Rabbit 2000 Development Kit Getting Started Manual

Part Number 019-0068 • 010118 - D • Printed in U.S.A.

Copyright

© 1999 Rabbit Semiconductor • All rights reserved.

Rabbit Semiconductor reserves the right to make changes and improvements to its products without providing notice.

Trademarks

• Dynamic C® is a registered trademark of Z-World, Inc.
• Windows® is a registered trademark of Microsoft Corporation
• Jackrabbit™ is a trademark of Z-World, Inc.
• Rabbit 2000™ is a trademark of Rabbit Semiconductor

Notice to Users

When a system failure may cause serious consequences, protecting life and property against such consequences with a backup system or safety device is essential. The buyer agrees that protection against consequences resulting from system failure is the buyer’s responsibility.

This device is not approved for life-support or medical systems.

All Rabbit Semiconductor products are 100 percent functionally tested. Additional testing may include visual quality control inspections or mechanical defects analyzer inspections. Specifications are based on characterization of tested sample units rather than testing over temperature and voltage of each unit. Rabbit Semiconductor may qualify components to operate within a range of parameters that is different from the manufacturer’s recommended range. This strategy is believed to be more economical and effective. Additional testing or burn-in of an individual unit is available by special arrangement.

Company Address

Rabbit Semiconductor
2932 Spafford Street
Davis, California  95616-6800
USA
Telephone: (530) 757-8400
Facsimile: (530) 757-8402
Web site: http://www.rabbitsemiconductor.com
# Table of Contents

About This Manual

1. Introduction......................................................................................................................... 1
   1.1 Kit Contents..................................................................................................................... 1
   1.2 Documentation ............................................................................................................... 1
   1.3 An Overview of Dynamic C for the Rabbit .................................................................. 2

2. Detailed Installation Instructions ...................................................................................... 5
   2.1 Software Installation ..................................................................................................... 5
   2.2 Getting Hooked Up ...................................................................................................... 5
     2.2.1 Prototyping Board.................................................................................................. 6
     2.2.2 Jackrabbit Board ................................................................................................. 7
   2.3 Starting Dynamic C ..................................................................................................... 8

3. Sample Programs .............................................................................................................. 9
   3.1 Running Sample Program DEMOJR1.C....................................................................... 10
   3.2 Single-Stepping ......................................................................................................... 11
     3.2.1 Watch Expression................................................................................................. 11
     3.2.2 Break Point ........................................................................................................ 11
     3.2.3 Editing the Program .......................................................................................... 12
     3.2.4 Watching Variables Dynamically....................................................................... 12
     3.2.5 Summary of Features ......................................................................................... 12
   3.3 Cooperative Multitasking ........................................................................................... 13
   3.4 Advantages of Cooperative Multitasking..................................................................... 15

4. Software Reference ......................................................................................................... 17
   4.1 More About Dynamic C.............................................................................................. 17
     4.1.1 Operating System Framework ............................................................................ 17
   4.2 I/O Drivers .................................................................................................................. 18
     4.2.1 Initialization.......................................................................................................... 18
     4.2.2 Digital Output ..................................................................................................... 18
     4.2.3 Analog Output .................................................................................................... 19
     4.2.4 Analog Input ...................................................................................................... 21
   4.3 Serial Communication Drivers ................................................................................... 22
     4.3.1 Open and Close Functions .................................................................................. 22
     4.3.2 Non-Cofunction Blocking Input Functions ......................................................... 23
     4.3.3 Non-Cofunction Blocking Output Functions ....................................................... 24
     4.3.4 Single-User Cofunction Input Functions ............................................................. 25
     4.3.5 Single-User Cofunction Output Functions ......................................................... 26
     4.3.6 Circular Buffer Functions .................................................................................. 27

Appendix A. Specifications ................................................................................................. 29

Schematics
About This Manual

This manual provides instructions for installing, testing, configuring, and interconnecting the Rabbit 2000 microprocessor using the Jackrabbit controller and the Jackrabbit Development board.

Assumptions

Assumptions are made regarding the user's knowledge and experience in the following areas:

- Understanding of the basics of operating a software program and editing files under Windows on a PC.
- Knowledge of basic assembly language and architecture for controllers.

For a full treatment of C, refer to the following texts:


and/or


Pin Number 1

A black square indicates pin 1 of all headers.

Measurements

All diagram and graphic measurements are in inches followed by millimeters enclosed in parenthesis.
1. Introduction

The Rabbit 2000 is a new and powerful microprocessor. Both hardware and software design are easy with the Rabbit.

This kit has the essentials that you need to design your own a microprocessor-based system, and includes a complete software development system (Dynamic C). This Development Kit contains a powerful single-board computer (the Jackrabbit board). With this kit you will be able to write and test complex software. You will be able to prototype circuits that interface to a Rabbit 2000 microprocessor.

1.1 Kit Contents

The items in the kit and their use is as follows:

- CD-ROM with Dynamic C software and Rabbit 2000 documentation. You may install this software by inserting the disk into your CD-ROM drive. If it doesn’t start automatically, click on “setup.exe.” This software runs under Windows ’95, ’98 and Windows NT. We suggest taking the option to load the documentation to your hard disk. The documentation is in HTML or Adobe PDF format, and may be viewed with a browser.

- Jackrabbit controller board. This is a complete controller board that includes a Rabbit 2000 processor, 128K of flash memory and 128K of RAM (Random Access Memory). You can use this board to demonstrate the use of the Rabbit 2000.

- Prototyping Board. The Jackrabbit board can be plugged into this board. The Prototyping Board includes various accessories such as pushbutton switches, LEDs, and a beeper. In addition, you can add your own circuitry.

- Programming cable. This is a cable that is used to connect your PC serial port to the Jackrabbit board to write and debug C programs that run on the Jackrabbit board.

- Loose parts kit. This bag of parts contains parts that you can solder to the Prototyping Board for various demonstrations.

- Wall transformer. This is used to power the Jackrabbit board. The wall transformer is supplied only for Development Kits sold for the North American market. The Jackrabbit board in the Development Kit can also be powered from any DC voltage source between 9 V and 15 V. Higher voltages can be used, but may make the regulator rather hot.

1.2 Documentation

Our documentation is provided in paperless form on the CD-ROM included in the Development Kit. (A paper copy of this “Getting Started” manual is included.) Most documents are provided in two formats: HTML and PDF. HTML documents can be viewed with an internet browser, either Netscape Navigator or Internet Explorer. HTML documents are very convenient because all the documents are hyperlinked together, and it is easy to navigate from one place to another. PDF documents can be viewed using the Adobe Acrobat reader, which is automatically invoked from the browser. The PDF format is best suited for documents requiring high resolution, such as schematics, or if you want to print the document. Don’t print a hardcopy from the HTML manuals because they have no page
1.3 An Overview of Dynamic C for the Rabbit

The Rabbit 2000 is programmed using Z-World’s Dynamic C, an integrated development environment that includes an editor, a C compiler, and a debugger. Library functions provide an easy-to-use interface for the Jackrabbit board included with the Development Kit.

The Jackrabbit board included with the Development Kit is a powerful board that includes a complete Rabbit microprocessor system. A Prototyping Board that includes pushbutton switches, LEDs, and a beeper can be plugged into the Jackrabbit board. By writing programs that run on the Jackrabbit board, you can flash the LEDs, beep the beeper, and otherwise demonstrate the capabilities of the Rabbit. Schematics for both boards are included on the CD-ROM in PDF format.

The Jackrabbit board has a standard Rabbit programming connector, which is a 10-pin, 2 mm header. A programming cable is used to connect a PC serial port (COM port) to the Jackrabbit board. The programming cable has a level converter board in the middle of the cable since the programming connector supports CMOS logic levels, and not the RS-232 levels that are used by PC serial ports. When the programming cable is connected, Dynamic C running on the PC can hard reset the Jackrabbit board and cold boot it. The cold boot includes compiling and downloading a BIOS program that stays resident while you work. If you crash the target, Dynamic C will automatically reboot and recompile the BIOS if it senses that a target communication error occurred.
You have a choice of doing your software development in the flash memory or in the static RAM included on the Jackrabbit board. There are 128K in each memory. Versions of the Jackrabbit board are available that support only 32K of static RAM. If you use one of these boards, you must do development in flash memory. The advantage of working in RAM is to save wear on the flash, which is limited to about 100,000 writes. Note that an application can only be developed in RAM, but cannot run standalone from RAM after the programming cable is disconnected. All applications can only run from flash.

When using flash, the compile to a file is followed by a download to the flash. The disadvantage of using flash is that interrupts must be disabled for approximately 5 ms whenever a break point is set in the program. This can crash fast interrupt routines that are running while you stop at a breakpoint or single-step the program. Flash or RAM is selected on the Options-Compiler menu.

Dynamic C provides a number of debugging features. You can single-step your program, either in C, statement by statement, or in assembly language, instruction by instruction. You can set breakpoints, where the program will stop, on any statement. You can evaluate watch expressions. A watch expression is any C expression that can be evaluated in the context of the program. If the program is at a breakpoint, a watch expression can view any expression using local or external variables. If the program is running and a call to the debugger is included in the user’s code (runwatch();), it is possible to evaluate watch expressions using global variables only while the target program continues to run, slowed down only by the need to refresh a display in response to a <ctrl-U> command.
2. Detailed Installation Instructions

Chapter 2 contains detailed instructions for installing the software on your PC and for connecting the Jackrabbit board to your PC in order to run sample programs.

2.1 Software Installation

You will need approximately 10 megabytes of free space on your hard disk. The software can be installed on your C drive or any other convenient drive.

2.2 Getting Hooked Up

Figure 1 below shows an overview of how the serial and power connections are made to Jackrabbit board, the Prototyping Board, and to your PC.
2.2.1 Prototyping Board

To attach the Jackrabbit board to the Prototyping Board, turn the Jackrabbit board over so that the battery is facing up. Plug headers J4 and J5 into the sockets on the Prototyping Board as indicated in Figure 2.

Figure 2. Attaching Jackrabbit Board to Prototyping Board
2.2.2 Jackrabbit Board

1. Connect the 10-pin PROG connector of the programming cable to header J3 on the Jackrabbit board as shown in Figure 3. (If your programming cable has only one unlabeled 10-pin connector, attach that connector to header J3 on the Jackrabbit board.) Connect the other end of the programming cable to a COM port on your PC. Note that COM1 is the default COM port used by Dynamic C.

![Figure 3. Power and Programming Cable Connections to Jackrabbit Board](image)

2. Hook up the connector from the wall transformer to header J1 on the Jackrabbit board as shown in Figure 3. The orientation of this connector is not important since the VIN (positive) voltage is the middle pin, and GND is available on both ends of the three-pin header J1.

3. Plug in the wall transformer. The Jackrabbit board and the Prototyping Board are ready to be used.

- A RESET button is provided on the Prototyping Board (see Figure 2) to allow a hardware reset.
2.3 Starting Dynamic C

Once the Jackrabbit board is connected as described in Section 2.2, start Dynamic C by double-clicking on the Dynamic C icon or by double-clicking on `dwc.exe` in the Dynamic C directory.

Dynamic C assumes, by default, that you are using serial port COM1 on your PC. If you are using COM1, then Dynamic C should detect the Jackrabbit board and go through a sequence of steps to cold-boot the Jackrabbit board and to compile the BIOS. If an error message appears, you have probably connected to a different PC serial port such as COM2, COM3, or COM4. You can change the serial port used by Dynamic C with the OPTIONS menu, then try to get Dynamic C to recognize the Jackrabbit board by selecting Recompile BIOS on the Compile menu. Try the different COM ports in the OPTIONS menu until you find the one you are connected to. If you can’t get Dynamic C to recognize the target on any port, then the hookup may be wrong or the COM port is not working on your PC.

If you receive the “BIOS successfully compiled …” message after pressing <ctrl-Y> or starting Dynamic C, and this message is followed by “Target not responding,” it is possible that your PC cannot handle the 115,200 bps baud rate. Try changing the baud rate to 57,600 bps as follows.

1. Open the BIOS source code file `RABBITBIOS.C` in the BIOS directory.
2. Change the line
   ```c
   #define USE115KBAUD 1 // set to 0 to use 57600 baud
   ```
   to read as follows.
   ```c
   #define USE115KBAUD 0 // set to 0 to use 57600 baud
   ```
3. Locate the Serial options dialog in the Dynamic C Options menu. Change the baud rate to 57,600 bps, then press <ctrl-Y>.

If you receive the “BIOS successfully compiled …” message and do not receive a “Target not responding” message, the target is now ready to compile a user program.
3. Sample Programs

A series of sample programs is provided in the Dynamic C `Samples/JackRab` folder. You can load a sample program by using the **File Open** menu in Dynamic C. The sample programs are listed in Table 1.

<table>
<thead>
<tr>
<th>Table 1. Jackrabbit Sample Programs</th>
</tr>
</thead>
<tbody>
<tr>
<td>DEMOJR1.C</td>
</tr>
<tr>
<td>DEMOJR2.C</td>
</tr>
<tr>
<td>DEMOJR3.C</td>
</tr>
<tr>
<td>DEMOJR6.C</td>
</tr>
<tr>
<td>JRIOTEST.C</td>
</tr>
<tr>
<td>JRIO_COF.C</td>
</tr>
<tr>
<td>RABDB01.C</td>
</tr>
<tr>
<td>RABDB02.C</td>
</tr>
</tbody>
</table>

The first five sample programs provide a step-by-step introduction to the Jackrabbit board. Additional sample programs illustrate more advanced topics.

Each sample program has comments that describe the purpose and function of the program.
### 3.1 Running Sample Program DEMOJR1.C

This sample program can be used to illustrate some of the functions of Dynamic C.

First, open the file `DEMOJR1.C`, which is in the `Samples/JackRab` folder. The program will appear in a window, as shown in Figure 4 below (minus some comments). Use the mouse to place the cursor on the function name `WrPortI` in the program and type `<ctrl-H>`. This will bring up a documentation box for the function `WrPortI`. In general, you can do this with all functions in Dynamic C libraries, including libraries you write yourself. Close the documentation box and continue.

```c
main(){
    WrPortI(SPCR,NULL,0x84);
    WrPortI(PADR,&PADRShadow,0xff);
    while(1) {
        BitWrPortI(PADR,&PADRShadow,1,2);
        for(j=0; j<25000; j++);
        BitWrPortI(PADR,&PADRShadow,0,2);
        for(j=0; j<1000; j++);
    } // end while(1)
} // end of main
```

Note: See Rabbit 2000 Microprocessor User’s Manual (Software Chapter) for details on the routines that read and write I/O ports.

To run the program `DEMOJR1.C`, load it with the `File` menu, compile it using the `Compile` menu, and then run it by selecting `Run` in the `Run` menu. The LED on the Development Board should start flashing if everything went well. If this doesn’t work review the following points.

- The target should be ready, which is indicated by the message “BIOS successfully compiled...” If you did not receive this message or you get a communication error, recompile the BIOS by typing `<ctrl-Y>` or select `Recompile BIOS` from the `Compile` menu.

Figure 4. Sample Program DEMOJR1.C
• A message reports that “No Rabbit processor detected” in cases where the Jackrabbit and Prototyping Board are not connected together, the wall transformer is not connected, or is not plugged in. (The red power LED lights whenever power is connected.)

• The programming cable must be connected to the Jackrabbit board. (The colored wire on the programming cable is closest to pin 1 on header J3 on the Jackrabbit board, as shown in Figure 3 on page 7.) The other end of the programming cable must be connected to the PC serial port, possibly, using the 9- to 25-pin adapter if necessary. The COM port specified in the Dynamic C Options menu must be the same as the one the programming cable is connected to.

• To check if you have the correct serial port, select Compile, then Compile BIOS, or type <ctrl-Y>. If the “BIOS successfully compiled …” message does not display, try a different serial port using the Dynamic C Options menu until you find the one you are plugged into. Don’t change anything in this menu except the COM number. The baud rate should be 115,200 bps and the stop bits should be 1.

3.2 Single-Stepping

Compile or re-compile DEMOJR1.c by clicking the Compile button on the task bar. The program will compile and the screen will come up with a highlighted character (green) at the first executable statement of the program. Use the F8 key to single-step. Each time the F8 key is pressed, the cursor will advance one statement. When you get to the for(j=0, j< ... statement, it becomes impractical to single-step further because you would have to press F8 thousands of times. We will use this statement to illustrate watch expressions.

3.2.1 Watch Expression

Type <ctrl-W> or chose Add/Del Watch Expression in the Inspect menu. A box will come up. Type the lower case letter j and click on add to top and close. Now continue single-stepping with F8. Each time you step, the watch expression (j) will be evaluated and printed in the watch window. Note how the value of j advances when the statement j++ is executed.

3.2.2 Break Point

Move the cursor to the start of the statement:

    for(j=0; j<1000; j++);

To set a break point on this statement, type F2 or select Breakpoint from the Run menu. A red highlight will appear on the first character of the statement. To get the program running at full speed, type F9 or select Run on the Run menu. The program will advance until it hits the break point. Then the break point will start flashing and show both red and green colors. Note that LED DS3 is now solidly turned on. This is because we have passed the statement turning on LED DS3. Note that j in the watch window has the value 25000. This is because the loop above terminated when j reached 25000.

To remove the break point, type F2 or select Toggle Breakpoint on the Run menu. To continue program execution, type F9 or select Run from the Run menu. Now the LED should be flashing again since the program is running at full speed.
You can set break points while the program is running by positioning the cursor to a statement and using the F2 key. If the execution thread hits the break point, a break point will take place. You can toggle the break point off with the F2 key and continue execution with the F9 key. Try this a few times to get the feel of things.

3.2.3 Editing the Program

Click on the Edit box on the task bar. This will set Dynamic C into the edit mode so that you can change the program. Use the Save as choice on the File menu to save the file with a new name so as not to change the demo program. Save the file as MYTEST.C. Now change the number 25000 in the for (.. statement to 10000. Then use the F9 key to recompile and run the program. The LED will start flashing, but it will flash much faster than before because you have changed the loop counter terminal value from 25000 to 10000.

3.2.4 Watching Variables Dynamically

Go back to edit mode (select edit) and load the program DEMOJR2.C using the File menu Open command. This program is the same as the first program, except that a variable k has been added along with a statement to increment k each time around the endless loop. The statement:

```c
runwatch();
```

has been added. This is a debugging statement that makes it possible to view variables while the program is running.

Use the F9 key to compile and run DEMOJR2.C. Now type <ctrl-W> to open the watch window and add the watch expression k to the top of the list of watch expressions. Now type <ctrl-U>. Each time you type <ctrl-U>, you will see the current value of k, which is incrementing about 5 times a second.

As an experiment add another expression to the watch window:

```c
k*5
```

Then type <ctrl-U> several times to observe the watch expressions k and k*5.

3.2.5 Summary of Features

So far you have practiced using the following features of Dynamic C.

- Loading, compiling and running a program. When you load a program it appears in an edit window. You can compile by selecting Compile on the task bar or from the Compile menu. When you compile the program, it is compiled into machine language and downloaded to the target over the serial port. The execution proceeds to the first statement of main where it pauses, waiting for you to command the program to run, which you can do with the F9 key or by selecting Run on the Run menu. If want to compile and start the program running with one keystroke, use F9, the run command. If the program is not already compiled, the run command will compile it first.

- Single-stepping. This is done with the F8 key. The F7 key can also be used for single-stepping. If the F7 key is used, then descent into subroutines will take place. With the F8 key the subroutine is executed at full speed when the statement that calls it is stepped over.
• Setting break points. The F2 key is used to turn on or turn off (toggle) a break point at
the cursor position if the program has already been compiled. You can set a break point
if the program is paused at a break point. You can also set a break point in a program
that is running at full speed. This will cause the program to break if the execution
thread hits your break point.

• Watch expressions. A watch expression is a C expression that is evaluated on command
in the watch window. An expression is basically any type of C formula that can include
operators, variables and function calls, but not statements that require multiple lines
such as for or switch. You can have a list of watch expressions in the watch window. If
you are single-stepping, then they are all evaluated on each step. You can also com-
mand the watch expression to be evaluated by using the <ctrl-U> command. When a
watch expression is evaluated at a break point, it is evaluated as if the statement was at
the beginning of the function where you are single-stepping. If your program is running
you can also evaluate watch expressions with a <ctrl-U> if your program has a run-
watch() command that is frequently executed. In this case, only expressions involv-
ing global variables can be evaluated, and the expression is evaluated as if it were in a
separate function with no local variables.

3.3 Cooperative Multitasking

Cooperative multitasking is a convenient way to perform several different tasks at the
same time. An example would be to step a machine through a sequence of steps and at the
same time independently carry on a dialog with the operator via a human interface. Coop-
erative multitasking differs from a different approach called preemptive multitasking.
Dynamic C supports both types of multitasking. In cooperative multitasking each separate
task voluntarily surrenders its compute time when it does not need to perform any more
activity immediately. In preemptive multitasking control is forcibly removed from the
task via an interrupt.

Dynamic C has language extensions to support multitasking. The major C constructs are
called costatements, cofunctions, and slicing. These are described more completely in the
Dynamic C Reference Manual. The example below, sample program DEMOJR3.C, uses
costatements. A costatement is a way to perform a sequence of operations that involve
pauses or waits for some external event to take place. A complete description of costate-
ments is in the Dynamic C Reference Manual. The DEMOJR3.C sample program has two
independent tasks. The first task flashes LED DS4 once a second. The second task uses
button S1 on the Prototyping Board to toggle the logical value of a virtual switch,
vswitch, and flash DS1 each time the button is pressed. This task also debounces button
S1.
```c
int vswitch;  // state of virtual switch controlled by button S1
main(){       // begin main program
    // set up parallel port A as output
    WrPortI(SPCR,NULL,0x84);        // turn off all LEDs
    WrPortI(PADR,&PADRShadow,0xff); // initialize virtual switch off
    vswitch=0;  // state of virtual switch controlled by button S1
(1) while (1) {  // Endless loop
    BigLoopTop();  // Begin a big endless loop

    // first task flash LED DS4 every second for 200 milliseconds
    (2) costate {  // begin a costatement
        BitWrPortI(PADR,&PADRShadow,0,3);  // LED DS4 on
        waitfor(DelayMs(200));  // light on for 200 ms
        BitWrPortI(PADR,&PADRShadow,1,3);  // LED DS4 off
        waitfor(DelayMs(800));  // light off for 800 ms
    (4) }  // end of costatement

    // second task - debounce switch #1 and toggle virtual switch vswitch
    // check button 1 and toggle vswitch on or off
    costate {
        (5) if(BitRdPortI(PBDR,2)) abort;  // if button not down skip out
            waitfor(DelayMs(50));  // wait 50 ms
        if(BitRdPortI(PBDR,2)) abort;  // if button not still down skip out
        vswitch=!vswitch;  // toggle virtual switch- button was down 50 ms
        while (1) {  // wait for button to be off 200 ms
            waitfor(BitRdPortI(PBDR,2));  // wait for button to go up
            waitfor(DelayMs(200));  // wait for 200 milliseconds
            if(BitRdPortI(PBDR,2)) break;  // if button up break
        }  // end of while(1)
    }  // end of costatement

    // make LED agree with vswitch if vswitch has changed
    (6) if( (PADRShadow & 1) == vswitch) {
        BitWrPortI(PADR,&PADRShadow,!vswitch,0);
    }
(7) }  // end of while loop, go back to start
}  // end of main, never come here
```

The numbers in the left margin are reference indicators and are not a part of the code.
Load and run the program. Note that LED DS4 flashes once per second. Push button S1 several times and note how LED DS1 is toggled.

The flashing of LED DS4 is performed by the costatement starting at the line marked (2). Costatements need to be executed regularly, often at least every 25 ms. To accomplish this, the costatements are enclosed in a `while` loop. The term while loop is used as a handy way to describe a style of real-time programming in which most operations are done in one loop. The while loop starts at (1) and ends at (7). The function `BigLoopTop()` is used to collect some operations that are helpful to do once on every pass through the loop. Place the cursor on this function name `BigLoopTop()` and hit `<Ctrl-H>` to learn more.
The statement at (3) waits for a time delay, in this case 200 ms. The costatement is being executed on each pass through the big loop. When a `waitfor` condition is encountered the first time, the current value of `MS_TIMER` is saved and then on each subsequent pass the saved value is compared to the current value. If a `waitfor` condition is not encountered, then a jump is made to the end of the costatement (4), and on the next pass of the loop, when the execution thread reaches the beginning of the costatement, execution passes directly to the `waitfor` statement. Once 200 ms has passed, the statement after the `waitfor` is executed. The costatement has the property that it can wait for long periods of time, but not use a lot of execution time. Each costatement is a little program with its own statement pointer that advances in response to conditions. On each pass through the big loop, as little as one statement in the costatement is executed, starting at the current position of the costatement’s statement pointer. Consult the *Dynamic C Reference Manual* for more details.

The second costatement in the program debounces the switch and maintains the variable `vswitch`. Debouncing is performed by making sure that the switch is either on or off for a long enough period of time to ensure that high-frequency electrical hash generated when the switch contacts open or close does not affect the state of the switch. The `abort` statement is illustrated at (5). If executed, the internal statement pointer is set back to the first statement within the costatement, and a jump to the closing brace of the costatement is made.

At (6) a use for a shadow register is illustrated. A shadow register is used to keep track of the contents of an I/O port that is write only - it can’t be read back. If every time a write is made to the port the same bits are set in the shadow register, then the shadow register has the same data as the port register. In this case a test is made to see the state of the LED and make it agree with the state of `vswitch`. This test is not strictly necessary, the output register could be set every time to agree with `vswitch`, but it is placed here to illustrate the concept of a shadow register.

To illustrate the use of snooping, use the watch window to observe `vswitch` while the program is running. Add the variable `vswitch` to the list of watch expressions. Then toggle `vswitch` and the LED. Then type `<ctrl-U>` to observe `vswitch` again.

### 3.4 Advantages of Cooperative Multitasking

Cooperative multitasking, as implemented with language extensions, has the advantage of being intuitive. Unlike preemptive multitasking, variables can be shared between different tasks without having to take elaborate precautions. Sharing variables between tasks is the greatest cause of bugs in programs that use preemptive multitasking. It might seem that the biggest problem would be response time because of the big loop time becoming long as the program grows. Our solution for that is a device caused slicing that is further described in the *Dynamic C Reference Manual*. 
4. Software Reference

4.1 More About Dynamic C

Dynamic C has been in use worldwide since 1989. Dynamic C is specially designed for programming embedded systems. Dynamic C features quick compile and interactive debugging in the real environment. A complete reference to Dynamic C is contained in the *Dynamic C Reference Manual*.

Dynamic C for Rabbit™ processors uses the standard Rabbit programming interface. This is a 10-pin connector that connects to the Rabbit serial port A. It is possible to reset and cold-boot a Rabbit processor via the programming port. No software needs to be present in the target system. More details are available in the *Rabbit 2000 Microprocessor User’s Manual*.

Dynamic C cold-boots the target system and compiles the BIOS. The BIOS is a basic program of a few thousand bytes in length that provides the debugging and communication facilities that Dynamic C needs. Once the BIOS has been compiled, the user can compile his own program and test it. If the BIOS fails because of a crash, a new cold boot and BIOS compile can be done at any time.

Each type of Rabbit microprocessor system can have a different BIOS, or the BIOS program can be customized by using `#define` options. The Jackrabbit board is supplied with one BIOS, and a flash memory and a RAM memory to hold the program. RAM memory is useful for holding a program while debugging is being done because it is more flexible than flash memory.

Dynamic C does not use include files, rather it has libraries which are used for the same purpose, that is, to supply function prototypes to programs before they are compiled. Libraries are much easier to use compared to include files.

Dynamic C supports assembly language, either as separate programs or as fragments embedded in C programs. Interrupt routines may be written in Dynamic C or in assembly language.

4.1.1 Operating System Framework

Dynamic C does not include an operating system in the usual sense of a complex software system that is resident in memory. The user has complete control of what is loaded as a part of his program, other than those routines that support loading and debugging and which are inactive at embedded run time. However, certain routines are very basic and normally should always be present and active.

- Periodic interrupt routine. This interrupt routine is driven by the Rabbit periodic interrupt facility, and when enabled creates an interrupt every 16 ticks of the 32.768 kHz oscillator, or every 488 µs. This routine drives three long global variables that keep track of the time: `SEC_TIMER`, `MS_TIMER`, and `TICK_TIMER` that respectively count seconds, milliseconds, and 488 µs ticks. These variables are needed by virtually all functions that measure time. The `SEC_TIMER` is set to seconds elapsed since 1 Jan 1980, and thus also keeps track of the time and date. The periodic interrupt routine must be disabled when the microprocessor enters sleepy mode and the processor clock...
is operating at 32.768 kHz. The interrupt routine cannot complete at this slow speed before the next tick of the periodic interrupt. In this situation, the hardware real-time clock can be read directly to provide the time.

- Watchdog support routines. Although the Rabbit watchdog can be disabled, this is not recommended since the watchdog is an essential facility for recovering from crashes. Very few systems are crash-free in real life.

### 4.2 I/O Drivers

The Jackrabbit board contains four high-power digital output channels, two D/A converter output channels, and one A/D converter input channel. These I/O channels can be accessed using the functions found in the **JRIO.LIB** library.

#### 4.2.1 Initialization

The function `jrioInit()` must be called before any other function from the **JRIO.LIB** library. This function initializes the digital outputs and sets up the driver for the analog input/outputs. The digital outputs correspond to the Rabbit processor’s port E bits 0–3, and the analog I/O uses timer B; bits 1, 2, and 4 of port D; and bits 6 and 7 of port E.

The function `void jrioInit()` initializes the I/O drivers for Jackrabbit. In particular, it sets up parallel port D bits 1, 2, and 4 for analog output, port E bits 0–3 for digital output, and starts up the pulse-width modulation routines for the A/D and D/A channels. Note that these routines can consume up to 20% of the CPU’s processing power; the routines use timer B and the B1 and B2 match registers.

#### 4.2.2 Digital Output

The Jackrabbit board contains four high-power digital output drivers, HV0–HV3, on header J4. These can be turned on and off with the following functions from the library **JRIO.LIB**.

HV0, HV1, and HV2 are open-collector sinking outputs, and are able to sink up to 1 A (200 mA for the BL1810 and BL1820) from a 30 V source connected to the K line on header J4. HV3 is a sourcing output that is able to source up to 500 mA (100 mA for the BL1810 and BL1820) from a 30 V source connected to the K line.

Remember to cut the trace between K and Vcc inside the outline for header JP2 on the top side of the Prototyping Board if you are supplying K from a separate power supply. An exacto knife, a precision grinder tool, or a screwdriver may be used to cut through the traces as shown in Figure 5.

Failure to do this could lead to the destruction of the Rabbit 2000 microprocessor and other components once the Jackrabbit is connected to the Prototyping Board.

![Figure 5. Cut Trace on Prototyping Board When Vcc and K Are Different](image)
void digOut(int channel, int value)

sets the state of a digital output bit.

jrioInit must be called first.
channel is the output channel number (0-3 on the Jackrabbit).
value is the output value (0 or 1).

void digOn(int channel)

sets the state of a digital output bit to on (1).

jrioInit must be called first.
channel is the output channel number (0–3 on the Jackrabbit).

void digOff(int channel)

sets the state of a digital output bit to off (0).

jrioInit must be called first.
channel is the output channel number (0–3 on the Jackrabbit).

See the sample program JRIOTEST.C for an example of using the digital output functions.

4.2.3 Analog Output

The two analog output channels on the Jackrabbit (DA0 and DA1 on header J5) are controlled by a pulse-width modulation (PWM) driver. This requires the use of some fraction of the CPU cycles when the driver is running (up to 20% when both D/A channels are used). A voltage is selected by giving a value from 0 to 1024 to the driver, corresponding roughly to 0.1 V to 3.5 V on DA0. Because of the PWM interrupt frequency, the PWM driver can provide a continuous range of voltage output in the range from 0.1 V to 3.0 V for DA0, and 0.6 V to 3.6 V for DA1. These ranges can be specified with the constants PWM_MIN, PWM_MAX0, and PWM_MAX1. In other words, setting channel DA0 to the value PWM_MIN will output 0.1 V, and setting it to PWM_MAX0 will output 3.0 V. Similarly, setting DA1 to PWM_MIN will output 0.6 V, and setting it to PWM_MAX1 will output 3.6 V. Values below PWM_MIN will be rounded down to 0, and values above PWM_MAX0 (PWM_MAX1 for DA1) will be rounded up to 1024.

The output channels can also be set in an “always on” or “always off” mode, which does not require CPU cycles. The “always on” mode is set by requesting an output value of 1024, and will provide about 3.4 V on channel DA0, and 3.6 V on DA1. The “always off” mode is selected by asking for a value of 0, and provides an output of around 0.1 V on DA0 and 0.0 V on DA1.
See Table 2 for a summary of the possible analog output voltages corresponding to values given in the `anaOut` function.

**Table 2. Typical Analog Output Voltages Corresponding to Values in anaOut Function**

<table>
<thead>
<tr>
<th>Channel</th>
<th>0</th>
<th>PWM_MIN</th>
<th>PWM_MAX</th>
<th>1024</th>
</tr>
</thead>
<tbody>
<tr>
<td>DA0</td>
<td>0.08 V</td>
<td>0.08 V</td>
<td>2.875 V</td>
<td>3.4 V</td>
</tr>
<tr>
<td>DA1</td>
<td>0.004 V</td>
<td>0.63 V</td>
<td>3.6 V</td>
<td>3.6 V</td>
</tr>
</tbody>
</table>

The output value is set using the following function.

```c
void anaOut(int channel, int value)
```

sets the state of an analog output channel.

- `jrioInit` must be called first.
- `channel` is the output channel number (0 or 1 on the Jackrabbit).
- `value` is an integer from 0–1024 that corresponds to an output voltage as shown in Table 2.

See the sample program `JRIOTEST.C` for examples of using the `anaOut` function.

**Effect of Interrupts on Analog I/O**

The stability of the voltage output (and hence the voltage input determination as well) depends on the ability of the driver to respond quickly to interrupt requests. Dynamic C debugging, use of the `printf` function, or any serial communications can disrupt the pulse-width modulation utilized by the driver and cause fluctuations in the voltage outputs. Avoid using serial communications or `printf` statements during portions of your program where the voltage must remain steady. Also be aware that debugging and running Dynamic C in polling mode will cause fluctuations. Finally, be certain to disable the PWM drivers by setting the output values to 0 or 1024 when you are done using them to free up the CPU.

**Calibration of Values to Voltages**

The analog output channels on the Jackrabbit board can be more accurately calibrated for each individual Jackrabbit board in the following manner (calibration of DA0 is assumed in this example, calibration of DA1 would proceed similarly):

- Set desired channel output to `PWM_MIN`.
- Measure voltage $V_{\text{min}}$ on DA0.
- Set desired channel output to `PWM_MAX0`.
- Measure voltage $V_{\text{max}}$ on DA0.
- A linear relation between input value and voltage can now be calculated:
4.2.4 Analog Input

The analog input channel on the Jackrabbit (AD0 on header J5) works by varying analog output channel DA0 until its voltage matches the input voltage on AD0. DA0 obviously cannot be used while an input voltage is being measured, although channel DA0 is still available. The value returned corresponds to the value that DA0 required to match the input voltage (you would call `anaOut(0,value)` for DA0 to provide that same voltage). If the value returned is negative, then the function considers the value suspect for some reason (most likely a failure of the DA0 voltage to settle quickly). The value can be taken as is, or another measurement can be done.

```c
void anaIn(int channel, int *value)
```

Analog input for the Jackrabbit analog input channel (AD0).

`jrioInit` must be called first.

`channel` is the input channel number (0 only on the Jackrabbit).

An integer between 0 and 1024 will be returned in `value`, corresponding to a voltage obtained if output channel DA0 was set to that value. If a value is found, but the voltage has not appeared to fully settle, the value will be negative (but equal in magnitude to the found voltage) to allow remeasurement if desired.

See sample program `JRIOTEST.C` for an example of the use of `anaIn`.

Two versions of the analog input function are available: the standard function, listed above, that does not return until the measurement has been made, and a cofunction version that can be called from within a costatement. This cofunction version allows other tasks to be performed while the voltage match is being made. The voltage measurement will take ten calls of the cofunction version to make a measurement.

```c
void cof_anaIn(int channel, int *value)
```

The parameters are identical to those described above for `anaIn`.

See sample program `JRIO_COF.C` for an example of the use of `cof_anaIn`.

\[
m = \frac{V_{\text{max}} - V_{\text{min}}}{\text{PWM}_{\text{MAX0}} - \text{PWM}_{\text{MIN}}}
\]

\[
b = V_{\text{max}} - m \times \text{PWM}_{\text{MAX0}}
\]

\[
voltage = m \times \text{value} + b
\]
4.3 Serial Communication Drivers

The interface to the Rabbit serial library, **RSERIAL.LIB**, is designed to provide users with a set of functions that send and receive entire blocks of data without yielding to other tasks, a set of single user cofunctions that send and receive data but yield to other tasks, and a set of circular buffer functions.

The naming convention is **serXfn**:  
- **ser** - serial  
- **X** - the port being used: A, B, C, or D  
- **fn** - the function being implemented

For example, **serBgetc()** is the serial port B function **getc()**, which returns a character.

The Rabbit serial functions are listed in the following groups.

- **Open and Close Functions**
- **Non-Cofunction Blocking Input Functions**
- **Non-Cofunction Blocking Output Functions**
- **Single-User Cofunction Input Functions**
- **Single-User Cofunction Output Functions**
- **Circular Buffer Functions**

4.3.1 Open and Close Functions

The open and close functions enable and disable serial communication over the specified port.

```c
int serXopen (long baud);
```

Currently only 8N1 transmission (8 data bits, no parity, 1 stop bit) is supported. The **open** function sets up the interrupt service routine vector.

**Parameters**

- **baud**—desired baud rate in bits per second

**Return Value**

- 1—The baud rate set on the Rabbit is the same as the input baud rate.  
- 0—The baud rate set on the rabbit does not match the input baud rate.

```c
int serXclose ();
```

Disables the serial port interrupt service routine.

**Parameters**

- None.

**Return Value**

- 1
4.3.2 Non-Cofunction Blocking Input Functions

These are simple functions that do not use Dynamic C costatements. If no input data are available when called, they return immediately with appropriate status information in their return value. Once they begin to receive characters, they do not yield to other tasks until they complete their operation or until a character-to-character timeout period elapses.

```c
int serXgetc ( );
```

Gets a single character. Always returns immediately, either with the next available input byte, or with –1 if none is available.

**Parameters**

None

**Return Value**

An integer with return character in the low byte. No character is represented by a return of –1.

```c
int serXread (void *data, int length, unsigned long tmout);
```

Reads a block of characters. Returns the number of bytes read from an input serial stream. The stream is considered to be ended when all length bytes have been read or when the timeout period elapses waiting for data to appear in the input buffer.

**Parameters**

- `data`—Destination data structure. The user must ensure data is allocated for at least length bytes.
- `length`—The number of bytes to read.
- `tmout`—The number of milliseconds to wait for receipt of each byte before timing out.

**Return Value**

The number of bytes read into data until timed out or until all length bytes have been read.
4.3.3 Non-Cofunction Blocking Output Functions

These are simple functions that do not use Dynamic C costatements. They immediately begin to perform their task, not yielding to other tasks until all characters have been written.

```c
int serXputc (char c);
```

Writes a character to the serial port.

Parameters

- `c`—Character to write

Return Value

1 for success, 0 if the character could not be written to the port.

```c
int serXputs (char *s);
```

Calls `serXwrite (s, strlen (s))`.

Parameters

- `s`—Null-terminated character string source to write to the serial port.

Return Value

The number of characters written.

```c
int serXwrite (void *data, int length);
```

Writes a block of `length` bytes to the serial port.

Parameters

- `data`—Destination data structure. The user must ensure data is allocated for at least `length` bytes.
- `length`—The number of bytes to read.

Return Value

The number of bytes written to the serial port.
4.3.4 Single-User Cofunction Input Functions

These are Dynamic C cofunctions. If the input buffer they use is locked or becomes full during the course of their operation, they yield to other tasks, but do not return to execute the next statement within their own costatement block until they have completed their operation.

```c
scofunc int cof_serXgetc ( );
```

Reads a single character from the serial port, yielding when not successful, and only returning when a character is successfully read.

**Parameters**

None

**Return Value**

An integer with the character read in the low byte.

```c
scofunc int cof_serXgets(char *s, int length, unsigned long tmout);
```

Reads a null-terminated string, completes its execution when a carriage return is read, `length` number of characters are read, or the character to character timeout period elapses after the first character is read. It yields to other tasks while the input buffer is locked or becomes empty during its execution, and only returns control to the following statement in its own costatement block when it completes.

**Parameters**

- `data`—Destination data structure. The user must ensure data is allocated for at least length bytes.
- `length`—The number of bytes to read.
- `tmout`—The number of milliseconds to wait for each character after the first character is read.

**Return Value**

1—CR or `length` bytes read into `s`.
0—Function times out before reading CR or `length` bytes.

```c
scofunc int cof_serXread(void *data, int length, unsigned long tmout);
```

Reads a block of characters, completes its execution when `length` number of characters are read, or the character-to-character timeout period elapses after the first character is read. It yields to other tasks while the input buffer is locked or becomes empty during its execution and only returns control to the following statement in its own costatement block when it completes.

**Parameters**

- `data`—Destination data structure. The user must ensure data is allocated for at least length bytes.
- `length`—The number of bytes to read.
- `tmout`—The number of milliseconds to wait for each character after the first.

**Return Value**

The number of bytes read.
4.3.5 Single-User Cofunction Output Functions

These are Dynamic C cofunctions. If the output buffer they use is locked or becomes empty during the course of their operation, they yield to other tasks, but do not return to execute the next statement within their own costatement block until they have completed their operation.

```c
scofunc void cof_serXputc (char c);
```

Writes a single character to the serial port, yielding to other tasks when unsuccessful, and returning only when the character is successfully written.

**Parameters**
- `c`—Character to write to the serial port.

**Return Value**
None

```c
scofunc void cof_serXputs (char *s);
```

Writes a null-terminated character string to the serial port, yielding to other tasks when unsuccessful or whenever the buffer is full, returning only when the string is successfully written.

**Parameters**
- `s`—Null-terminated character string written to the serial port.

**Return Value**
None

```c
scofunc void cof_serXwrite (void *data, int length);
```

Writes a block of characters to the serial port, yielding to other tasks when unsuccessful or whenever the buffer is full, returning only when all the data is successfully written.

**Parameters**
- `data`—Source data structure to write to the serial port.
- `length`—Number of characters in data to write.

**Return Value**
None
4.3.6 Circular Buffer Functions

These functions act on or report status of the circular transmit/receive buffers.

Macro definitions are used to establish the buffer sizes:

\[ x\text{INBUFSIZE} \] — read buffer size, where \( x \) is A, B, C, or D

\[ x\text{OUTBUFSIZE} \] — write buffer size where \( x \) is A, B, C, or D

The user must define each buffer size for each port being used to be a power of 2 minus 1 with a macro. The size of \( 2^n - 1 \) enables masking for fast rollover calculations. If no value or an illegal value is defined, a default size of 31 will be used and a compiler warning will be given. When using cofunctions, smaller buffer sizes can yield more frequently to other tasks, but have the risk of a large input data stream overrunning the buffer and losing data if the other task executes for too long relative to the baud rate.

\[ \text{int serXpeek ( );} \]

Returns the first character in the receive buffer, if any are available, without removing it from the buffer.

**Parameters**

None

**Return Value**

An integer with return character in the low byte. No character is represented by a return of \(-1\).

\[ \text{void serXrdFlush ( );} \]

Flushes the serial port receive buffer.

**Parameters**

None

**Return Value**

None

\[ \text{void serXwrFlush ( );} \]

Flushes the serial port transmit buffer.

**Parameters**

None

**Return Value**

None
```c
int serXrdFree ( );
```
Calculates the free space in the serial port receive buffer.

**Parameters**

None

**Return Value**

The number of characters the serial port receive buffer can accept before becoming full.

```c
int serXwrFree ( );
```
Calculates the free space in the serial port transmit buffer.

**Parameters**

None

**Return Value**

The number of characters the serial port transmit buffer can accept before becoming full.

```c
int serXrdUsed ( );
```
Calculates the number of characters ready to read from the serial port receive buffer.

**Parameters**

None

**Return Value**

The number of characters currently in the serial port receive buffer.
## Appendix A. Specifications

Table A-1 lists the electrical, mechanical, and environmental specifications for the Jack-rabbit board.

### Table A-1. Jackrabbit Board Specifications

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Specification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Board Size</td>
<td>3.50&quot; × 2.50&quot; × 0.94&quot; (89 mm × 64 mm × 24 mm)</td>
</tr>
<tr>
<td>Humidity</td>
<td>5% to 95%, noncondensing</td>
</tr>
<tr>
<td>Input Voltage and Current</td>
<td>7.5 V to 25 V DC, 100 mA typical, 150 mA maximum, linear regulator</td>
</tr>
<tr>
<td>Configurable I/O</td>
<td>44—28 independent 16 shared with onboard peripherals 3 additional shared with programming/debugging</td>
</tr>
<tr>
<td>Analog Inputs</td>
<td>One low-grade A/D input—input range 0 V to 3 V, 10-bit resolution, 8-bit accuracy, average acquisition time 150 ms (165 ms maximum) with 14.7 MHz clock</td>
</tr>
<tr>
<td>Analog Outputs</td>
<td>Two filtered and buffered PWM outputs</td>
</tr>
<tr>
<td>Digital Outputs</td>
<td>Four high-current, high-voltage outputs—3 sink up to 200 mA and 35 V standoff</td>
</tr>
<tr>
<td>Clock</td>
<td>14.74 MHz</td>
</tr>
<tr>
<td>SRAM</td>
<td>128K (supports 32K–512K)</td>
</tr>
<tr>
<td>Flash EPROM</td>
<td>128K (supports 32K–512K)</td>
</tr>
<tr>
<td>Timers</td>
<td>Five 8-bit timers, one 10-bit timer with two match registers, five timers are cascadable</td>
</tr>
<tr>
<td>Serial Ports</td>
<td>Four serial ports—two RS-232 or one RS-232 (with CTS/RTS) and one RS-485 One 5 V CMOS-compatible line Two serial ports can be clocked</td>
</tr>
<tr>
<td>Serial Rate</td>
<td>Up to 1.84 MHz</td>
</tr>
<tr>
<td>Watchdog/Supervisor</td>
<td>Yes</td>
</tr>
<tr>
<td>Time/Date Clock</td>
<td>Yes</td>
</tr>
<tr>
<td>Backup Battery</td>
<td>Yes, 3 V lithium coin type, 950 mA-h, includes external battery holder</td>
</tr>
</tbody>
</table>
Schematics
<table>
<thead>
<tr>
<th>CIRCUIT</th>
<th>PART</th>
<th>MODEL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>BL1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MAIN</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>J2</td>
<td>100 ohm</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td>R1</td>
<td>4.3M ohm</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td>R12</td>
<td>NOT INSTALLED</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td>R13</td>
<td>1.3k ohm</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>BACKUP BATTERY</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q1</td>
<td>5088 nF</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td>R14</td>
<td>220k ohm</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td>R15</td>
<td>2M ohm</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>REGULATOR</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>D88</td>
<td>NOT INSTALLED</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td>D81</td>
<td>NOT INSTALLED</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td>D82</td>
<td>NOT INSTALLED</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td>R80</td>
<td>ZERO ohm</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td>R81</td>
<td>ZERO ohm</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td>R82</td>
<td>ZERO ohm</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>CHIP SELECT CONTROL</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C23</td>
<td>220pF</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td>D28</td>
<td>914 nF</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td>Q20</td>
<td>2000 nF</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td>Q21</td>
<td>FV3802</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td>Q22</td>
<td>3904 nF</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td>R37</td>
<td>18k ohm</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td>R39</td>
<td>10k ohm</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td>R40</td>
<td>30k ohm</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td>R41</td>
<td>47k ohm</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td>R42</td>
<td>100k ohm</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>W/OUT BATTERY BACKUP</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R38</td>
<td>NOT INSTALLED</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>POWER TO CRT SWITCH</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R39</td>
<td>NOT INSTALLED</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>POWER SUPPLY</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>C2</td>
<td>NOT INSTALLED</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td>U1</td>
<td>1uF</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td>U2</td>
<td>LM2575-5.0</td>
<td>NOT INSTALLED</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>CIRCUIT</th>
<th>PART</th>
<th>MODEL</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>BL1000</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>MAIN</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U5</td>
<td>128K SRAM</td>
<td>128K SRAM</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>SRAM SELECT</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JP1</td>
<td>ZERO ohm</td>
<td>ACROSS PINS 1-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FLASH</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>U22</td>
<td>256K FLASH</td>
<td>128K FLASH</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>FLASH SELECT</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>JP2</td>
<td>ZERO ohm</td>
<td>ACROSS PINS 1-2</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>HIGH POWER DRIVERS</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Q25</td>
<td>FM6169</td>
<td>4401 pnp</td>
</tr>
<tr>
<td>Q26</td>
<td>FM6169</td>
<td>4401 pnp</td>
</tr>
<tr>
<td>Q27</td>
<td>FM6169</td>
<td>4401 pnp</td>
</tr>
<tr>
<td>Q28</td>
<td>FM6169</td>
<td>3906 pnp</td>
</tr>
<tr>
<td>Q29</td>
<td>FM6169</td>
<td>4401 pnp</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>PROCESSOR</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y3</td>
<td>PROCESSOR</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td>Y4</td>
<td>RTC CRYSTAL</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>PROCESSOR CRYSTAL</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R4</td>
<td>NOT INSTALLED</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td>Y1</td>
<td>160 ohms</td>
<td>NOT INSTALLED</td>
</tr>
<tr>
<td>Y2</td>
<td>29.49MHz Crystal</td>
<td>NOT INSTALLED</td>
</tr>
</tbody>
</table>
THIS DOCUMENT: APPEND THE FOLLOWING
DOCUMENTS WHEN CHANGING
ECO APPROVAL CONTROL
DATE
REVISION HISTORY
ENGINEER PROJECT DATE
REV DESCRIPTION
REVISION APPROVAL
NONE DATE
B
DRAWING CONTENT:
APPROVALS: INITIAL RELEASE
SIGNATURES
2900 SPAFFORD ST.
DAVIS, CA 95616
530 - 757 - 4616
THIS DOCUMENT: APPEND THE FOLLOWING
ECO APPROVAL CONTROL
DATE
REVISION HISTORY
ENGINEER PROJECT DATE
REV DESCRIPTION
REVISION APPROVAL
NONE DATE
B
DRAWING CONTENT:
APPROVALS: INITIAL RELEASE
SIGNATURES
2900 SPAFFORD ST.
DAVIS, CA 95616
530 - 757 - 4616

IMPORTANT:
CUT TRACES BETWEEN K AND
VCC BEFORE CONNECTING K
TO ANOTHER SOURCE!

AFTER RUNNING DEMO PROGRAMS,
CAN CUT TRACES IN JP1 TO DISCONNECT
DEVICES FROM JACKRABBIT
PINS

REVISION HISTORY
REV ECO DESCRIPTION
C E18812 CHANGE VALUE OF R1 FROM 100 OHMS TO 3.3 OHMS
D E18904 CORRECT & CLARIFY INFORMATION IN SCHEMATIC
E E1105 ADD WARNING ABOUT CUTTING TRACES BETWEEN K & VCC

JP2 JUMPER SETTINGS
FUNCTION SETTING
Buzzer 1-2
K to 45V 3-5
K to GND 5-6
18K POT OR
7-8
1K POT OR
9-10

POWER
DCM VCC 4,6,7,9,10
1VOUT

12V COIL
RESISTOR

RESISTOR