Welcome to E-XFL.COM

Understanding Embedded - Microcontroller, Microprocessor, FPGA Modules

Embedded - Microcontroller, Microprocessor, and FPGA Modules are fundamental components in modern electronic systems, offering a wide range of functionalities and capabilities. Microcontrollers are compact integrated circuits designed to execute specific control tasks within an embedded system. They typically include a processor, memory, and input/output peripherals on a single chip. Microprocessors, on the other hand, are more powerful processing units used in complex computing tasks, often requiring external memory and peripherals. FPGAs (Field Programmable Gate Arrays) are highly flexible devices that can be configured by the user to perform specific logic functions, making them invaluable in applications requiring customization and adaptability.

Applications of Embedded - Microcontroller,

Details	
Product Status	Obsolete
Module/Board Type	MPU Core
Core Processor	Rabbit 2000
Co-Processor	-
Speed	22.1MHz
Flash Size	256KB
RAM Size	128KB
Connector Type	IDC Header 2x20
Size / Dimension	2.6" x 3" (66mm x 76mm)
Operating Temperature	-40°C ~ 85°C
Purchase URL	https://www.e-xfl.com/product-detail/digi-international/101-0497

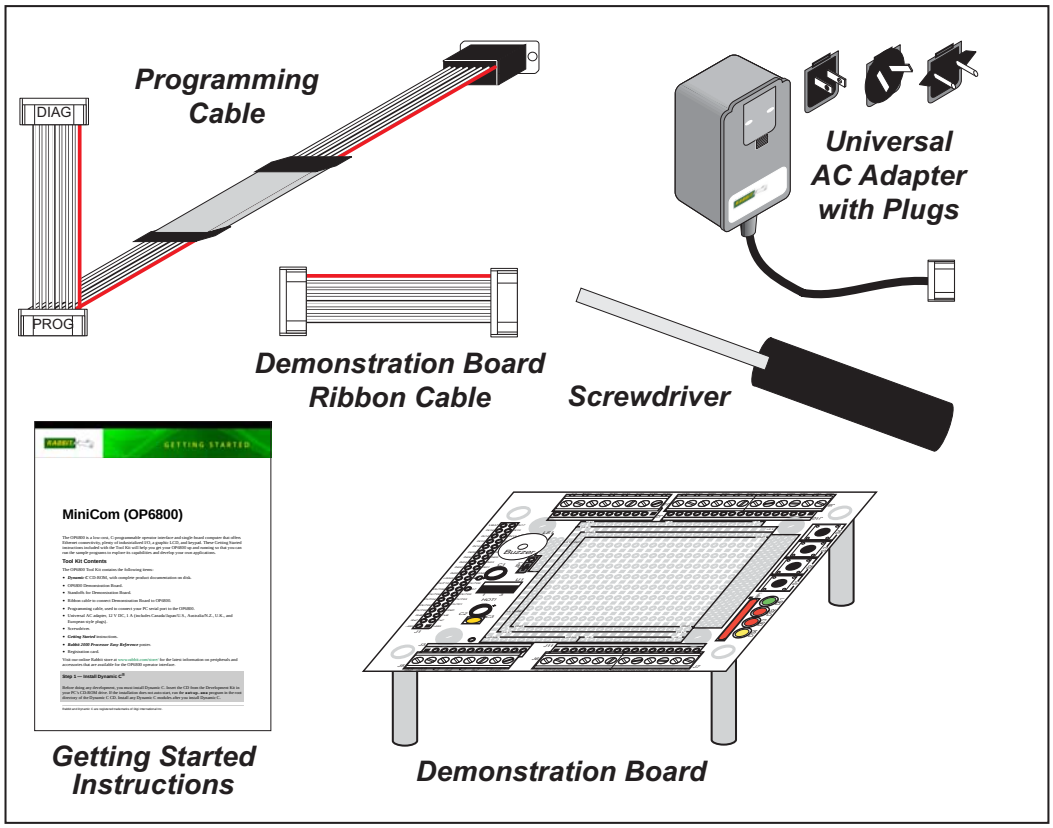

Figure 1. OP6800 Tool Kit

1.3.2 Software

The OP6800 is programmed using version 7.06 or later of Rabbit's Dynamic C. A compatible version is included on the Tool Kit CD-ROM. Library functions provide an easy-to-use interface for the OP6800. Software drivers for the display and keypad, TCP/IP, I/O, and serial communication are included with Dynamic C.

Dynamic C v. 9.60 includes the popular µC/OS-II real-time operating system, point-to-point protocol (PPP), FAT file system, RabbitWeb, and other select libraries that were previously sold as individual Dynamic C modules.

Rabbit also offers for purchase the Rabbit Embedded Security Pack featuring the Secure Sockets Layer (SSL) and a specific Advanced Encryption Standard (AES) library. In addition to the Web-based technical support included at no extra charge, a one-year telephone-based technical support subscription is also available for purchase. Visit our Web site at www.rabbit.com for further information and complete documentation, or contact your Rabbit sales representative or authorized distributor.

2.7 PONG.C

You are now ready to test your programming connections by running a sample program.

Find the file **PONG.C**, which is in the Dynamic C **SAMPLES** folder. To run the program, open it with the **File** menu (if it is not still open), compile it using the **Compile** menu, and then run it by selecting **Run** in the **Run** menu. The **STDIO** window will open and will display a small square bouncing around in a box.

This program shows that the CPU is working. The sample program described in Section 5.2.3, "Run the PINGME.C Demo," tests the TCP/IP portion of the board (if you have the OP6800 model—the OP6810 does not have an Ethernet capability).

2.8 Where Do I Go From Here?

NOTE: If you purchased your OP6800 through a distributor or Rabbit partner, contact the distributor or partner first for technical support.

If there are any problems at this point:

- Use the Dynamic C **Help** menu to get further assistance with Dynamic C.
- Check the Rabbit Technical Bulletin Board and forums at www.rabbit.com/support/bb/ and at www.rabbit.com/forums/.
- Use the Technical Support e-mail form at www.rabbit.com/support/.

If the sample program ran fine, you are now ready to go on to explore other OP6800 features and develop your own applications.

The following sample programs illustrate the features and operation of the OP6800.

OP6800 (SAMPLES\LCD_KEYPAD\122x32_1x7)	Demonstration Board (SAMPLES\OP6800\DEMO_BD)
KEYBASIC.C KEYMENU.C SCROLLING.C TEXT.C	KEYPAD.C SWITCHES.C

These sample programs can be used as templates for applications you may wish to develop.

Chapter 3, "Subsystems," provides a description of the OP6800's features, Chapter 4, "Software," describes the Dynamic C software libraries and describes the sample programs, and Chapter 5, "Using the TCP/IP Features," explains the TCP/IP features and describes some sample programs.

3. SUBSYSTEMS

Chapter 3 describes the principal subsystems for the OP6800.

- Digital I/O
- Serial Communication
- Memory

Figure 7 shows these Rabbit-based subsystems designed into the OP6800.

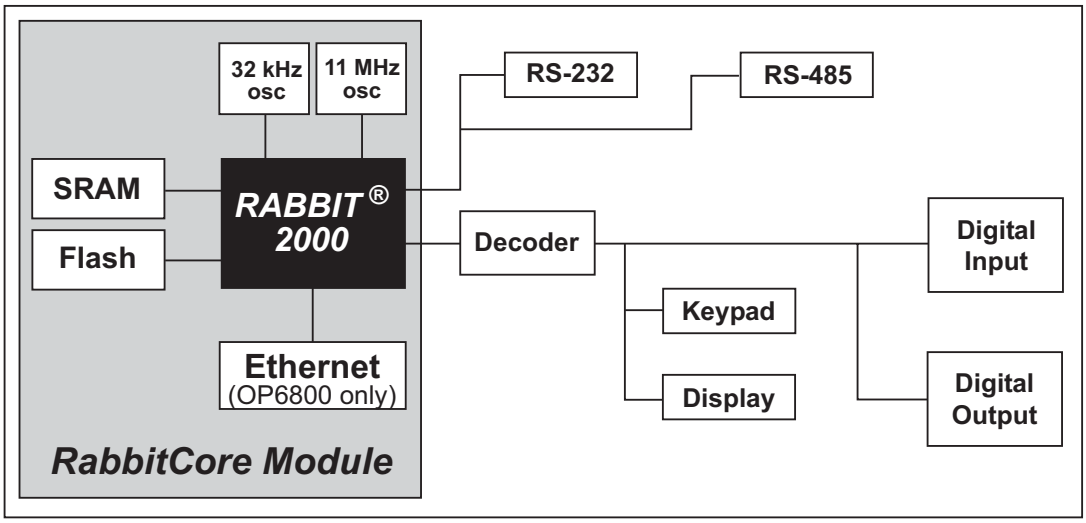

Figure 7. OP6800 Subsystems

3.3.4 **Ethernet Port** (OP6800 models only)

Figure 13 shows the pinout for the Ethernet port (J2 on the OP6800 module). Note that there are two standards for numbering the pins on this connector—the convention used here, and numbering in reverse to that shown. Regardless of the numbering convention followed, the pin positions relative to the spring tab position (located at the bottom of the RJ-45 jack in Figure 13) are always absolute, and the RJ-45 connector will work properly with off-the-shelf Ethernet cables.

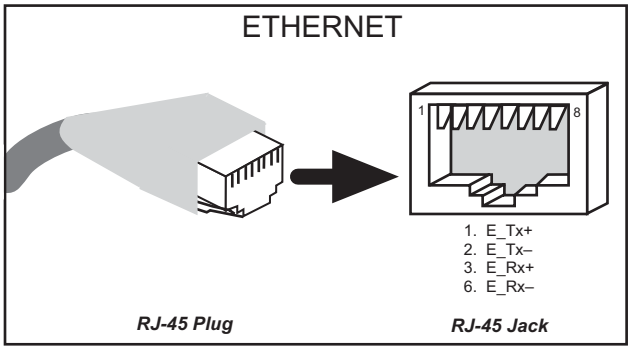

Figure 13. RJ-45 Ethernet Port Pinout

RJ-45 pinouts are sometimes numbered opposite to the way shown in Figure 13.

Two LEDs are placed next to the RJ-45 Ethernet jack, one to indicate an Ethernet link (**LNK**) and one to indicate Ethernet activity (**ACT**).

The transformer/connector assembly ground is connected to the BL2100 module printed circuit board digital ground via a 0 Ω resistor "jumper," R29, as shown in Figure 14.

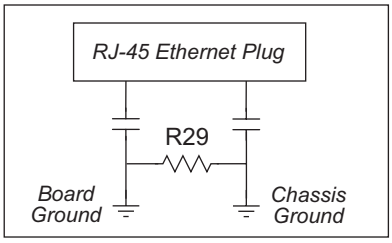

Figure 14. Isolation Resistor R29

The factory default is for the 0 Ω resistor "jumper" at R29 to be installed. In high-noise environments, remove R29 and ground the transformer/connector assembly directly through the chassis ground. This will be especially helpful to minimize ESD and/or EMI problems.

3.4 Programming Cable

The programming cable is used to connect the programming port of the OP6800 to a PC serial COM port. The programming cable converts the RS-232 voltage levels used by the PC serial port to the voltage levels used by the Rabbit 2000.

When the **PROG** connector on the programming cable is connected to the OP6800 programming port, programs can be downloaded and debugged over the serial interface between the PC and the Rabbit 2000.

The **DIAG** connector of the programming cable may be used on header J1 of the OP6800 RabbitCore module with the OP6800 operating in the Run Mode. This allows the programming port to be used as a regular serial port.

3.4.1 Changing Between Program Mode and Run Mode

The OP6800 is automatically in Program Mode when the **PROG** connector on the programming cable is attached, and is automatically in Run Mode when no programming cable is attached. When the Rabbit 2000 is reset, the operating mode is determined by the status of the SMODE pins. When the programming cable's **PROG** connector is attached, the SMODE pins are pulled high, placing the Rabbit 2000 in the Program Mode. When the programming cable's **PROG** connector is not attached, the SMODE pins are pulled low, causing the Rabbit 2000 to operate in the Run Mode.

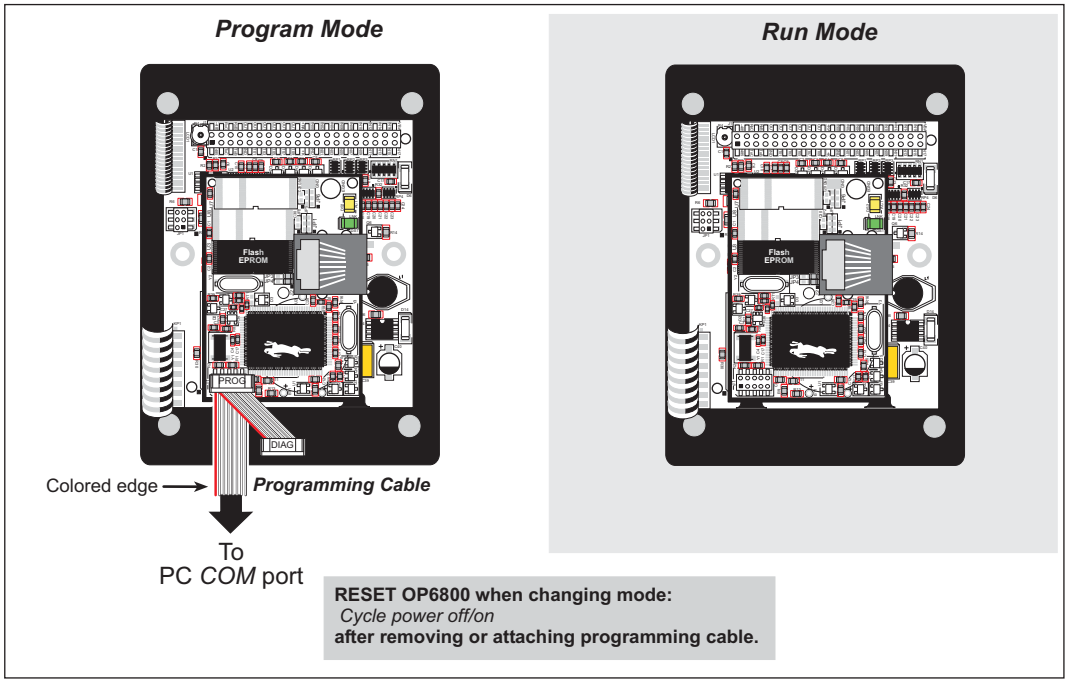

Figure 15. OP6800 Program Mode and Run Mode Set-Up

A program "runs" in either mode, but can only be downloaded and debugged when the OP6800 is in the Program Mode.

Refer to the ***Rabbit 2000 Microprocessor User's Manual*** for more information on the programming port and the programming cable.

3.5 Other Hardware

3.5.1 Clock Doubler

The OP6800 takes advantage of the Rabbit 2000 microprocessor's internal clock doubler. A built-in clock doubler allows half-frequency crystals to be used to reduce radiated emissions. The 22.1 MHz frequency is generated using an 11.0592 MHz crystal. The clock doubler is disabled automatically in the BIOS for crystals with a frequency above 12.9 MHz.

The clock doubler may be disabled if 22.1 MHz clock speeds are not required. Disabling the Rabbit 2000 microprocessor's internal clock doubler will reduce power consumption and further reduce radiated emissions. The clock doubler is disabled with a simple global macro as shown below.

1. Select the "Defines" tab from the Dynamic C **Options > Project Options** menu.
2. Add the line **CLOCK_DOUBLED=0** to always disable the clock doubler.

   The clock doubler is enabled by default, and usually no entry is needed. If you need to specify that the clock doubler is always enabled, add the line **CLOCK_DOUBLED=1** to always enable the clock doubler. The clock speed will be doubled as long as the crystal frequency is less than or equal to 26.7264 MHz.
3. Click **OK** to save the macro. The clock doubler will now remain off whenever you are in the project file where you defined the macro.

4.2 Font and Bitmap Converter

A ***Font and Bitmap Converter*** tool is available to convert Windows fonts and monochrome bitmaps to a library file format compatible with Rabbit's Dynamic C applications and graphical displays. Non-Roman characters can also be converted by applying the monochrome bitmap converter to their bitmaps.

Start the ***Font and Bitmap Converter*** tool by double-clicking on the **fbmcnvtr.exe** file in the Dynamic C directory. You then select and convert existing fonts or bitmaps. Complete instructions are available via the **Help** menu that is in the ***Font and Bitmap Converter*** tool.

Once you are done, the converted file is displayed in the editing window. Editing may be done, but should not be necessary. Save the file as ***libraryfilename*.lib**, where ***libraryfilename*** is a file name of your choice.

Add the library file(s) to applications with the statement **#use *libraryfilename*.lib,** or by cutting and pasting from the library file(s) you created into the application program.

> **TIP:** If you used the **#use *libraryfilename*.lib** statement, remember to enter ***libraryfilename*.lib** into **lib.dir**, which is located in your Dynamic C directory.

You are now ready to add the font or bitmap to your application using the **glXFontInit** or the **glXPutBitmap** function calls.

4.3 Sample Programs

Sample programs are provided in the Dynamic C **Samples** folder. The sample program **PONG.C** demonstrates the output to the **STDIO** window.

The various directories in the **Samples** folder contain specific sample programs that illustrate the use of the corresponding Dynamic C libraries.

The **OP6800** folder provides sample programs specific to the OP6800. Each sample program has comments that describe the purpose and function of the program. Follow the instructions at the beginning of the sample program.

To run a sample program, open it with the **File** menu (if it is not still open), then compile and run it by pressing **F9**. The OP6800 must be in **Program** mode (see Section 3.4) and must be connected to a PC using the programming cable as described in Section 2.1.

More complete information on Dynamic C is provided in the ***Dynamic C User's Manual***. TCP/IP specific functions are described in the ***Dynamic C TCP/IP User's Manual***. Information on using the TCP/IP features and sample programs is provided in Section 5, "Using the TCP/IP Features."

4.3.1 Board ID

The following sample program can be found in the **SAMPLES\OP6800** subdirectory.

- **BOARD_ID.C**—Detects the type of single-board computer and displays the information in the **STDIO** window. For the OP6800, the **STDIO** window should show **OP6800**.

4.3.2 Demonstration Board

The following sample programs are found in the **DEMO_BD** subdirectory in **SAMPLES\OP6800**.

- **BUZZER.C**—Demonstrates the use of the buzzer on the Demonstration Board. Remember to set the jumper across pins 1–2 of header JP1 on the Demonstration Board (see Figure C-4) to enable the buzzer on. When you finish with **BUZZER.C**, it is recommended that you reconnect the jumper across pins 2–3 of header JP1 on the Demonstration Board to disable the buzzer.
- **KEYPAD.C**—Flashes the LED above a keypad button when the corresponding keypad button is pressed. The corresponding LED on the Demonstration Board will also flash if a keypad button in the top row of the keypad is pressed. A message is also displayed on the LCD.
- **SWITCHES.C**—Flashes the LED on the Demonstration Board and the OP6800 when the corresponding pushbutton switch on the Demonstration Board is pressed. A message is also displayed on the LCD.

APPENDIX A. SPECIFICATIONS

Appendix A provides the specifications for the OP6800 and describes the conformal coating.

A.1 Electrical and Mechanical Specifications

Figure A-1 shows the mechanical dimensions for the OP6800.

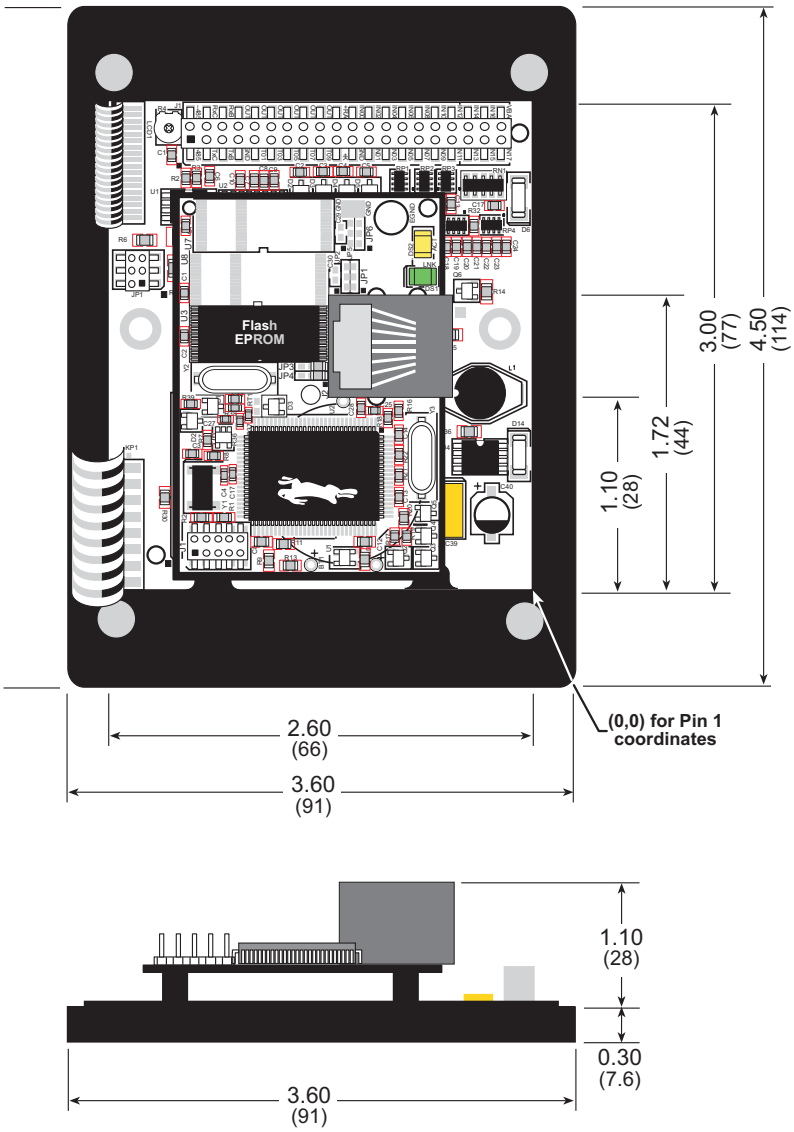

Figure A-1. OP6800 Dimensions

NOTE: All measurements are in inches followed by millimeters enclosed in parentheses.

Table A-1 provides the pin 1 locations for the OP6800 headers as viewed in Figure A-1.

Table A-1. OP6800 Header J1 Pin 1 Locations

Header	Pin 1 (x,y) Coordinates (inches)
J1	(-2.101, 2.720)

It is recommended that you allow for an “exclusion zone” of 0.25" (6 mm) around the OP6800 in all directions when the OP6800 is incorporated into an assembly that includes other components. This “exclusion zone” that you keep free of other components and boards will allow for sufficient air flow, and will help to minimize any electrical or EMI interference between adjacent boards. Figure A-2 shows this “exclusion zone.”

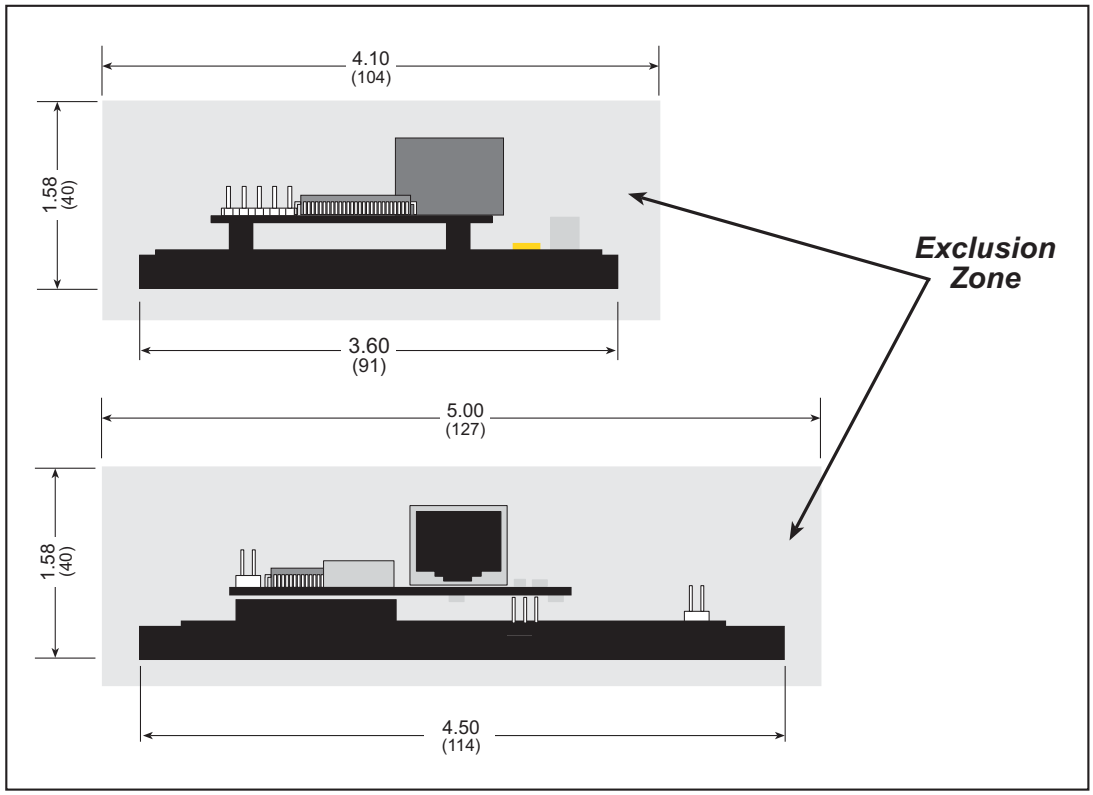

Figure A-2. OP6800 “Exclusion Zone”

B.2 Batteries and External Battery Connections

The SRAM and the real-time clock have provision for battery backup. Power to the SRAM and the real-time clock (VRAM) is provided by two different sources, depending on whether the main part of the OP6800 is powered or not. When the OP6800 is powered normally, and Vcc is within operating limits, the SRAM and the real-time clock are powered from Vcc. If power to the board is lost or falls below 4.63 V, the VRAM and real-time clock power must come from a backup battery in your system which you would connect to pin 40 of header J1 on the OP6800 via the ribbon cable. The backup battery should be able to supply 2.85 V–3.15 V at 10 µA.

The reset generator circuit controls the source of power by way of its **/RESET** output signal.

B.2.1 Battery-Backup Circuit

Figure B-1 shows the battery-backup circuit located on the OP6800 module.

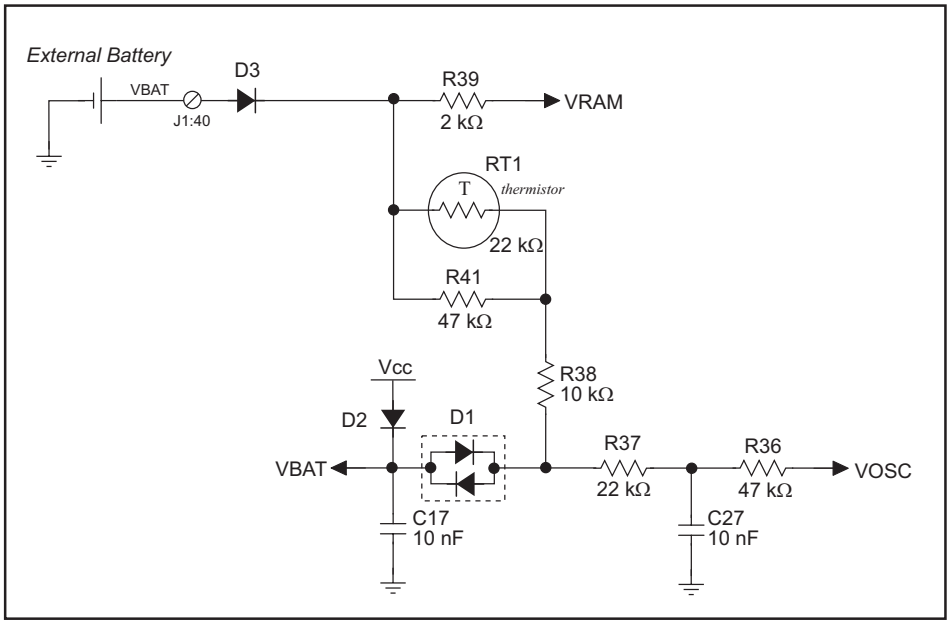

Figure B-1. OP6800 Backup Battery Circuit

The battery-backup circuit serves three purposes:

- It reduces the battery voltage to the SRAM and to the real-time clock, thereby limiting the current consumed by the real-time clock and lengthening the battery life.
- It ensures that current can flow only ***out*** of the battery to prevent charging the battery.
- A voltage, VOSC, is supplied to U6, which keeps the 32.768 kHz oscillator working when the voltage begins to drop.

VRAM and Vcc are nearly equal (<100 mV, typically 10 mV) when power is supplied to the OP6800.

void ser485Rx(void);

Resets pin 3 (DE) low to disable the RS-485 transmitter. Remember to call **serMode** before calling **ser485Rx**.

SEE ALSO

serMode, ser485Tx, serCflowcontrolOn, serCflowcontrolOff

void glSetContrast(unsigned level);

Sets display contrast.

NOTE: This function is not used with the OP6800 since the support circuits are not available on the LCD/keypad module used with the OP6800.

void glFillScreen(int pattern);

Fills the LCD display screen with a pattern.

PARAMETER

The screen will be set to all black if **pattern** is 0xFF, all white if **pattern** is 0x00, and vertical stripes for any other pattern.

RETURN VALUE

None.

SEE ALSO

glBlock, glBlankScreen, glPlotPolygon, glPlotCircle

void glBlankScreen(void);

Blanks the LCD display screen (sets LCD display screen to the background color).

RETURN VALUE

None.

SEE ALSO

glFillScreen, glBlock, glPlotPolygon, glPlotCircle

void glFillRegion(int left, int top, int width, int height, char pattern);

Fills a rectangular block in the LCD buffer with the pattern specified. Any portion of the block that is outside the LCD display area will be clipped.

PARAMETERS

left is the *x* coordinate of the top left corner of the block.

top is the *y* coordinate of the top left corner of the block.

width is the width of the block.

height is the height of the block.

pattern is the bit pattern to display (all black if **pattern** is 0xFF, all white if **pattern** is 0x00, and vertical stripes for any other pattern).

RETURN VALUE

None.

SEE ALSO

glFillScreen, glBlankScreen, glBlock, glBlankRegion

void glRight1(int left, int top, int cols, int rows);

Scrolls byte-aligned window right one pixel, left column is filled by current pixel type (color).

PARAMETERS

left is the top left corner of bitmap, must be evenly divisible by 8, otherwise truncates.

top is the top left corner of the bitmap.

cols is the number of columns in the window, must be evenly divisible by 8, otherwise truncates.

rows is the number of rows in the window.

RETURN VALUE

None.

SEE ALSO

glHScroll, glLeft1

void glUp1(int left, int top, int cols, int rows);

Scrolls byte-aligned window up one pixel, bottom column is filled by current pixel type (color).

PARAMETERS

left is the top left corner of bitmap, must be evenly divisible by 8, otherwise truncates.

top is the top left corner of the bitmap.

cols is the number of columns in the window, must be evenly divisible by 8, otherwise truncates.

rows is the number of rows in the window.

RETURN VALUE

None.

SEE ALSO

glVScroll, glDown1

void glDown1(int left, int top, int cols, int rows);

Scrolls byte-aligned window down one pixel, top column is filled by current pixel type (color).

PARAMETERS

left is the top left corner of bitmap, must be evenly divisible by 8, otherwise truncates.

top is the top left corner of the bitmap.

cols is the number of columns in the window, must be evenly divisible by 8, otherwise truncates.

rows is the number of rows in the window.

RETURN VALUE

None.

SEE ALSO

glVScroll, glUp1

D.5.1 Keypad

The functions used to control the keypad are in the **KEYPAD7.LIB** library located in the Dynamic C **LIB\KEYPADS** library folder.

```
void keyInit(void);
```

Initializes keypad process

RETURN VALUE

None.

SEE ALSO

brdInit

```
void keyConfig(char cRaw, char cPress,
   char cRelease, char cCntHold, char cSpdLo,
   char cCntLo, char cSpdHi);
```

Assigns each key with key press and release codes, and hold and repeat ticks for auto repeat and debouncing.

PARAMETERS

cRaw is a raw key code index.

1 × 7 keypad matrix with raw key code index assignments (in brackets):

[0]		[1]		[2]		[3]
	[4]		[5]		[6]	

User Keypad Interface

cPress is a key press code

An 8-bit value is returned when a key is pressed.
0 = Unused.

See **keypadDef()** for default press codes.

cRelease is a key release code.

An 8-bit value is returned when a key is pressed.
0 = Unused.

cCntHold is a hold tick, which is approximately one debounce period or 5 µs.

How long to hold before repeating.
0 = No Repeat.

cSpdLo is a low-speed repeat tick, which is approximately one debounce period or 5 µs.

How many times to repeat.
0 = None.

cCntLo is a low-speed hold tick, which is approximately one debounce period or 5 µs.

How long to hold before going to high-speed repeat.
0 = Slow Only.

cSpdHi is a high-speed repeat tick, which is approximately one debounce period or 5 µs.

How many times to repeat after low speed repeat.
0 = None.

RETURN VALUE

None.

SEE ALSO

keyProcess, keyGet, keypadDef

void keyProcess(void);

Scans and processes keypad data for key assignment, debouncing, press and release, and repeat.

NOTE: This function is also able to process an 8 × 8 matrix keypad.

RETURN VALUE

None

SEE ALSO

keyConfig, keyGet, keypadDef

char keyGet(void);

Get next keypress.

RETURN VALUE

The next keypress, or 0 if none

SEE ALSO

keyConfig, keyProcess, keypadDef

int keyUnget(char cKey);

Pushes the value of **cKey** to the top of the input queue, which is 16 bytes deep.

PARAMETER

cKey

RETURN VALUE

None.

SEE ALSO

keyGet