr$	Z
<code>inc w, fr</code>	$W = fr + 1$	Increment $fr$ into $W$	Z
<code>inc snz fr</code>	$fr = fr + 1$ if $fr \neq 0$ then skip	Increment $fr$ into $fr$ then skip if not zero (STATUS not updated)	None
<code>inc snz w, fr</code>	$W = fr + 1$ if $fr \neq 0$ then skip	Increment $fr$ into $W$ then skip if not zero (STATUS not updated)	None
<code>inc sz fr</code>	$fr = fr + 1$ if $fr = 0$ then skip	Increment $fr$ into $fr$ then skip if zero (STATUS not updated)	None
<code>inc sz w, fr</code>	$W = fr + 1$ if $fr = 0$ then skip	Increment $fr$ into $W$ then skip if zero (STATUS not updated)	None
<code>mul s w, fr</code>	MULH    $W = W \times fr$	Signed $8 \times 8$ multiply (bit 7 = sign) $W, fr$ into MULH    $W$	None
<code>mul s w, #lit8</code>	MULH    $W = W \times lit8$	Signed $8 \times 8$ multiply (bit 7 = sign) $W, literal$ into MULH    $W$	None
<code>mul u w, fr</code>	MULH    $W = W \times fr$	Unsigned $8 \times 8$ multiply $W, fr$ into MULH    $W$	None
<code>mul u w, #lit8</code>	MULH    $W = W \times lit8$	Unsigned $8 \times 8$ multiply $W, literal$ into MULH    $W$	None



**Table A-5 Arithmetic and Shift Instructions** (continued)

Assembler Syntax	Pseudocode Definition	Description	Flags Affected
<code>rl fr</code>	$fr \parallel C = C \parallel fr$	Rotate <code>fr</code> left through carry into <code>fr</code>	C
<code>rl w,fr</code>	$W \parallel C = C \parallel fr$	Rotate <code>fr</code> left through carry into <code>W</code>	C
<code>rr fr</code>	$C \parallel fr = fr \parallel C$	Rotate <code>fr</code> right through carry into <code>fr</code>	C
<code>rr w,fr</code>	$C \parallel W = fr \parallel C$	Rotate <code>fr</code> right through carry into <code>W</code>	C
<code>sub fr,w</code>	$fr = fr - W$	Subtract <code>W</code> from <code>fr</code> into <code>fr</code>	C, DC, Z
<code>sub w,fr</code>	$W = fr - W$	Subtract <code>W</code> from <code>fr</code> into <code>W</code>	C, DC, Z
<code>sub w,#lit8</code>	$W = lit8 - W$	Subtract <code>W</code> from literal into <code>W</code>	C, DC, Z
<code>subc fr,w</code>	$fr = fr - \overline{C} - W$	Subtract $\overline{carry}, W$ from <code>fr</code> into <code>fr</code>	C, DC, Z
<code>subc w,fr</code>	$W = fr - \overline{C} - W$	Subtract $\overline{carry}, W$ from <code>fr</code> into <code>W</code>	C, DC, Z
<code>swap fr</code>	$fr = fr3:0 \parallel fr7:4$	Swap high and low nibbles of <code>fr</code> into <code>fr</code>	None
<code>swap w,fr</code>	$W = fr3:0 \parallel fr7:4$	Swap high and low nibbles of <code>fr</code> into <code>W</code>	None
<code>test fr</code>	if <code>fr</code> = 0 then Z = 1 else Z = 0	Test <code>fr</code> for zero and update Z	Z



**Table A-6 Bit Operation Instructions**

<b>Assembler Syntax</b>	<b>Pseudocode Definition</b>	<b>Description</b>	<b>Flags Affected</b>
<code>clrb fr.bit</code>	<code>fr.bit = 0</code>	Clear bit in fr	None
<code>sb fr.bit</code>	if <code>fr.bit = 1</code> then skip	Test bit in fr then skip if set	None
<code>setb fr.bit</code>	<code>fr.bit = 1</code>	Set bit in fr	None
<code>snb fr.bit</code>	if <code>fr.bit = 0</code> then skip	Test bit in fr then skip if clear	None

**Table A-7 Data Movement Instructions**

<b>Assembler Syntax</b>	<b>Pseudocode Definition</b>	<b>Description</b>	<b>Flags Affected</b>
<code>mov fr,w</code>	<code>fr = W</code>	Move W into fr	None
<code>mov w,fr</code>	<code>W = fr</code>	Move fr into W	Z
<code>mov w,#lit8</code>	<code>W = lit8</code>	Move literal into W	None
<code>push fr</code>	<code>(SP) = fr</code> <code>SP = SP - 1</code>	Move fr onto top of stack	None
<code>push #lit8</code>	<code>(SP) = lit8</code> <code>SP = SP - 1</code>	Move literal onto top of stack	None
<code>pop fr</code>	<code>SP = SP + 1</code> <code>fr = (SP)</code>	Move top of stack into fr	None



**Table A-8 Program Control Instructions**

<b>Assembler Syntax</b>	<b>Description</b>	<b>Flags Affected</b>
<code>call addr13</code>	Call subroutine	None
<code>jmp addr13</code>	Jump	None
<code>int</code>	Software interrupt	None
<code>nop</code>	No operation	None
<code>ret</code>	Return from subroutine	PA2:0
<code>reti #lit3</code>	Return from interrupt	All
<code>retw #lit8</code>	Return from subroutine with literal into W	PA2:0

**Table A-9 System Control Instructions**

<b>Assembler Syntax</b>	<b>Description</b>	<b>Flags Affected</b>
<code>break</code>	Software breakpoint	None
<code>ferase</code>	Erase flash block	None
<code>fread</code>	Read from flash memory	None
<code>fwrite</code>	Write into flash memory	None
<code>iread</code>	Read from program memory	None
<code>iwrite</code>	Write into program RAM	None
<code>loadh addr16</code>	Load high data address into DPH	None
<code>loadl addr16</code>	Load low data address into DPH	None
<code>page addr16</code>	Load page bits from program address	PA2:0
<code>speed #lit8</code>	Change CPU speed from literal	None





## A.5 Summary of CPU and Peripheral Registers

**Table A-10 Register Addresses and Reset State**

Address	Name	Description	Reset Value
0x0001	Reserved	Reserved	Reserved
0x0002	Reserved	Reserved	Reserved
0x0003	Reserved	Reserved	Reserved
0x0004	IPH	Indirect Pointer (high byte)	00000000
0x0005	IPL	Indirect Pointer (low byte)	00000000
0x0006	SPH	Stack Pointer (high byte)	00000000
0x0007	SPL	Stack Pointer (low byte)	00000000
0x0008	PCH	Current PC bits 15:8 (read-only)	11111111
0x0009	PCL	Virtual register for direct PC modification	11110000
0x000A	W	W register	00000000
0x000B	STATUS	STATUS register	On POR or RST Reset: 11100000 On Brown-out Reset: 11101000 On WDT Overflow: 11110000
0x000C	DPH	Data Pointer (high byte)	00000000
0x000D	DPL	Data Pointer (low byte)	00000000



**Table A-10 Register Addresses and Reset State** (continued)

Address	Name	Description	Reset Value
0x000E	SPDREG	Current speed (read-only)	10010011
0x000F	MULH	Multiply result (high byte)	00000000
0x0010	ADDRH	Program memory address (high byte)	00000000
0x0011	ADDRL	Program memory address (low byte)	00000000
0x0012	DATAH	Program memory data (high byte)	00000000
0x0013	DATAL	Program memory data (low byte)	00000000
0x0014	INTVECH	Interrupt vector (high byte)	00000000
0x0015	INTVECL	Interrupt vector (low byte)	00000000
0x0016	INTSPD	Interrupt speed register	00000000
0x0017	RBINTF	Port B interrupt flags	00000000
0x0018	RBINTE	Port B interrupt enable bits	00000000
0x0019	RBINTED	Port B interrupt edge select bits	00000000
0x001A	FCFG	Flash configuration register	00000000
0x001B	TCTRL	Timer 1/2 common control register	00000000
0x001C	XCFG	Extended configuration	00000001
0x001D	Reserved	Reserved	Reserved
0x001E	IPCH	Interrupt return address (high byte)	00000000
0x001F	IPCL	Interrupt return address (low byte)	00000000
0x0020	RAIN	Data on Port A pins	N/A
0x0021	RAOUT	Port A output latch	00000000
0x0022	RADIR	Port A direction register	11111111
0x0023	Reserved	Reserved	Reserved
0x0024	RBIN	Data on Port B pins	N/A



**Table A-10 Register Addresses and Reset State** (continued)

Address	Name	Description	Reset Value
0x0025	RBOUT	Port B output latch	00000000
0x0026	RBDIR	Port B direction register	11111111
0x0027	Reserved	Reserved	Reserved
0x0028	RCIN	Data on Port C pins	N/A
0x0029	RCOUT	Port C output latch	00000000
0x002A	RCDIR	Port C direction register	11111111
0x002B	Reserved	Reserved	Reserved
0x002C	RDIN	Data on Port D pins	N/A
0x002D	RDOUT	Port D output latch	00000000
0x002E	RDDIR	Port D direction register	11111111
0x002F	Reserved	Reserved	Reserved
0x0030	REIN	Data on Port E pins	N/A
0x0031	REOUT	Port E output latch	00000000
0x0032	REDIR	Port E direction register	11111111
0x0033	Reserved	Reserved	Reserved
0x0034	RFIN	Data on Port F pins	N/A
0x0035	RFOUT	Port F output latch	00000000
0x0036	RFDIR	Port F direction register	11111111
0x0037	Reserved	Reserved	Reserved
0x0038	Reserved	Reserved	Reserved
0x0039	RGOUT	Port G output latch	00000000
0x003A	RGDIR	Port G direction register	11111111
0x003B	Reserved	Reserved	Reserved
0x003C	Reserved	Reserved	Reserved



**Table A-10 Register Addresses and Reset State** (continued)

Address	Name	Description	Reset Value
0x003D	Reserved	Reserved	Reserved
0x003E	Reserved	Reserved	Reserved
0x003F	Reserved	Reserved	Reserved
0x0040	RTTMR	Real-time timer value	00000000
0x0041	RTCFCG	Real-time timer configuration register	00000000
0x0042	T0TMR	Timer 0 value	00000000
0x0043	T0CFG	Timer 0 configuration register	00000000
0x0044	T1CNTH	Timer 1 counter register (high byte, read-only)	00000000
0x0045	T1CNTL	Timer 1 counter register (low byte, read-only)	00000000
0x0046	T1CAP1H	Timer 1 Capture 1 register (high byte, read-only)	00000000
0x0047	T1CAP1L	Timer 1 Capture 1 register (low byte, read-only)	00000000
0x0048	T1CAP2H T1CMP2H	Timer 1 Capture 2 (high byte) Timer 1 Compare 2 (high byte)	00000000
0x0049	T1CAP2L T1CMP2L	Timer 1 Capture 2 (low byte) Timer 1 Compare 2 (low byte)	00000000
0x004A	T1CMP1H	Timer 1 Compare 1 register (high byte)	00000000
0x004B	T1CMP1L	Timer 1 Compare 1 register (low byte)	00000000



**Table A-10 Register Addresses and Reset State** (continued)

Address	Name	Description	Reset Value
0x004C	T1CFG1H	Timer 1 configuration register 1 (high byte)	00000000
0x004D	T1CFG1L	Timer 1 configuration register 1 (low byte)	00000000
0x004E	T1CFG2H	Timer 1 configuration register 2 (high byte)	00000000
0x004F	T1CFG2L	Timer 1 configuration register 2 (low byte)	00000000
0x0050	ADCH	ADC value (high byte)	00000000
0x0051	ADCL	ADC value (low byte)	00000000
0x0052	ADCCFG	ADC configuration register	00000000
0x0053	ADCTMR	ADC timer register	00000000
0x0054	T2CNTH	Timer 2 counter register (high byte, read-only)	00000000
0x0055	T2CNTL	Timer 2 counter register (low byte, read-only)	00000000
0x0056	T2CAP1H	Timer 2 Capture 1 register (high byte, read-only)	00000000
0x0057	T2CAP1L	Timer 2 Capture 1 register (low byte, read-only)	00000000
0x0058	T2CAP2H T2CMP2H	Timer 2 Capture 2 (high byte) Timer 2 Compare 2 (high byte)	00000000
0x0059	T2CAP2L T2CMP2L	Timer 2 Capture 2 (low byte) Timer 2 Compare 2 (low byte)	00000000



**Table A-10 Register Addresses and Reset State** (continued)

Address	Name	Description	Reset Value
0x005A	T2CMP1H	Timer 2 Compare 1 register (high byte)	00000000
0x005B	T2CMP1L	Timer 2 Compare 1 register (low byte)	00000000
0x005C	T2CFG1H	Timer 2 configuration register 1 (high byte)	00000000
0x005D	T2CFG1L	Timer 2 configuration register 1 (low byte)	00000000
0x005E	T2CFG2H	Timer 2 configuration register 2 (high byte)	00000000
0x005F	T2CFG2L	Timer 2 configuration register 2 (low byte)	00000000
0x0060	S1TMRH	Serializer 1 clock timer register (high byte)	00000000
0x0061	S1TMRL	Serializer 1 clock timer register (low byte)	00000000
0x0062	S1TBUFH	Serializer 1 transmit buffer (high byte)	00000000
0x0063	S1TBUFL	Serializer 1 transmit buffer (low byte)	00000000
0x0064	S1TCFG	Serializer 1 transmit configuration	00000000
0x0065	S1RCNT	Serializer 1 received bit count (actual) (read-only)	00000000
0x0066	S1RBUFH	Serializer 1 receive buffer (high byte)	00000000
0x0067	S1RBUFL	Serializer 1 receive buffer (low byte)	00000000



**Table A-10 Register Addresses and Reset State** (continued)

Address	Name	Description	Reset Value
0x0068	S1RCFG	Serializer 1 receive configuration	00000000
0x0069	S1RSYNC	Serializer 1 receive bit sync pattern	00000000
0x006A	S1INTF	Serializer 1 status/interrupt flags	00000000
0x006B	S1INTE	Serializer 1 interrupt enable bits	00000000
0x006C	S1MODE	Serializer 1 serial mode/clock select register	00000000
0x006D	S1SMASK	Serializer 1 receive sync mask	00000000
0x006E	PSPCFG	Parallel slave peripheral configuration register	00000000
0x006F	CMPCFG	Comparator configuration register	00000000
0x0070	S2TMRH	Serializer 2 clock timer register (high byte)	00000000
0x0071	S2TMRL	Serializer 2 clock timer register (low byte)	00000000
0x0072	S2TBUFH	Serializer 2 transmit buffer (high byte)	00000000
0x0073	S2TBUFL	Serializer 2 transmit buffer (low byte)	00000000
0x0074	S2TCFG	Serializer 2 transmit configuration	00000000
0x0075	S2RCNT	Serializer 2 received bit count (actual) (read-only)	00000000
0x0076	S2RBUFH	Serializer 2 receive buffer (high byte)	00000000
0x0077	S2RBUFL	Serializer 2 receive buffer (low byte)	00000000



**Table A-10 Register Addresses and Reset State** (continued)

Address	Name	Description	Reset Value
0x0078	S2RCFG	Serializer 2 receive configuration	00000000
0x0079	S2RSYNC	Serializer 2 receive bit sync pattern	00000000
0x007A	S2INTF	Serializer 2 status/interrupt flags	00000000
0x007B	S2INTE	Serializer 2 interrupt enable bits	00000000
0x007C	S2MODE	Serializer 2 serial mode/clock select register	00000000
0x007D	S2SMASK	Serializer 2 receive sync mask	00000000
0x007E	CALLH	Top of call stack (high byte)	11111111
0x007F	CALLL	Top of call stack (low byte)	11111111

## A.6 List of IP2022-Specific Reserved Words

The following list shows all of the instruction mnemonic names and special-purpose register names. Both the uppercase and lowercase versions of these names are reserved words. Reserved words may not be used as symbolic names.

```
adccfg adch adcl adctmr add addc addrh addrl
and break call callh calll clr clrb cmp
cmpcfg cse csne cwdt datah datal dec decsnz
decsz dph dpl fcfg ferase fread fwrite inc
incsnz incsz int intspd intvech intvecl ipch
ipcl iph ipl iread iwrite jmp loadh loadl mov
mulh muls mulu nop not or page pch pcl pop
pspcfg push radir rain raout rbdire rbin
```





```
rbinte rbinted rbintf rbout rcdir rcin rcout
rddir rdin rdout redir rein reout ret rfdir
rfin rfout rgdir rgout rl rr rtcfg rttmr
slinte slintf slmode slrbufh slrbufl slrcfg
slrcnt slrsync slsmask sltbufh sltbufl
sltcfg sltmrh sltmrl s2inte s2intf s2mode
s2rbufh s2rbufl s2rcfg s2rcnt s2rsync
s2smask s2tbufh s2tbufl s2tcfg s2tmrh s2tmrl
sb setb snb spdreg speed sph spl status sub
subc swap t0cfg t0tmr t1cap1h t1cap1l t1cap2h
t1cap2l t1cfg1h t1cfg1l t1cfg2h t1cfg2l
t1cmp1h t1cmp1l t1cmp2h t1cmp2l t1cnth
t1cntl t2cap1h t2cap1l t2cap2h t2cap2l
t2cfg1h t2cfg1l t2cfg2h t2cfg2l t2cmp1h
t2cmp1l t2cmp2h t2cmp2l t2cnth t2cntl tctrl
test wreg xcfg xor
```

## A.7 Directives

The IP2022 assembler has four directives in addition to the standard list:

- `.word`—four bytes
- `.long`—four bytes
- `.half`—two bytes
- `.short`—two bytes



## A.8 In-Line Assembly in C Source Files

In-line assembly code is embedded in a C source file using the `asm` statement. To prevent the compiler from re-ordering instructions during its optimization phase, keep blocks of assembly language instructions together in a single `asm` statement with the `volatile` qualifier, as shown in the following example:

```
asm volatile ("  
    1:  sb      S2INTF,4  
       page   1b  
       jmp    1b  
       clrb   S2INTF,4  
       clrb   STATUS,0  
       rl    w,%0  
       mov    S2TBUFL,w  
       mov    w,#1  
       rl    wreg  
       mov    S2TBUFH,w"  
    :/* No output */  
    : "rs" (data)  
);
```

For more information about the interface between C and in-line assembly language, see the *GNUPro Toolkit—GNUPro Compiler Tools* manual, pages 253 to 261.

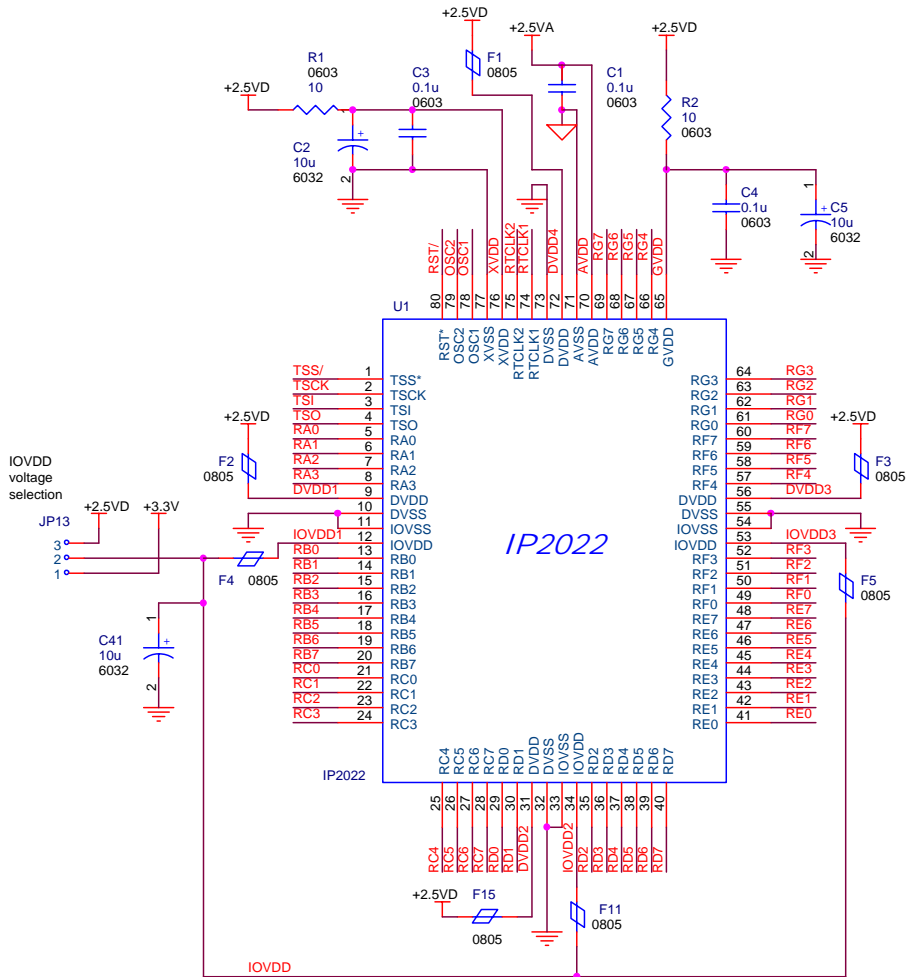
# B.O

## Demo Board Schematics

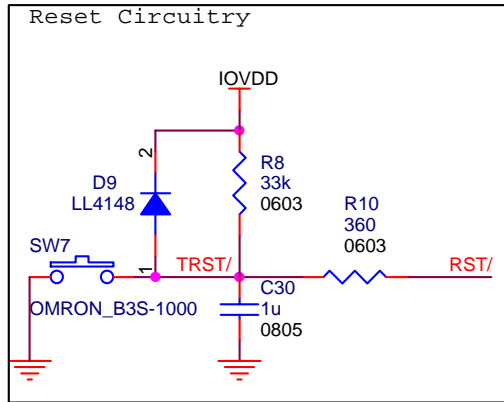
This appendix shows the schematic diagrams for the Demo Board. Resistors R3 and R4 and capacitors C16 through C19 appear in the crystal oscillator circuits, however these components are not required, and their locations on the Demo Board are not populated.



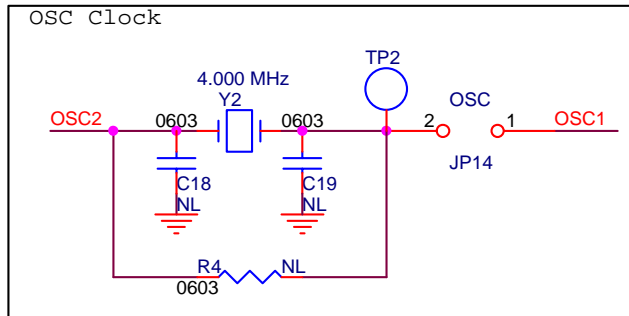
## B.1 IP2022



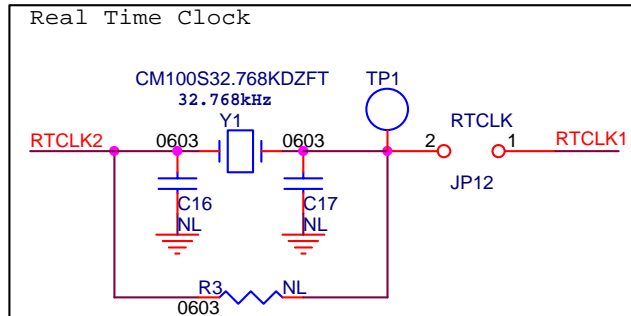
## B.2 Reset



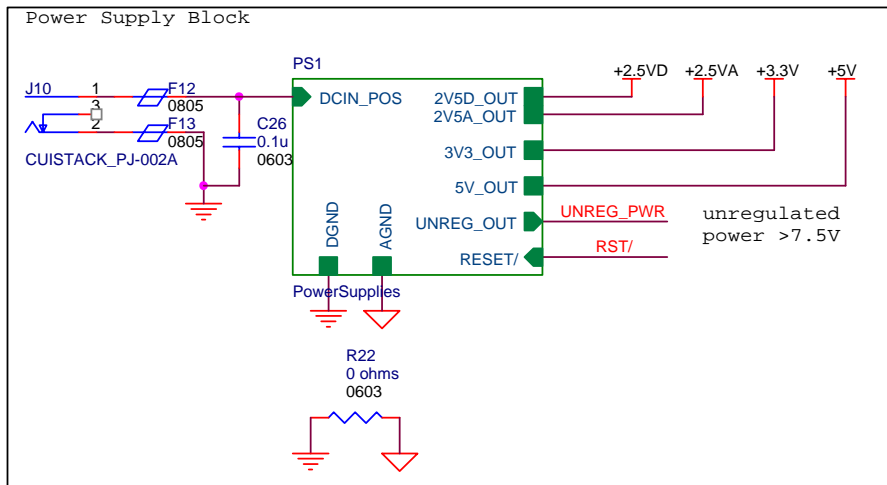
## B.3 Clock



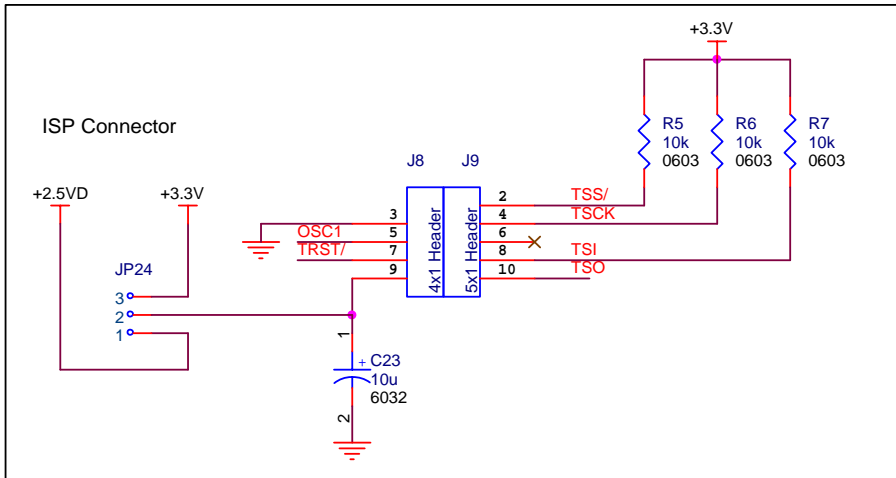
## B.4 RTCLK



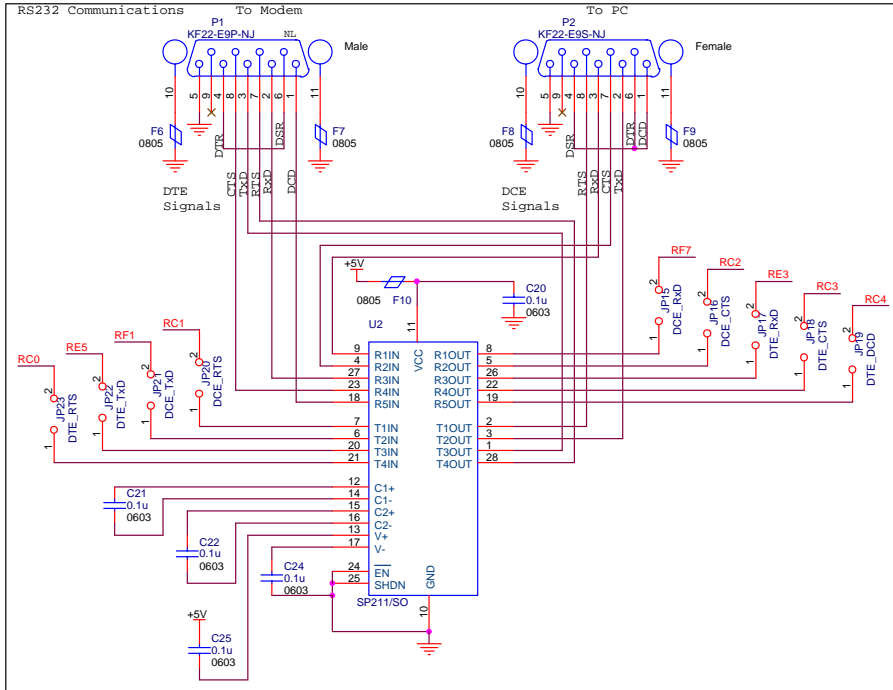
## B.5 Power



## B.6 ISD/ISP

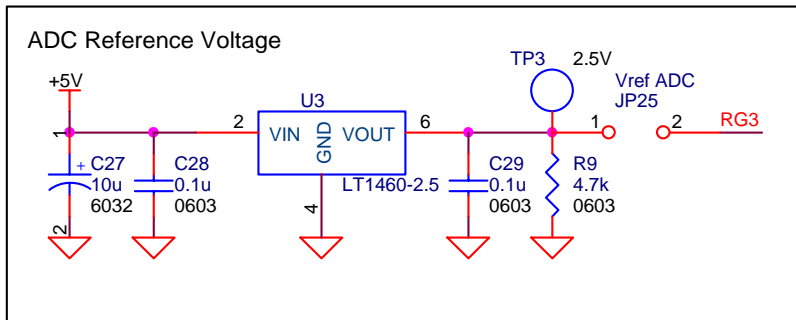
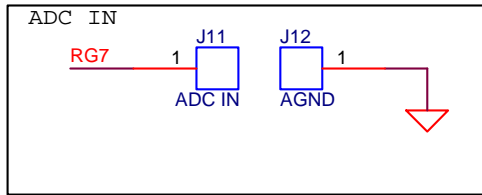


## B.7 RS-232 Communications

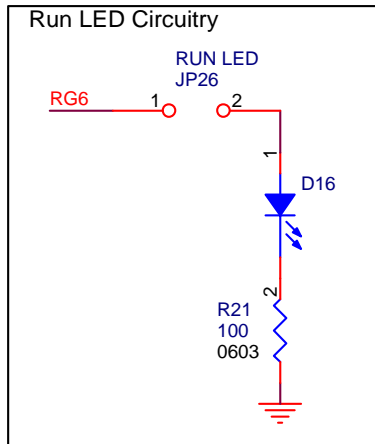




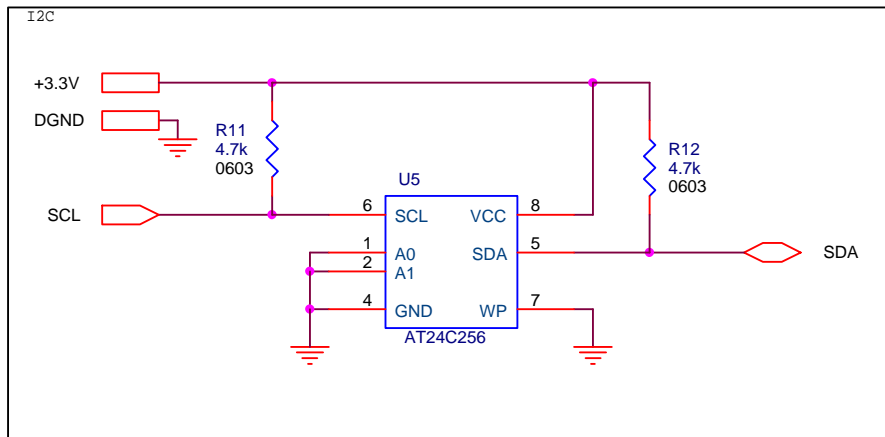
## B.8 ADC and ADC Reference



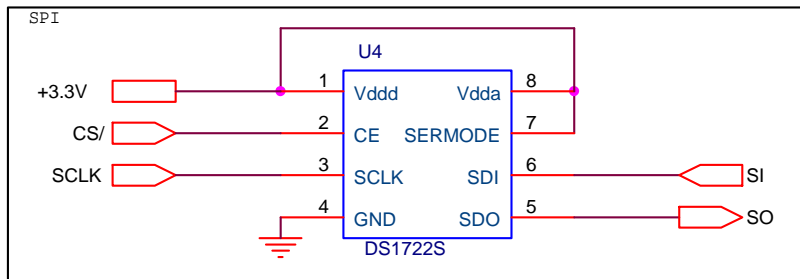
## B.9 Run LED



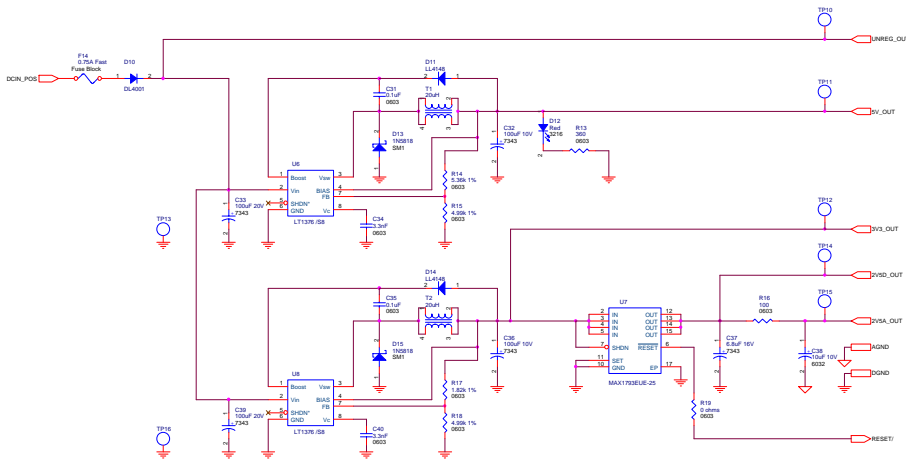
## B.10 I<sup>2</sup>C EEPROM



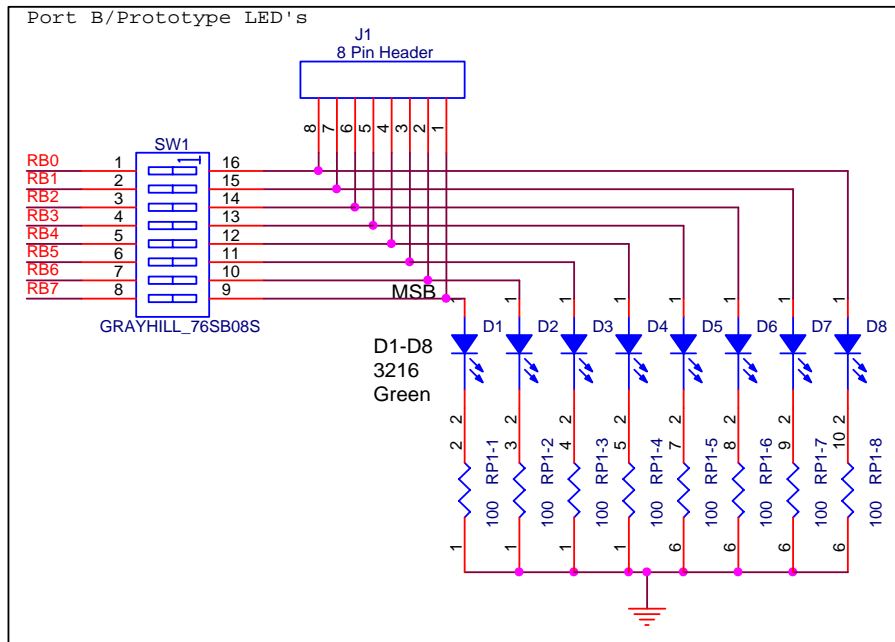
## B.11 SPI Temperature Sensor



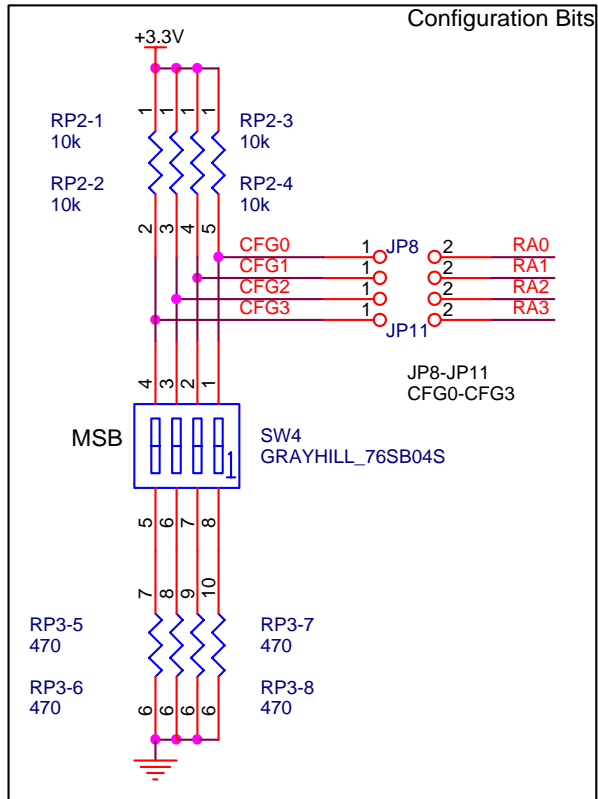
## B.12 On-Board Power Supply

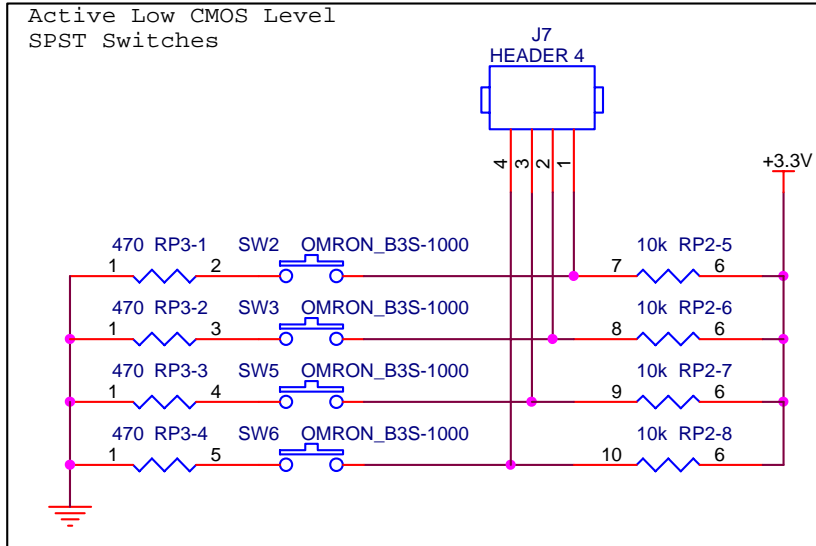


### B.13 User LEDs

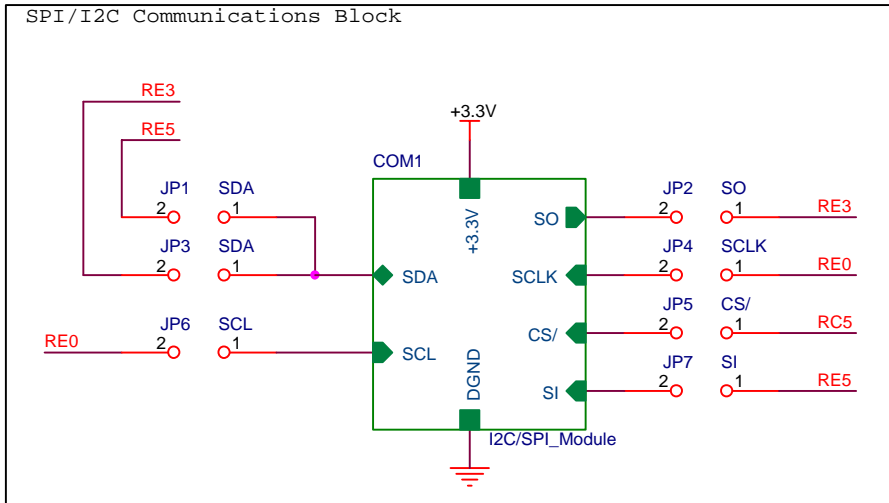


## B.14 Switches

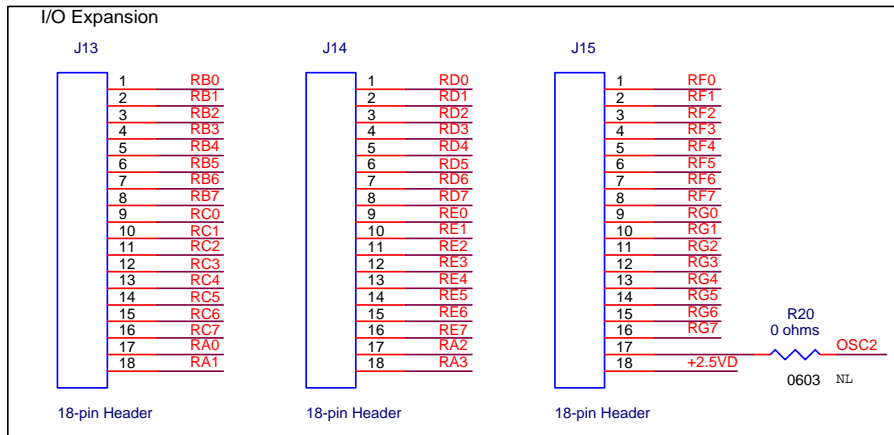




## B.15 Jumpers

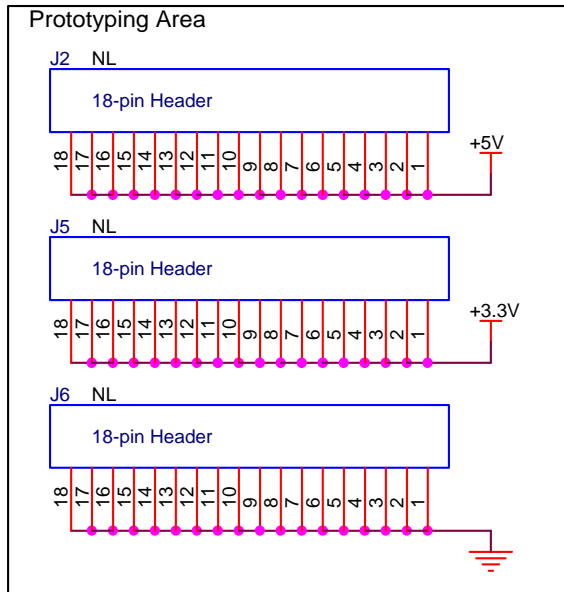


## B.16 Connectors

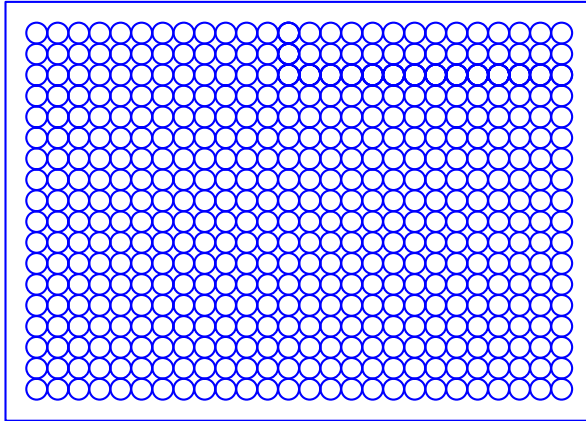




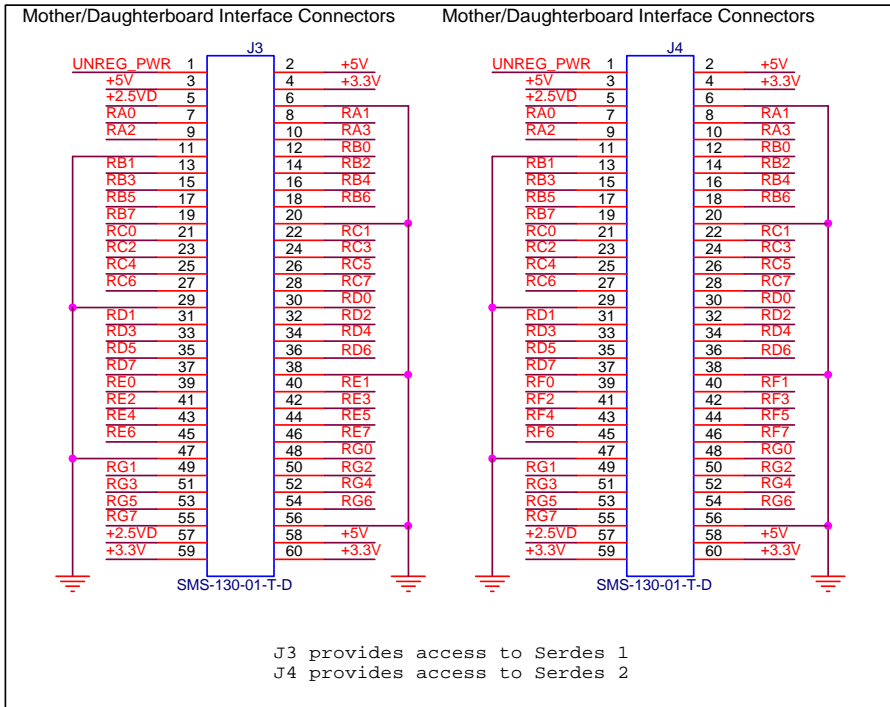
## B.17 Prototype Area



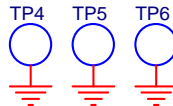
PROTOTYPEAREA-18X26



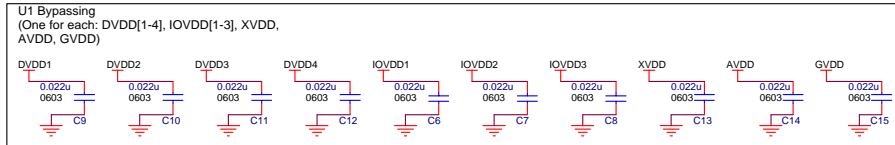
## B.18 Daughter Board Connectors (J3, J4)



## B.19 Test Points



## B.20 Bypass Capacitors



## B.21 Mounting Holes

