PRODUCT MANUAL

RabbitFLEX™ BL300F
User's Manual

019-0155 • 070720-D

The latest revision of this manual is available on the Rabbit Web site, www.rabbit.com, for free, unregistered download.

RabbitFLEX BL300F User's Manual

Part Number 019-0155 • 070720–D • Printed in U.S.A.

Trademarks

Rabbit and Dynamic C® are registered trademarks of Rabbit.

Windows® is a registered trademark of Microsoft Corporation

Table of Contents

1. RabbitFLEX BL300F Introduction

We all know you cannot fit a square peg into a round hole. But sometimes that is exactly what you are asked to do when forced to choose an off-the-shelf single-board computer (SBC) for your embedded system application. In the past, price and time constraints could make a custom-designed board unattainable.

That has all changed with the introduction of the RabbitFLEX BL300F.

Now, you can design your own perfect-fit controller board using Rabbit's new online design and order system, the RabbitFLEX Configurator. Because the controller board is customized during surface mount assembly, a custom solution can be obtained more quickly and at a lower price than previously possible.

1.1 RabbitFLEX BL300F Options

The items in the following list may be specified to customize your board design.

- **PowerCore Module**
  PowerCore 3800 (Ethernet, 512 K Flash, 1 MB SRAM, 1 MB Serial Flash)
  PowerCore 3810 (512 K Flash, 256 K SRAM)
- **Serial Communication Channels**
  RS-232 (3- or 5-wire)
  RS-485 (2-wire) or a RabbitNet port[i]
- **ADCs**
  Up to 16 ADC channels
- **DACs**
  Up to 2 DAC channels
- **Digital I/O**
  Up to 40 pins for combinations of:
  - Digital Inputs
  - Sinking Outputs
  - Sourcing Outputs
  - Line Drivers
  - Bidirectional lines
- **PWM**
  Up to 4 PWM outputs
- **External Interrupt**
- **External Reset**

i. RabbitNet is a proprietary serial protocol used with a line of peripheral devices for adding extra I/O, ADCs, DACs, relays, or an additional keypad/LCD interface. See the *RabbitNet User's Manual* for more information.

- **Keypad**
  Range of matrix keypad options: m x n, where m+n <=8
- **LCD**
  With or without backlight and/or contrast control
- **Power Routing**
  5 V available on all user-selectable connectors
  3.45 V available on most connectors

1.2 The RabbitFLEX Configurator

When designing and ordering a RabbitFLEX BL300F, you have the opportunity to tailor the board according to the demands of your application and not to limitations imposed by an off-the-shelf SBC. You are in control when you design a board using the RabbitFLEX Configurator.

1.2.1 Configurator Access

The RabbitFLEX Configurator is available at:

www.rabbit.com/products/RabbitFLEX

From the above location you can order a customized board, check previous orders, or peruse the site for ideas and tips for your design.

All information entered at the website is secure and confidential—nobody but you or those whom you authorize will be able to see the board designs you have saved or ordered. You will be asked to log in to your account with a user name and password when you save, order or reload a board design.

1.2.2 Ordering Information and History

An XML file encapsulates the board design. It can be downloaded by a person who has password access to it. The design file is very important. You must safeguard this file to ensure that you will be able to reorder the design or to derive other designs from it.

A customized Dynamic C library file is automatically generated when a board design is ordered. The library file is downloadable from the RabbitFLEX Configurator. This file is necessary for proper operation of library functions for your board.

See Section 6.2 for more information on RabbitFLEX BL300F files.

1.2.3 Assumptions

The following assumptions are made regarding the use of the RabbitFLEX Configurator:

- Compatible Web Browsers - Internet Explorer ver. 6 or later, Mozilla and Firefox are all compatible with the Configurator interface.
- You should have a clear and comprehensive idea of your application parameters before selecting the options for your RabbitFLEX BL300F.
- An electrical engineering background is recommended to make the best decisions among the options available.

1.3 RabbitFLEX Tool Kit

The RabbitFLEX Tool Kit contains the software and the extra hardware needed to develop your application. This section lists the tool kit items.

Tool Kit Items

- *RabbitFLEX BL300F Getting Started Instructions.*
- Programming cable, 10-pin header to DB9 connector with integrated level-matching circuitry.
- 48 V AC, 800 mA center-tapped transformer. A transformer compatible with power outlets in continental Europe is included with kits sold overseas.
- CD, containing Dynamic C 9.41 (or later) with complete product documentation, including this manual: the *RabbitFLEX BL300F User's Manual*.
- Demonstration board with pushbutton switches and LEDs.
- Cable assemblies to access connectors on RabbitFLEX BL300F.
- Hardware bag containing mounting standoffs and screws, female DB9 connector.
- Rabbit 3000 Processor Easy Reference poster.
- Registration card.

1.4 CE Compliance

Equipment is generally divided into two classes.

CLASS A	CLASS B
Digital equipment meant for light industrial use	Digital equipment meant for home use
Less restrictive emissions requirement: less than 40 dB µV/m at 10 m (40 dB relative to 1 µV/m) or 300 µV/m	More restrictive emissions requirement: 30 dB µV/m at 10 m or 100 µV/m

These limits apply over the range of 30 to 230 MHz. The limits are 7 dB higher for frequencies above 230 MHz. Although the test range goes to 1 GHz, the emissions from Rabbit-based systems at frequencies above 300 MHz are generally well below background noise levels.

The RabbitFlex single-board computer has been tested and was found to be in conformity with the following applicable immunity and emission standards. Boards that are CE-compliant have the CE mark.

CE

NOTE: Earlier versions of the RabbitFlex boards that do not have the CE mark are ***not*** CE-compliant.

1.4.1 Immunity

The RabbitFlex series of single-board computers meets the following EN55024/1998 immunity standards.

- EN61000-4-3 (Radiated Immunity)
- EN61000-4-4 (EFT)
- EN61000-4-6 (Conducted Immunity)

Additional shielding or filtering may be required for a heavy industrial environment.

1.4.2 Emissions

The RabbitFlex series of single-board computers meets the following emission standards.

- EN55022:1998 Class A
- FCC Part 15 Class A

Your results may vary, depending on your application, so additional shielding or filtering may be needed to maintain the Class A emission qualification.

1.4.3 Design Requirements for CE Compliance

Note the following requirements for incorporating the RabbitFlex series of single-board computers into your application to comply with CE requirements.

1.4.3.1 General Design Requirements

- The power supply provided with the Tool Kit is for development purposes only. It is the customer's responsibility to provide a CE-compliant power supply for the end-product application.
- The LCD/keypad module was tested with the RabbitFlex, but no ESD testing was done. It is the customer's responsiblity to provide a properly designed enclosure for the LCD/keypad interface to meet ESD requirements.
- When connecting a RabbitFlex single-board computer to outdoor cables, the customer is responsible for providing CE-approved surge/lightning protection.
- Rabbit recommends placing digital I/O or analog cables that are 3 m or longer in a metal conduit to assist in maintaining CE compliance and to conform to good cable design practices.
- When installing or servicing the RabbitFlex board, it is the responsibility of the end-user to use proper ESD precautions to prevent ESD damage to the RabbitFlex board.

1.4.3.2 Safety Design Requirements

- All inputs and outputs to and from the RabbitFlex series of single-board computers must **not** be connected to voltages exceeding SELV levels (42.4 V AC peak, or 60 V DC).
- The lithium backup battery circuit on the RabbitFlex single-board computer has been designed to protect the battery from hazardous conditions such as reverse charging and excessive current flows. Do not disable the safety features of the design.

1.4.4 Interfacing RabbitFlex to Other Devices

Since the RabbitFlex series of single-board computers is designed to be connected to other devices, good EMI reduction practices should be followed to ensure compliance. CE compliance is ultimately the responsibility of the integrator. Additional information, tips, and technical assistance are available from your authorized Rabbit distributor, and are also available on our Web site at: www.rabbit.com.

1.5 What to Expect

This manual describes the RabbitFLEX BL300F, its core module choices and the hardware and software components for the sample applications. Your level of experience determines which chapters to read next.

Chapter 2 - Read this chapter for instructions on connecting the hardware and installing the software you will need to develop your application. This chapter also contains contact information and some trouble-shooting tips if you experience difficulty running some initial sample programs.

Chapter 3 - Read this chapter for an understanding of the physical implementation of the logical functionality of the RabbitFLEX BL300F. This chapter contains a roadmap of the RabbitFLEX board and design units that will allow you to identify the physical location of circuit components, such as specific resistors, capacitors, etc.

Chapter 4 - Read this chapter for information on the RabbitFLEX BL300F options that are available. This chapter contains circuit schematics and characterization tables for the different options.

Chapter 5 - Read this chapter for links and references for more details on the RabbitFLEX BL300F options. This chapter also contains some more in-depth discussions on selected topics.

Chapter 6 - Read this chapter to understand the software specifics needed for programming your RabbitFLEX BL300F.

Appendix A - Contains the electrical, mechanical and environmental specifications of the RabbitFLEX BL300F.

Appendix B - Contains information on power and ground.

Appendix C - Contains information on the demonstration board that comes with the RabbitFLEX BL300F Tool Kit.

Appendix D - Contains information on the RabbitFLEX Keypad/Display Kit

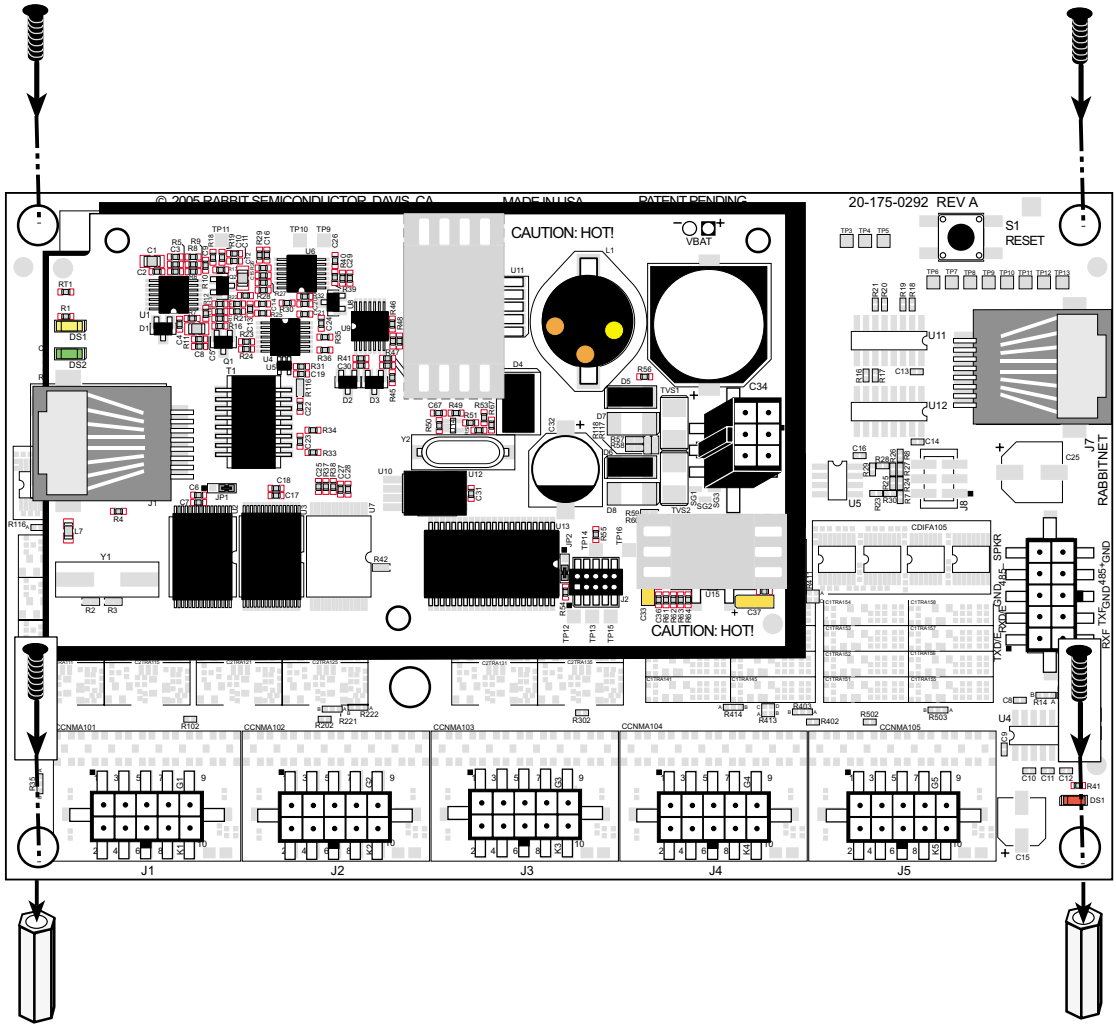
2. GETTING STARTED

The information in this chapter will get you up and running. Follow the instructions given here to install, connect and verify the software and hardware components.

2.1 Hardware Preparation

This section physically prepares the RabbitFLEX board for use in developing and debugging applications. As a way to place the board in a more convenient position for development, the first thing to do is screw on the hex standoffs that came in the RabbitFLEX as shown in Figure 2.1.

Figure 2.1 Connecting Standoffs to the RabbitFLEX BL300F



2.1.1 Hardware Connections

There are two steps to readying the RabbitFLEX BL300F for use with Dynamic C:

1. Connect the programming cable between the RabbitFLEX BL300F and the host PC where you will install Dynamic C.
2. Connect the power supply.

The core module comes attached to the RabbitFLEX BL300F from the factory. The core module and the board it is attached to are a matched set. They must be used together. Do not remove the core module and attach a different core module to the RabbitFLEX board.

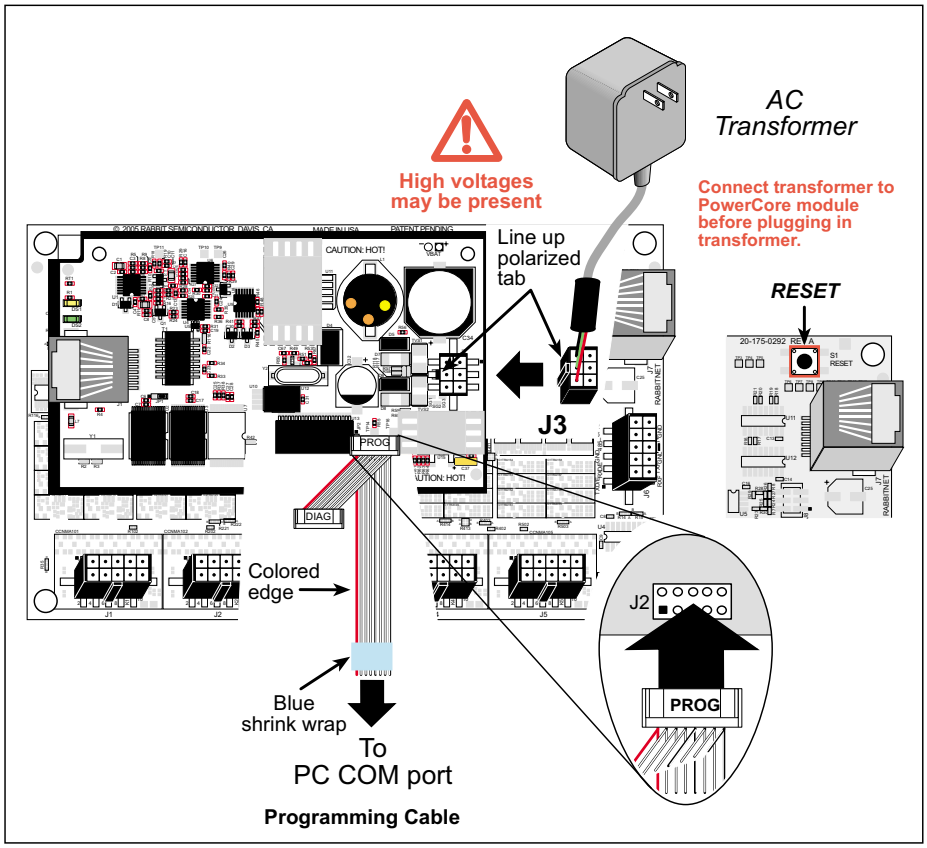
2.1.1.1 Connect Programming Cable

The programming cable completes the communications channel for downloading programs to the core module. Once a program has been downloaded and is running, you can debug it over the programming cable.

Connect the 10-pin connector labeled PROG on the programming cable to header J2 on the PowerCore module as shown in Figure 2.2.

Attach the other end (the DB9 connector) of the programming cable to a COM port on the host PC. Some PCs now come equipped with a USB port and no COM port. In this case, you can connect the programming cable's DB9 connector to a RS-232/USB converter (Rabbit Part No. 540-0070), and then plug the other end of the converter into the USB port of the host PC.

Figure 2.2 Programming and Power Connections



2.1.1.2 Connect Power

Connect the locking plug of the AC adapter to connector J3 on the PowerCore module as shown in Figure 2.2. When the other end of the AC adapter is plugged into a power source, a small red LED labeled DS1 on the RabbitFLEX board (located in the corner by connector J5) will come on.

You can cycle power to the board by unplugging and then plugging in the AC adapter. There is a reset button located on the top-right corner on your board that can be used restart the microcontroller.

2.2 Software Preparation

Dynamic C version 9.41 or later must be installed on the host PC before you can download any programs to the RabbitFLEX BL300F.

2.2.1 Installing Dynamic C

Insert the installation disk or CD in the appropriate disk drive on your PC. The installation should begin automatically. If it doesn't, issue the Windows "Run..." command and type the following command:

```
<disk>:\SETUP
```

The installation program will begin and guide you through the installation process. Your IBM-compatible PC should have at least one free COM port and be running Windows 95 or later.

Now you must make your board-specific library available to Dynamic C. The board-specific library is the one that was generated when you ordered your board. If you have not already downloaded this file, follow these steps.

1. Log in to your RabbitFLEX Configurator account at:

   www.rabbit.com/products/RabbitFLEX

2. Select your RabbitFLEX board(s) from your list of boards to get the corresponding library or libraries, which are named FAxxxxxxx.LIB.
3. Do not change the library name(s). You will be prompted during the download process for the location to save the library or libraries. It is recommended that you use the `LIB\RabbitFlex_SBC40` folder that was created during the Dynamic C installation process.

2.2.2 Starting Dynamic C

Once Dynamic C has been installed, you will have up to three related icons on your PC desktop. One icon is for running Dynamic C, one opens the documentation menu and the last one is for the Rabbit Field Utility, a tool used to download compiled software (.bin files) to a target board.

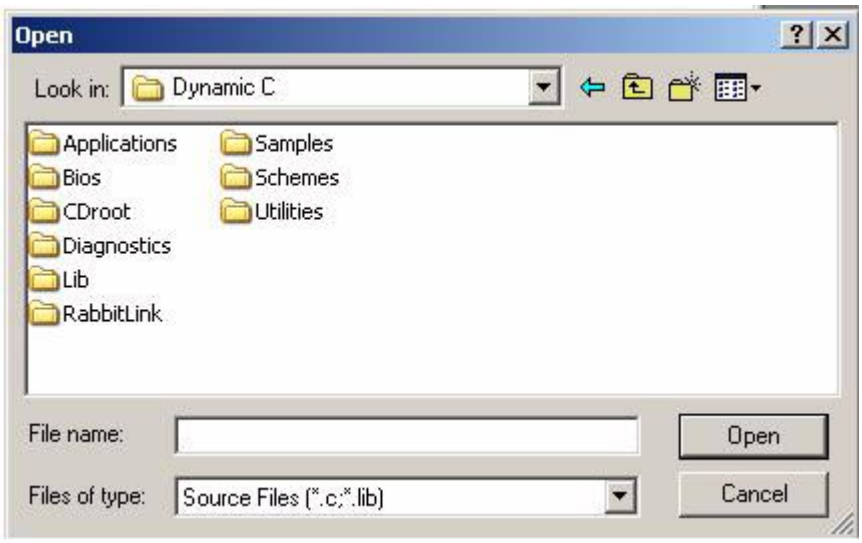
Start Dynamic C by clicking on its desktop icon or by clicking on `dcrabXXXX.exe` in the Dynamic C root directory, where XXXX is the Dynamic C version number.

Dynamic C uses the serial port on the host PC that you specified during installation. If you are using a USB port, you must select "Use USB to Serial Converter" from the Communications tab of the Options | Project Options menu before downloading a program to the RabbitFLEX BL300F.

2.3 Verify Serial Connections

Run the sample program `pong.c` to verify that everything is installed and connected properly. There are three ways to open a program from within Dynamic C: the File menu, the File icon or the keyboard shortcut Ctrl+O. They all open the same dialog box that allows you to browse for the file or type in its pathname. The sample program `pong.c` is in the Samples folder where you installed Dynamic C.

Figure 2.3 Open Dialog Box Showing Dynamic C Root Folder



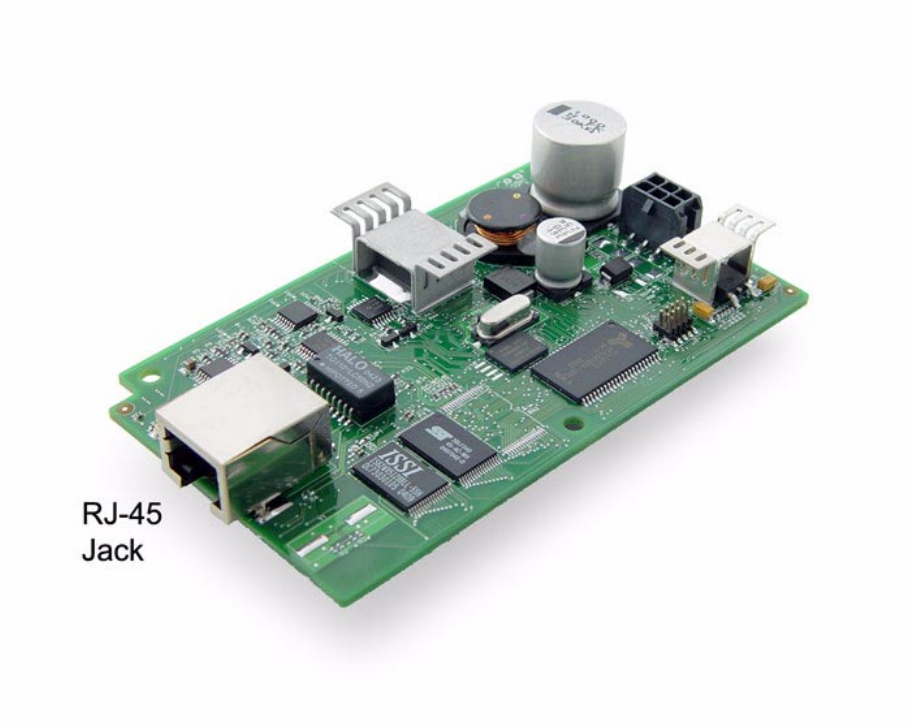
Press function key F9 to compile and run the sample program. Dynamic C will display a status message during compilation and download. If Dynamic C reports no errors and the Stdio window then opens and displays a small square bouncing around in a box you have verified that the serial connection is functioning.

If the program did not compile or run correctly, please see Section 2.5 for some troubleshooting tips that may help diagnose and fix the problem.

2.4 Ethernet Communication

This section describes the additional hardware connections needed to use the RabbitFLEX BL300F Ethernet port. An Ethernet port is available on the PowerCore 3800, but not on the PowerCore 3810. You must have a core module with an Ethernet port and your host computer will need one as well. You may create a direct or an indirect network connection from the target board to the host computer. The core module uses traditional 10Base-T Ethernet, with an RJ-45 jack which is shown in Figure 2.4.

Figure 2.4 RJ-45 Connector on the PowerCore 3800



To communicate on a network, the RabbitFLEX board must be connected to the network with the appropriate cable and the board must be assigned an appropriate IP address. Other addresses may be necessary, depending on whether the network is isolated or is addressable on the Internet. For development, it is better (read easier) to use an isolated network as opposed to one connected to the Internet. Either is possible, and both will be discussed here.

Before proceeding you will need to have two straight through Ethernet cables and a hub, or a crossover Ethernet cable. The Ethernet cables and a 10Base-T Ethernet hub are available from Rabbit in the TCP/IP 10Base-T Accessory Kit. More information is available at: `www.rabbit.com`

2.4.1 Direct Connection

A direct connection means that a crossover cable connects the target board to the host computer running Dynamic C. Plug one end of the crossover cable into the RJ-45 jack on the core module and the other end to the network interface card in your host PC.

2.4.2 Indirect Connection

An indirect connection means that a straight-through cable connects the target board to an Ethernet network via a hub.

2.4.3 Setting IP Addresses

IP addressing is necessary to communicate on networks. IP addresses are like street addresses or phone numbers in that they associate a unique number with an entity. Just as no two devices can have the same phone number, no two devices on a network can have the same IP address.

The Dynamic C implementation of TCP/IP makes network configuration very easy. In most cases an application only needs to define the configuration macro `TCPCONFIG`. This configuration macro is used to define other configuration macros. In the sample programs you will see `TCPCONFIG` defined as 1. This is the default network configuration, which will bring up the Ethernet interface and assign default IP addresses for the system, as well as a netmask.

You only need to set the network configuration in `Lib/tcpip/tcp_config.lib` for most sample programs to work. The following line of code was taken from `tcp_config.lib`:

```
#define _PRIMARY_STATIC_IP  "10.10.6.100"
```

The IP address is a 32-bit number that is expressed in the dotted decimal format shown in the above #define statement. Unless there is a need to set a specific IP address to your Rabbit-based device, you may proceed with the default address of 10.10.6.100. Otherwise, open tcp_config.lib and replace "10.10.6.100" with the IP address for your Rabbit-based device. If your device is on a local network, this change may be sufficient; however, if your device is addressable over the Internet, you must also set the macro `MY_GATEWAY` to the IP address of the default gateway (router) used by your network.

The Dynamic C implementation of TCP/IP supports subnetting. Subnetting is done with a logical bitwise AND of a netmask and the IP address. The following line of code was taken from `tcp_config.lib`:

```
#define _PRIMARY_NETMASK "255.255.255.0"
```

If you do not know the correct values for your IP address and netmask, ask your network administrator.

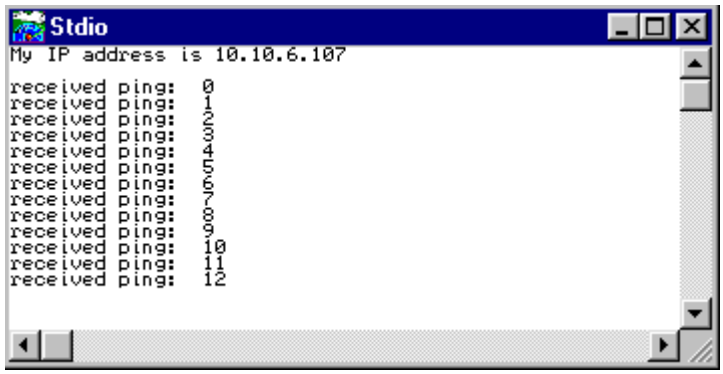
If you have created an isolated network of just the PC running Dynamic C and the RabbitFLEX BL300F, you can use any IP address and netmask that you like as long as they are in dotted decimal format and are on the same network. For example, if using the default netmask, the first three octets of each IP address must match.

2.4.4 Verify the Ethernet Connection

Run the sample program `ping.c` to verify that everything is connected properly. The sample program `ping.c` is in the `\Samples\tcpip` folder where you installed Dynamic C.

Press function key F9 to compile and run the program. Dynamic C will display a status message during compilation and download. If Dynamic C reports no errors and the Stdio window then opens and reports that ping responses were received from the remote host, as shown in the screenshot in Figure 2.5, you have verified that the network connection is functioning.

Figure 2.5 Output from ping.c



2.5 Troubleshooting Tips

This section discusses troubleshooting tips to try when the above instructions did not result in the error-free running of the pong and/or ping program. There are various conditions to double-check.

If you see the message "No Rabbit Processor Detected"

- Be sure the programming cable is attached securely and that it is not misaligned or backwards.
- If your PC is busy running concurrent tasks, try lowering both the Debug and Download baud rates. Or exit the concurrent tasks.
- If you have something like Palm Desktop running, it may monopolize a serial port. Disable this program, or others like it (that is, any program that uses the serial port and stays open in the background).
- Use the power supply that came in the Tool Kit. Make sure it is attached securely.
- If you need to use a USB to serial converter, use the one sold at:

  www.rabbit.com/products/rs232_usb_cable/index.shtml

  since it has been tested for compatibility. Others may work as well, but some may not.
- Do some basic debugging. For example, see if there is power at the core module and check the 3.45 regulator. Figure 2.6 shows the points to probe with a multimeter. They are located in the lower right-hand side of the module and are labeled "A" through "F." Table 2-6 describes the labeled test points and the voltages you should expect to see.

Figure 2.6 Test Points for Checking Voltage Levels on the PowerCore 3800

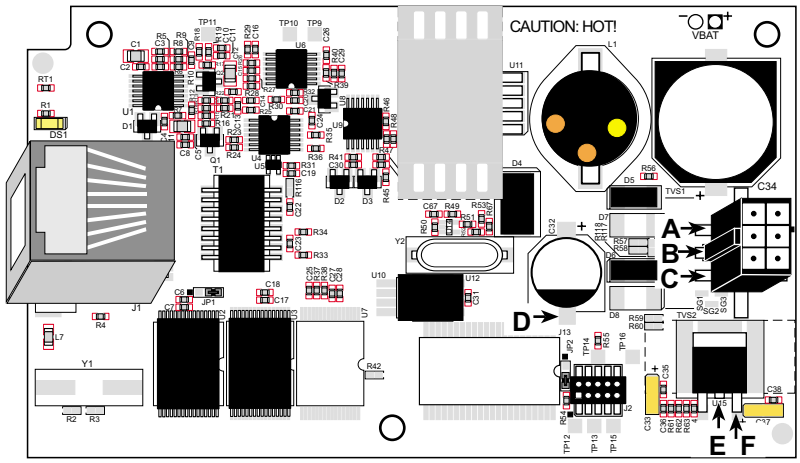


Table 2-6. Description for Test Points in Figure 2.6

Test Points	Expected Measurement
A, **B**, **C**: These pins are the center-tapped power supply input. "B" is ground. Both "A" and "C" measure the voltage coming from the external power supply	From B to A, 12 V AC or higher From B to C, 12 V AC or higher
D: This capacitor terminal is an easy place to access the board-level ground. **E**, **F**: These pins are the output and input, respectively, of the 3.45 V regulator.	From D to E, approx. 3.45 V DC From D to F, approx. 5 V DC

If you see the message "Could not open serial port"

- Another program (for example, a terminal emulator) is already using the COM port you chose for Dynamic C or the programming cable is not attached to the correct COM port on the PC. To change the COM port Dynamic C will use, choose Options > Project Options and click the Communications tab. Select another COM port from the Serial Port pull-down menu.

If you see the message "Timeout while waiting for response from target"

- Check the programming and power cable connections.
- You might have a baud rate mismatch problem. Choose Options > Project Options and click the Communications tab. Select a different value in the Debug Baud Rate dropdown menu.
- If you have a very slow PC or are doing other CPU-intensive tasks at the same time Dynamic C is trying to download, you might get other timeout error messages. Dynamic C needs a lot of CPU time during a download. Try stopping the other tasks to see if it helps.

Problems During and/or at the End of Compilation

- A few computers have trouble handling the default 115,200 baud rate. Change the stop bits from 1 to 2 in the Communications tab of Project Options. If that does not work, slow the baud rate to 57,600 and try the compilation again.

2.7 Contact Information

If there are any questions at this point:

- If you purchased your RabbitFLEX BL300F through a distributor or through a Rabbit partner, contact the distributor or partner first for technical support.
- Check the Rabbit Technical Bulletin Board at: www.rabbit.com/support/bb/.
- Use the Technical Support e-mail form at: www.rabbit.com/support/.

3. Design Implementation and Information

This chapter discusses the implementation of the RabbitFLEX™ SBC features. We start with information about the basic bare board, which is used as the foundation for all RabbitFLEX™ SBC boards. This foundation board is called the base board. Next, we discuss the physical and logical components that make up the feature set selected during the online ordering process.

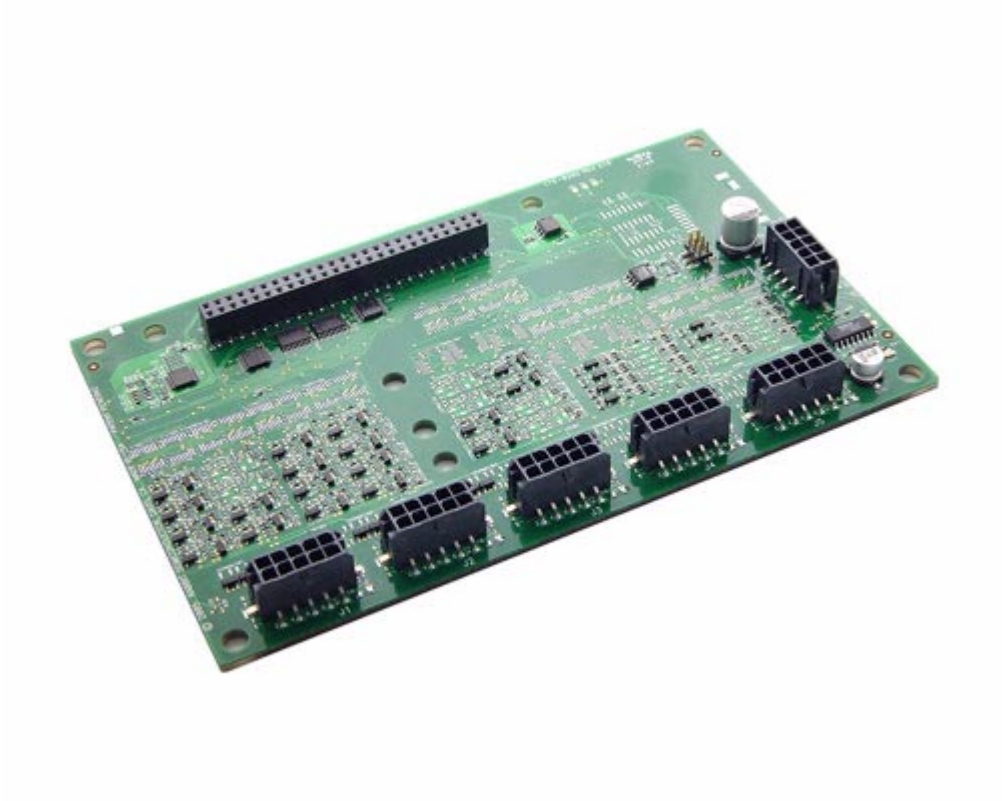
3.1 General Board Information

This section describes the RabbitFLEX BL300F base board and provides a high-level diagram of the subsystem options.

3.1.1 Base Board Layout

The base board for a RabbitFLEX BL300F is pictured in Figure 3.1.

Figure 3.1 Base Board for RabbitFLEX BL300F



3.1.2 Board-Specific Information

The PowerCore module and the board to which it is attached are linked by more than their physical connection. Switching the mated core module is not recommended, as the calibration and design numbers are contained in the core module and will thus no longer match the board.

3.1.2.1 System ID Block

The System ID block is a reserved area in flash memory that contains identification information for the core module and the design number for the RabbitFLEX BL300F. All RabbitFLEX BL300F devices have System ID blocks of version 5 or later. The information fields contained in the System ID block are described in the *Rabbit 3000 Designer's Handbook*.

3.1.2.2 User Block

The System ID block has information about the User block. The User block is where calibration constants are stored. There is memory in the User block for other persistent data that may be required by your application.

3.2 Cells and Circuits

The magic of a flex board is the innovative design of the cells, which are abstracted to a level that allows them to implement one of multiple functions that may be chosen by you, the board designer.

3.2.1 Cell Definition

Cells can be configured or unconfigured. Unconfigured cells are the basic structure unit at the PCB level. An unconfigured cell can be configured to implement one of several logical functions, thus making it a configured cell. A configured cell is one in which circuitry has been assembled to form a single function on a RabbitFLEX board, such as an I/O pin or a serial port.

3.2.2 Cell Descriptions

There are different types of cells available. Each one is designed to handle different types of circuits. Each I/O pin is supported by a one-transistor or two-transistor cell on the RabbitFLEX BL300F board.

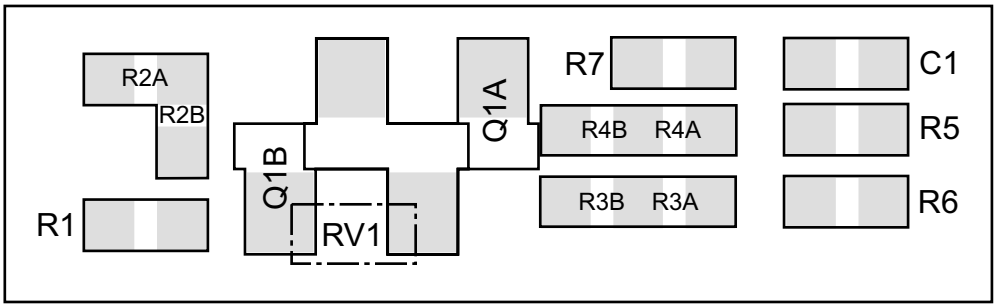
The one- and two-transistor cell layouts pictured in Figure 3.2 and Figure 3.3 can be used along with the board layout pictured in Figure 3.4 to determine the placement of components on your board. The pad labels (R2A, R2B, Q1B, etc.) that you see in the cell layout diagrams correspond to the component designators in the circuit schematics.

For example if you look at the circuit diagram for the 1.4 V trigger threshold digital input, you will see that the 100 ohm resistor is labeled "R6." Looking at the one-transistor cell layout, "R6" is located in the lower-right corner of the cell. The physical placement of "R6" is different when a two-transistor cell is used to implement the 1.4 V digital input.

3.2.2.1 One-Transistor Cells

Figure 3.2 shows the complete layout of a one-transistor cell. A one-transistor cell can handle digital inputs, certain outputs, bidirectional logic and keypads. The one-transistor cells on connector J4 can also be used as analog inputs.

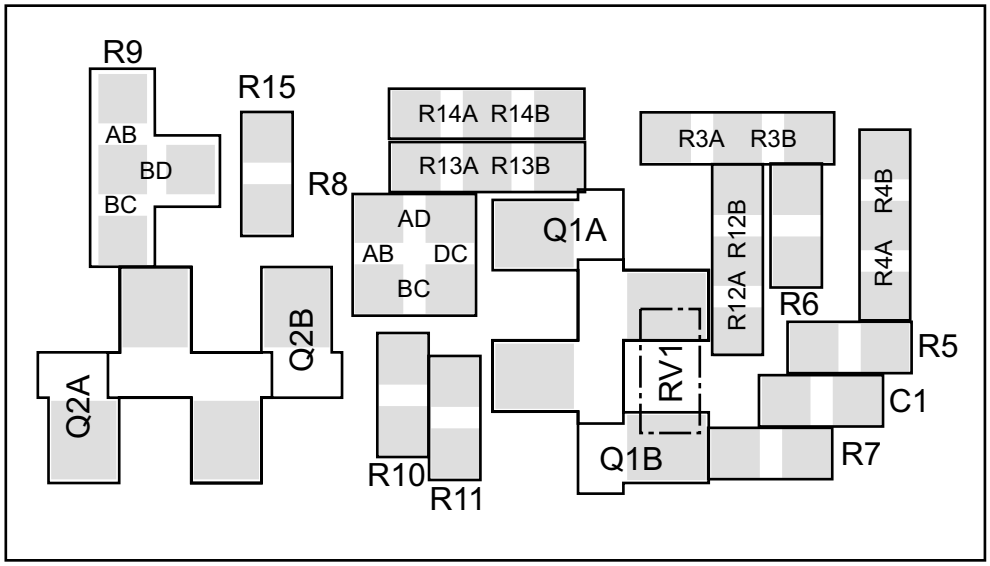
Figure 3.2 One-Transistor Cell



3.2.2.2 Two-Transistor Cells

A two-transistor cell is required for all sourcing outputs and the 1 A sinking output. The two-transistor cells on connector J3 can also be used as analog inputs. You can use any circuit that can be placed in a one-transistor cell, but not vice-versa.

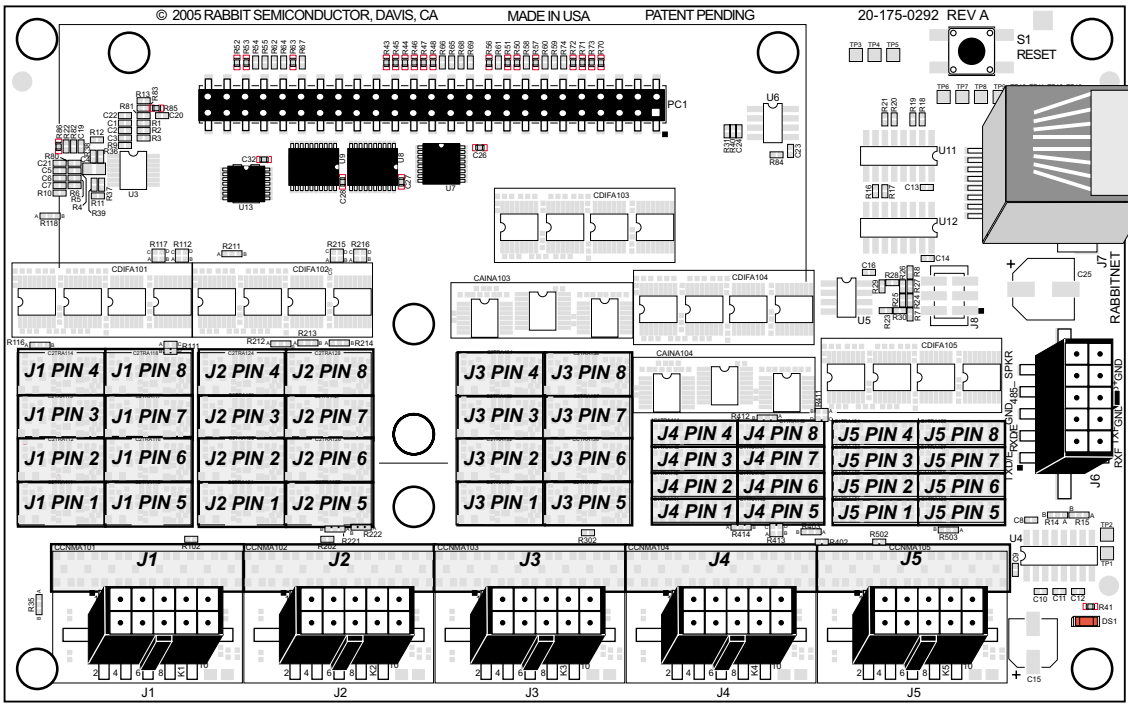
Figure 3.3 Two-Transistor Cell



3.2.3 Board Map

The board map shown in Figure 3.4 identifies the cell placement for each of the I/O pins. Each rectangle labeled "Jx PIN y" starts as an unconfigured cell before a circuit is selected and placed on it, causing it to become a configured cell.

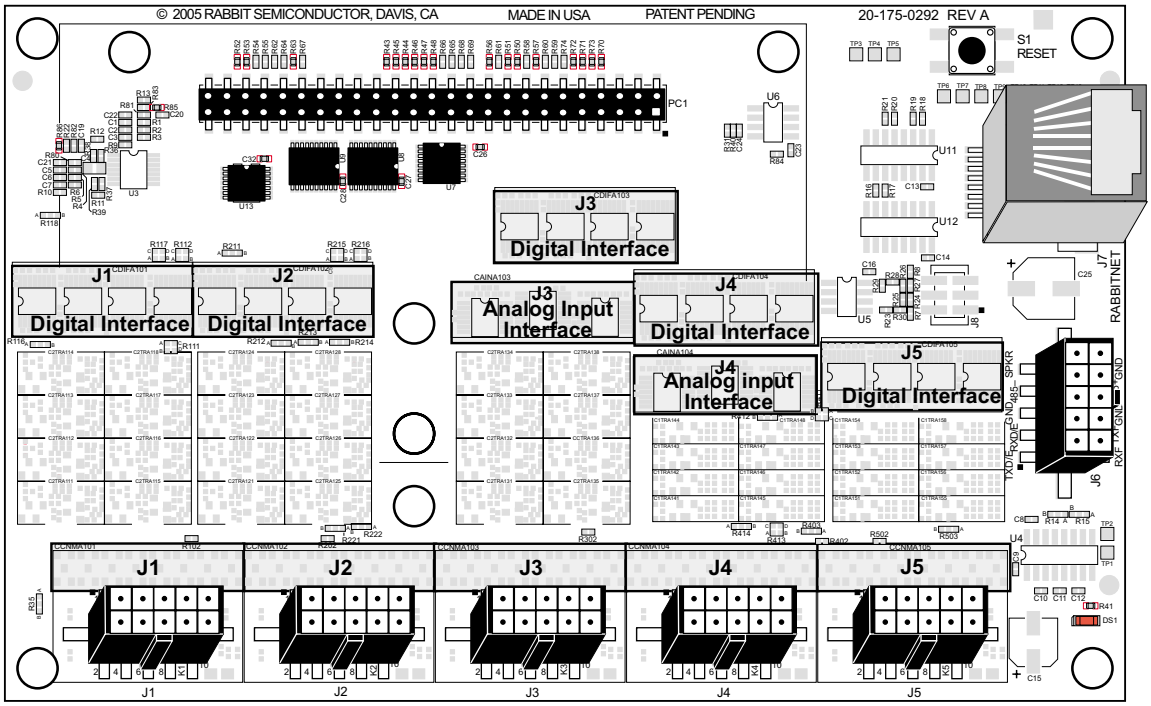
Figure 3.4 Map of Cells Supporting I/O Pins on Connectors J1–J5



Digital I/O is routed to a digital interface cell, which is routed to the core module. Analog inputs are routed to an analog input interface cell, which is also routed to the core module. An analog input cell takes in the ramp generator output, the eight possible analog inputs and three selector pins. A background process continually reads the analog inputs and stores the last values read.

Figure 3.5 shows the locations of the digital and analog interface cells corresponding to connectors J1–J5.

Figure 3.5 Map of Digital and Analog Input Interface Cells



4. RabbitFLEX BL300F Options

This chapter discusses the options available for customizing a RabbitFLEX BL300F. The options fall into one of three main categories: global, pin group or connector.

Global Options

Global options, as the name implies, have a more global effect on the board design than the options in the other two categories. Global options are things like the core module and serial communication channels.

Pin Group Options

Pin group options are circuits that populate more than one pin on a connector. Things like an LCD or keypad are in this category. Multiple pins are configured when an LCD or keypad is chosen.

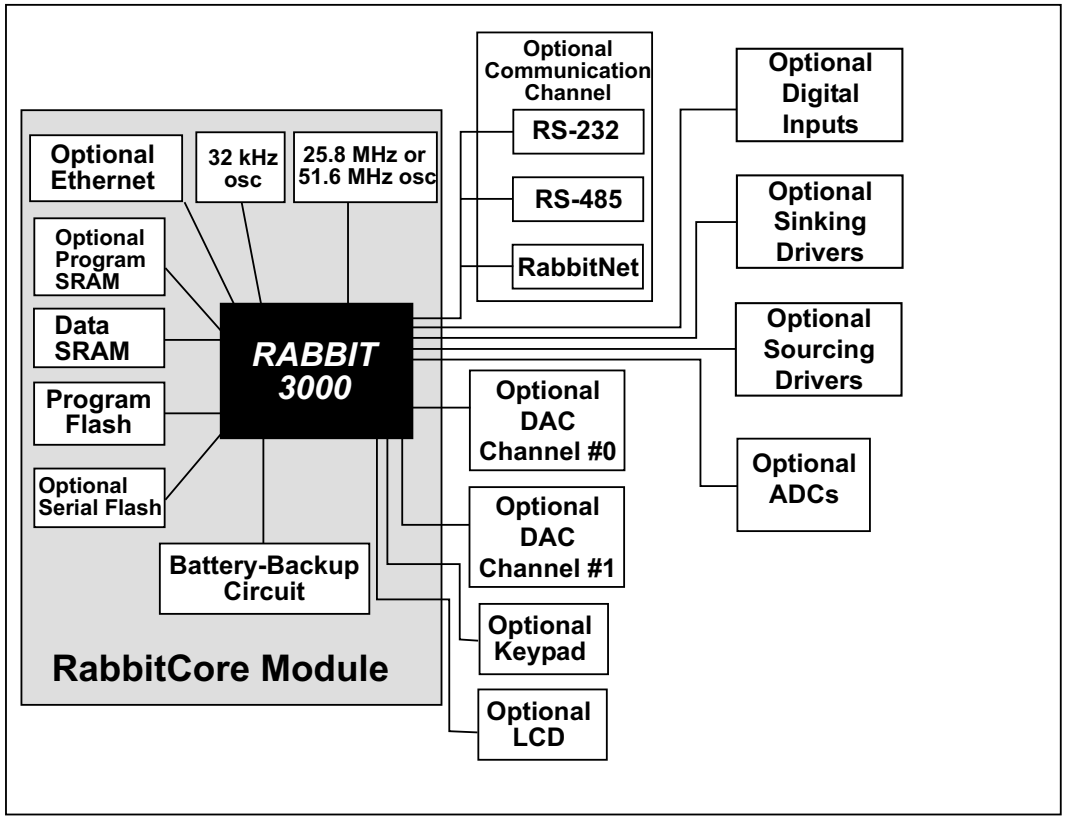
Connector Options

Connector options affect one pin on a connector, like digital or analog I/O.

4.1 Overview of Options

This section gives a visual overview of all available options and the connector/pins used.

Figure 4-1 RabbitFLEX BL300F Block Diagram



4.2 Overview of Connector and Cell Association

The table in this section summarizes the user-configurable connectors, their corresponding unconfigured cells and the available circuitry that can be used to configure the cells. (See Section 3.2 for more information about cells.)

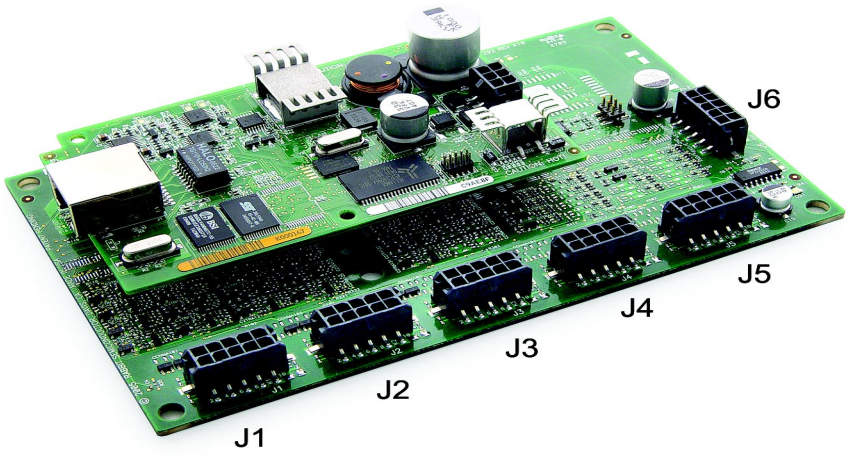
Table 4-1. Connectors and their Unconfigured Cell Types

Connector	Unconfigured Cell Type	Digital		Analog		3.45 V Power	5 V Power
		Digital In	Digital Out	ADC	DAC		
J1	2-transistor	X	X			X	X
J2	2-transistor	X	X		X	X	X
J3	2-transistor	X	X	X		X	X
J4	1-transistor	X	X	X			X
J5	1-transistor	X	X				X

4.2.1 Connector Pinout Diagram

This section identifies the location of the connectors and their customizable options.

Figure 4-2 Location of J1 through J6 on RabbitFLEX BL300F



For obvious reasons, a board-specific pinout diagram is impossible to include in a generally applicable user's manual such as this one. The following diagrams give the pinouts of the user-configurable connectors and the options available for each of the pins. To review the option/pin combinations that exist on your particular board, look in the generated library FAxxxxxxx.lib or the board summary file FAxxxxxxx.txt. These files, which are downloadable from the RabbitFLEX Configurator, tell you which option is connected to each of the configurable pins.

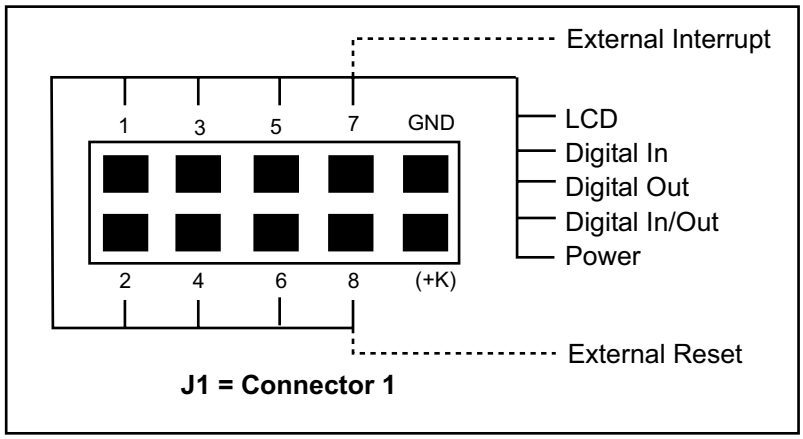
4.2.1.1 Connector 1

Connectors J1 through J5 all have digital I/O, power routing, ground and +K (external power) options. But the range of choices is greater in the digital I/O options for J1 through J3 because the pins are connected to two-transistor cells as opposed to the one-transistor cells of J4 and J5.

Connectors J3 and J4 add the ability to be configured as analog inputs.

If the LCD option is selected, it will populate the 8 available pins of J1.

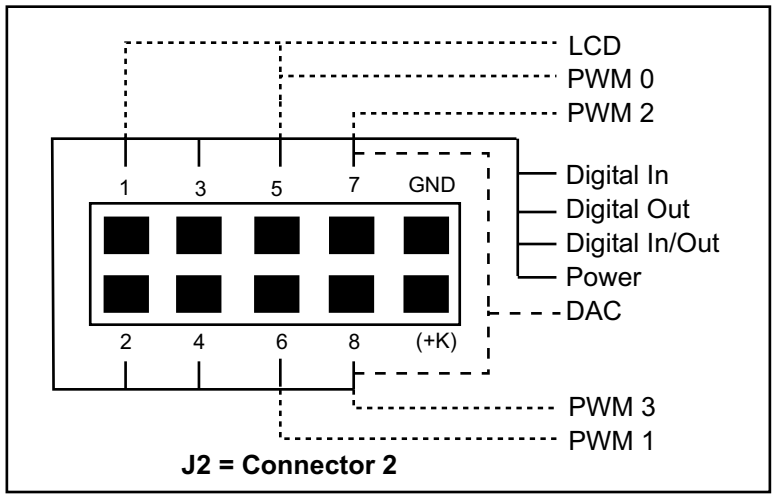
Figure 4-3 Pinout Options for J1



4.2.1.2 Connector 2

The two DAC channels and the PWM outputs are available on J2. The 0-10 V option requires an external +12 V power supply connected to +K on J2.

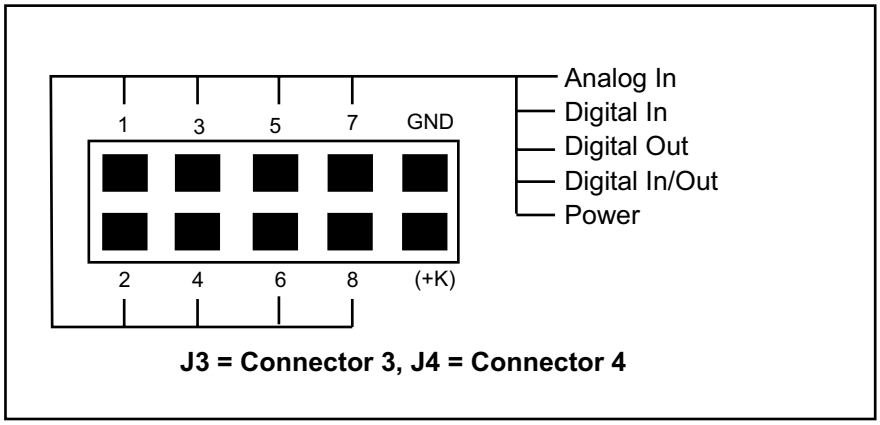
Figure 4-4 Pinout Options for J2



4.2.1.3 Connector 3 and 4

The pinout diagram for J4 looks exactly like the one for J3. The difference is that J4 is made up of one-transistor cells and J3 is made up of two-transistor cells. This difference means that, for example, the pins on J4 can be configured for the 100 mA sinking driver or the 100 ohm line driver, but not the 1 A sinking driver or either of the sourcing drivers available on J1, J2, or J3. Also the 3.45 V power supply is not available on J4.

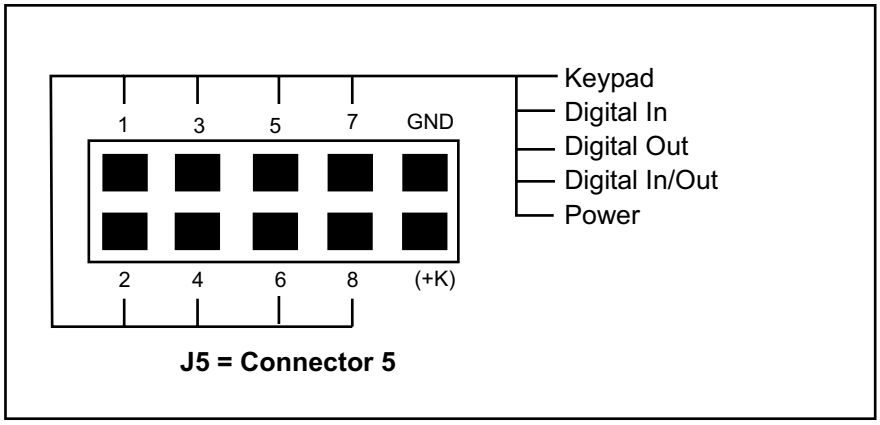
Figure 4-5 Pinout Options for J3 and J4



4.2.1.4 Connector 5

If the keypad option is selected, it will populate pins on J5. The number of pins depends on the number of rows and columns chosen. For a 2x3 keypad, six pins are used. For a 4x4 keypad, all eight configurable pins are used.

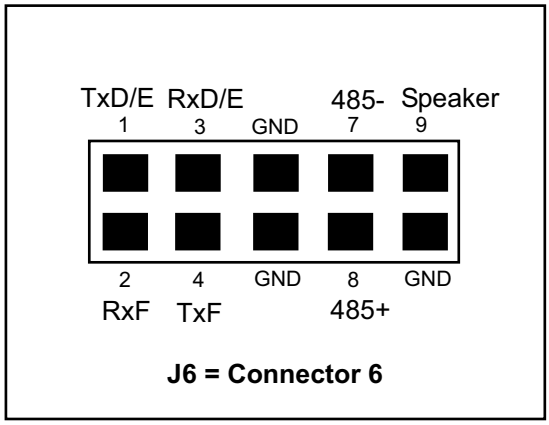
Figure 4-6 Pinout Options for J5f



4.2.1.5 Connector 6

J6 differs from J1-J5, in that its pins can only be customized from the global options page of the Rabbit-FLEX Configurator. This is because J6 is used for serial communications and the speaker.

Figure 4-7 Pinout Options for J6



4.2.2 Connector Mating Information

The connectors J1 through J6 on the RabbitFLEX BL300F are Molex Micro-Fit 3.0 connectors, Molex part # 43045-1018. There are a number of companies that offer custom harness assemblies that could be mated with these connectors. Molex has a list of recommended cable assembly manufacturers on their website:

www.molex.com/product/harness/stars.html

Rabbit recommends both Blaylock's, Inc. and Mann Organization for custom cable assembly.

www.blaylocksinc.com/

www.mannorganization.com/index.html

4.3 RabbitFLEX BL300F Option Specifications

This section lists all of the RabbitFLEX BL300F options, along with basic information and specifications.

4.3.1 Core Module

There are currently two core modules available for the RabbitFLEX BL300F: the PowerCore 3800 and PowerCore 3810. The table in this section compares the features that differ between these modules. See the *PowerCore FLEX User's Manual* for more information about these core modules.

Table 4-2. PowerCore Module Differences

Feature	PowerCore 3800	PowerCore 3810
Microprocessor	Rabbit 3000 @ 51.6 MHz	Rabbit 3000 @ 25.8 MHz
Ethernet	Yes	No
Onboard Power Supply	2 A, 8-43 V DC, 24-60 V AC (with center-tapped transformer otherwise 12-36 V AC)	1 A, 8-40 V DC, 1-19-57 V AC (with center-tapped transformer, otherwise 10-29 V AC)
Memory	512K Flash 1 MB SRAM (512K Code, 512K Data) 1 MB Serial Flash	512K Flash 256K SRAM

As you can see from the information in Table 4-2, the PowerCore 3800 is faster, can communicate on a network, supplies more power and has more memory than the PowerCore 3810. The serial flash on the PowerCore 3800 allows the use of the Dynamic C FAT module, adding a popular standard for file storage to your application. However, if your application requires a low-power solution and only requires serial communication, the PowerCore 3810 would be an appropriate choice.

4.3.2 RS-232

You can select one of the following RS-232 channel configurations:

- one 3-wire channel, or
- two 3-wire channels, or
- one 5-wire channel

Serial port F is used for the first 3-wire channel. If you select two 3-wire channels, the second one will use serial port D or E. A 5-wire channel will use serial port F for transmit/receive and serial port D or E for the handshaking lines. Note that whether to use serial port D or E is a decision made when you are designing/ordering your board. Once the board is built you cannot change which of these two serial ports to use.

The API function `serMode()` allows you to enable or disable hardware flow control.

The serial channel lines come out on J6 (see Figure 4-2). All signals are true RS-232 signals; the board provides the voltage output, slew rate, and input voltage immunity required to meet the RS-232 serial communication protocol. Basically, the signals from the Rabbit 3000 are converted to RS-232 signal levels. The polarity is reversed in an RS-232 circuit so that a +3.3 V input becomes approximately -7 V and 0 V is output as +7 V. The RS-232 transceiver also provides the proper line loading for reliable communication.

The following table lists the different capabilities of the serial ports D and E. More information on these serial ports can be found in the *Rabbit 3000 Microprocessor User's Manual*.

Table 4-3.

Serial Port D	Serial Port E
Asynchronous mode up to the baud rate of the system clock divided by 8	Asynchronous mode up to the baud rate of the system clock divided by 8
Clocked serial mode	HDLC
SPI device interfacing	SDLC

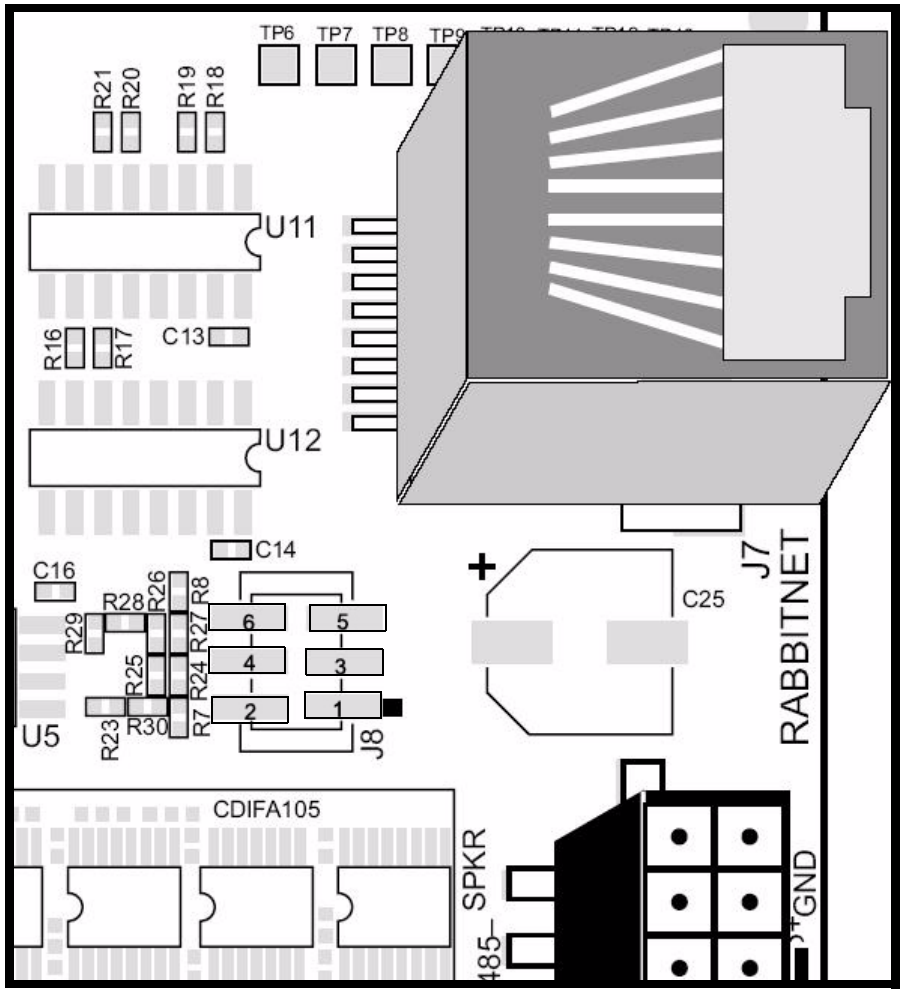
An asynchronous port can handle 7 or 8 data bits. A 9th bit address scheme, where an additional bit is sent to mark the first byte of a message, is also supported.

The RS-232 protocol is implemented by the library `RS232.lib`, which can be found in the folder named "Lib" where you installed Dynamic C.

4.3.3 RS-485

You can select an RS-485 communications channel with or without termination or jumpered termination for the most flexibility. The header J8, between J6 and the RS-485/RabbitNet port, is the header for RS-485 jumpered termination. J8 pin numbers are marked in Figure 4-8.

Figure 4-8 Location of J8 for RS-485 Jumper Termination



See Table 4-4 to determine jumper setting for enabling/disabling termination.

Table 4-4. Jumper Settings for RS-485 Termination

Termination Status	Jumpered Pins
Enabled	1 - 2 and 5 - 6
Disabled	No jumpers

For more information on termination see Section 5.2.

The RS-485 lines come out on connector J6 (see Figure 4-2). Signal A is pin 8 (485+) and signal B is pin 7 (485-). The RS-485 protocol is a packet driver implemented by the library `packet.lib`, which can be found in the folder named "lib" where you installed Dynamic C.

4.3.4 RabbitNet

RabbitNet is a high-speed synchronous protocol used to connect peripheral cards to a master and to allow them to communicate with each other. The available peripheral cards are:

- Digital I/O
- ADC
- DAC
- Display/Keypad Interface
- Relay

For complete information on RabbitNet see the *RabbitNet Peripheral Cards User's Manual* and then any manuals specific to the peripheral card you are interested in.

4.3.5 LCD

The RabbitFLEX™ SBC Configurator will automatically assign pins from connectors J1 and J2 to interface with the LCD. Pins from J2 are used only if you choose the backlight option (pin 1) and/or the contrast control option (pin 5).

The LCD must have a 4-bit data interface. Note that the RabbitFLEX BL300F LCD driver is intended to support modules based on the Hitachi HD44780 chipset, but we cannot guarantee compatibility will all LCD modules based on this chipset. To ensure compatibility, use the LCD module provided for the RabbitFLEX BL300F in the RabbitFLEX Keypad/Display Kit. For more information on the 4 x 20 character display in the kit, see Appendix D.

4.3.6 Keypad

The RabbitFLEX BL300F is compatible with XY matrix keypads, also known a crosspoint keypads. The keypad options are m x n, where m + n <= 8. The specific options available are:

- 1x3, 1x4, 1x5, 1x6, 1x7
- 2x3, 2x4, 2x5, 2x6
- 3x3, 3x4, 3x5
- 4x4

4.3.6.1 Keypad Inputs

The RabbitFLEX Configurator will automatically assign pins from connector J5 as the keypad inputs and outputs. Keypad inputs and outputs are digital I/O circuits. Keypad inputs are read by the API function `flexKeyProcess()`. This function must be called frequently in a software application in order to poll the keypad for key presses. See sample program `Samples/RabbitFLEX_BL300F/keypad.c` for an example of how to use the keypad API functions. (The "RabbitFLEX_BL300F" folder was formerly named "RabbitFLEX_SBC40.")

Figure 4-9 Keypad Input Circuit Schematic

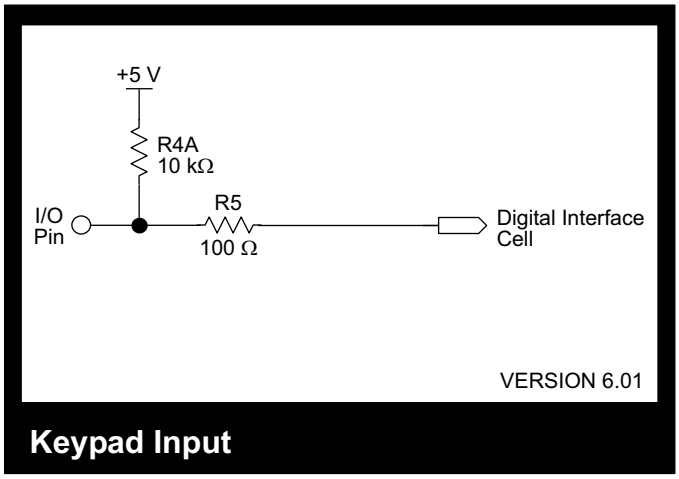


Table 4-5. Keypad Input Characteristics

Symbol	Parameter	Note	Min	Typical	Max	Unit
DC						
V_{IW}	Working Input Voltage		0		5	V
V_{IM}	Maximum Input Voltage		-0.5		6	V
V_{IH}	High Level Input Voltage		2.0			V
V_{IL}	Low Level Input Voltage				0.8	V
V_T	Trigger Threshold Voltage	1		1.4		V
I_{IH}	High Level Input Current	2		-0.3		mA
I_{IL}	Low Level Input Current	3		-0.42		mA
I_I	Input Current	4	-1		1.5	mA
I_{QH}	High Level Quiescent Current	2, 5, 6		0.3		mA
I_{QL}	Low Level Quiescent Current	3, 5, 6		0.42		mA
AC						
t_{I1}	Input Time Delay to 1	7		500		nS
t_{I0}	Input Time Delay to 0	8		100		nS

Note 1: $(V_{IH} + V_{IL}) / 2$

Note 2: at V_{IH}

Note 3: at V_{IL}

Note 4: Over Maximum Input Voltage range

Note 5: From PowerCore Supply5

Note 6: $I_Q = ((5 - V_{IW}) / 10)$ mA over Working Input Voltage Range

Note 7: Time delay from a cell pin input voltage change from 0 to 5 Volts to see V_{IH} at the cell output

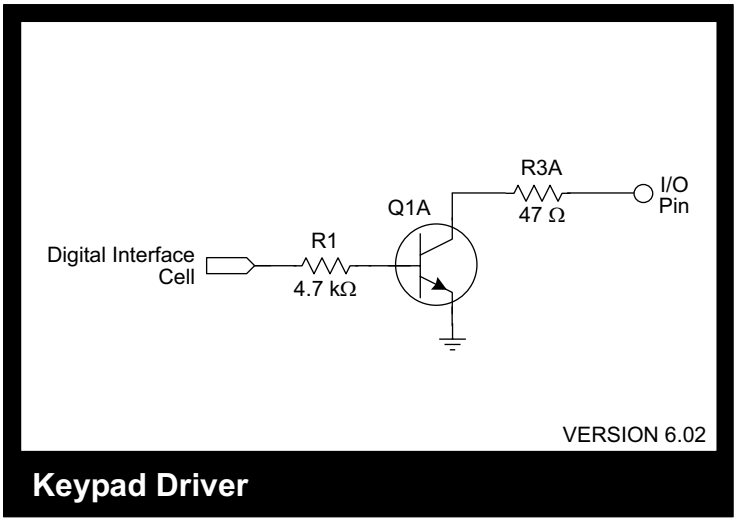
Note 8: Time delay from a cell pin input voltage change from 5 to 0 Volts to see V_{IL} at the cell output

v1.03
cell v6.01

4.3.6.2 Keypad Outputs

The RabbitFLEX Configurator will automatically assign pins from connector J5 as the keypad inputs and outputs. Keypad inputs and outputs are digital I/O circuits. See sample program `Samples/RabbitFLEX_BL300F/keypad.c` for an example of how to use the keypad API functions. (The "RabbitFLEX_BL300F" folder was formerly named "RabbitFLEX_SBC40.")

Figure 4-10 Keypad Driver Circuit Schematic



Power On/Reset State: Off, High impedance

Table 4-6. Keypad Driver Characteristics

Symbol	Parameter	Note	Min	Typical	Max	Unit
DC						
V_S	Supply Voltage		0		5	V
V_{OL}	Low Level Output Voltage				0.8	V
I_{OL}	Low Level Output Current		10			mA
I_L	Leakage Current				10	uA
I_{QH}	High Level Quiescent Current	1		1		uA
I_{QL}	Low Level Quiescent Current	1		1		mA
AC						
t_{O1}	Output Time Delay to 1	2		1		uS
t_{O0}	Output Time Delay to 0	3		72		nS
						v1.02
						cell v6.02

Note 1: From PowerCore Supply5

Note 2: Delay from a cell input voltage change from logical 1 to 0 to see V_{OH} at the output pin with 10K pullup

Note 3: Delay from a cell input voltage change from logical 0 to 1 to see V_{OL} at the output pin at I_{OL}

4.3.7 External Interrupt

The external interrupt can be configured to use J1, pin 7 on the RabbitFLEX BL300F. The external interrupt is enabled using the Rabbit's parallel port E, pin 0 (PE0), which is associated with interrupt vector 0, located at EIR:0x000. The following steps must be taken to prepare the external interrupt for use:

1. Write the interrupt vector to the external interrupt table. The interrupt vector is typically a call to an interrupt service routine (ISR). If it can fit into the 16-byte table entry, it is possible to place the ISR directly into the external interrupt table.
2. Write to I/O register I0CR to select what edges will be detected and the interrupt priority.

See the *Rabbit 3000 Microprocessor User's Manua*l for more information on interrupts.

4.3.8 External Reset

The external reset can be configured to use J1, pin 8 on the RabbitFLEX BL300F. This master reset input will initialize everything in the Rabbit 3000 except for the real-time clock registers. For more information regarding system reset, see the *Rabbit 3000 Microprocessor User's Manual*.

When a battery is installed, battery backup of the real-time clock and the RAM selected by /CS1 occurs during reset and powerdown. For more information regarding battery-backed circuits, see the *PowerCore FLEX User's Manual*.

4.3.9 PWM Outputs

PWM stands for pulse width modulation, which allows the creation of a digital pulse train with a variable on/off duty cycle. This can be filtered to create an analog voltage, or used directly to control many devices, such as lamps or LEDs. Creating an analog voltage then feeding it to an analog driver or amplifier creates a DAC system.

Four PWM outputs are available on a RabbitFLEX BL300F. A Rabbit PWM output has 10-bit resolution and consists of a train of pulses periodic on a 1024-count frame with a duty cycle that varies from 1/1024 to 1024/1024

Table 4-7. Location of PWM Outputs on RabbitFLEX BL300F

PWM Output	Connector and Pin
PWM 0	J2, pin 5
PWM 1	J2, pin 6
PWM 2	J2, pin 7
PWM 3	J2, pin 8

Each PWM output can be further configured as:

- Direct Output
- Sinking Driver, 100 mA
- Sinking Driver, 1 A, +K protected
- Sourcing Driver, +K V, 400 mA

The direct output has the advantage of being a push/pull driver. It has the same current capacity as a general output on the Rabbit: 6.8 mA, sinking or sourcing. A direct PWM output could be used as a control for a high impedence device.

For applications that need more current than is supplied with a direct PWM output, select one of the sinking drivers or the sourcing driver option. The driver options add current to the PWM output, giving it the ability to drive an low impedence load such as a motor or a solenoid.

4.3.10 Digital Inputs

Digital input circuits are available on connectors J1 through J5, pins 1 through 8. For a dedicated digital input, you may select from the following threshold voltages or special purpose inputs:

- 1.4 V
- 2.8 V
- 4.4 V
- Bidirectional Logic
- Contact Input

4.3.10.1 Digital Input 1.4 V Threshold

This section presents the circuit schematic and the characteristics for the 1.4 V threshold digital input.

Figure 4-11 Digital Input 1.4 V Threshold Circuit

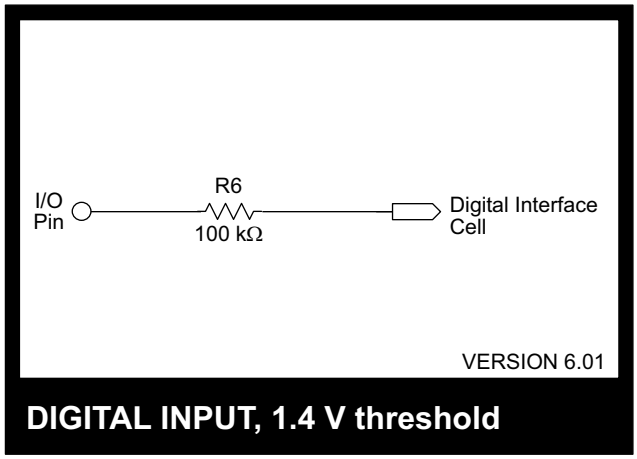


Table 4-8. Digital Input 1.4 V Threshold Characteristics

Symbol	Parameter	Note	Min	Typical	Max	Unit
DC						
V_{IW}	Working Input Voltage		0		5	V
V_{IM}	Maximum Input Voltage		-40		40	V
V_{IH}	High Level Input Voltage		2.0			V
V_{IL}	Low Level Input Voltage				0.8	V
V_T	Trigger Threshold Voltage	1		1.4		V
I_{IH}	High Level Input Current	2		1		uA
I_{IL}	Low Level Input Current	3		1		uA
I_I	Input Current	4	-0.5		0.5	mA
I_{QH}	High Level Quiescent Current	5		1		uA
I_{QL}	Low Level Quiescent Current	5		1		uA
AC						
t_{I1}	Input Time Delay to 1	6		1.5		uS
t_{I0}	Input Time Delay to 0	7		5.5		uS

Note 1: $(V_{IH} + V_{IL}) / 2$ v1.03

Note 2: at V_{IH} cell v6.01

Note 3: at V_{IL}

Note 4: Over Maximum Input Voltage range

Note 5: From PowerCore Supply5

Note 6: Time delay from a cell pin input voltage change from 0 to 5 Volts to see V_{IH} at the cell output

Note 7: Time delay from a cell pin input voltage change from 5 to 0 Volts to see V_{IL} at the cell output

4.3.10.2 Digital Input 2.8 V Threshold

This section presents the circuit schematic and the characteristics for the 2.8 V threshold digital input.

Figure 4-12 Digital Input 2.8 V Threshold Circuit

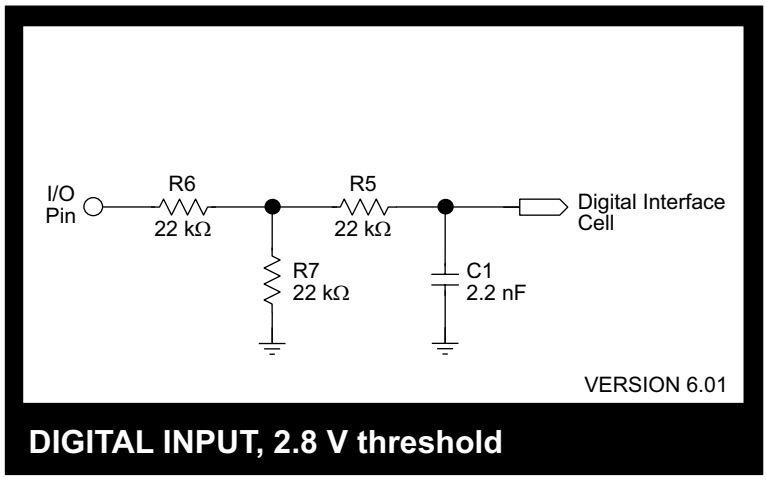


Table 4-9. Digital Input 2.8 V Threshold Characteristics

Symbol	Parameter	Note	Min	Typical	Max	Unit
DC						
V_{IW}	Working Input Voltage		0		10	V
V_{IM}	Maximum Input Voltage		-40		40	V
V_{IH}	High Level Input Voltage		4.0			V
V_{IL}	Low Level Input Voltage				1.6	V
V_T	Trigger Threshold Voltage	1		2.8		V
I_{IH}	High Level Input Current	2		91		uA
I_{IL}	Low Level Input Current	3		36		uA
I_I	Input Current	4	-1.25		1.25	mA
I_{QH}	High Level Quiescent Current	5		1		uA
I_{QL}	Low Level Quiescent Current	5		1		uA
AC						
t_{I1}	Input Time Delay to 1	6		37		uS
t_{I0}	Input Time Delay to 0	7		125		uS

Note 1: $(V_{IH} + V_{IL}) / 2$

Note 2: at V_{IH}

Note 3: at V_{IL}

Note 4: Over Maximum Input Voltage range

Note 5: From PowerCore Supply5

Note 6: Time delay from a cell pin input voltage change from 0 to 10 Volts to see V_{IH} at the cell output

Note 7: Time delay from a cell pin input voltage change from 10 to 0 Volts to see V_{IL} at the cell output

v1.03
cell v6.01

4.3.10.3 Digital Input 4.4 V Threshold

This section presents the circuit schematic and the characteristics for the 4.4 V threshold digital input.

Figure 4-13 Digital Input 4.4 V Threshold Circuit

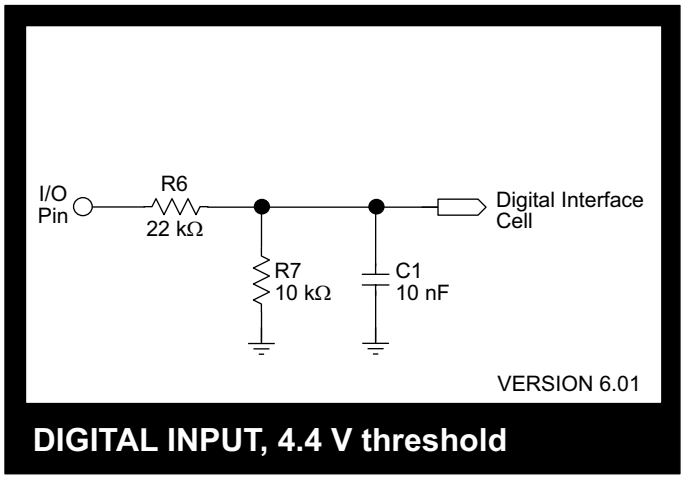


Table 4-10. Digital Input 4.4 V Threshold Characteristics

Symbol	Parameter	Note	Min	Typical	Max	Unit
DC						
V_{IW}	Working Input Voltage		0		16	V
V_{IM}	Maximum Input Voltage		-40		40	V
V_{IH}	High Level Input Voltage		6.4			V
V_{IL}	Low Level Input Voltage				2.5	V
V_T	Trigger Threshold Voltage	1		4.45		V
I_{IH}	High Level Input Current	2		200		uA
I_{IL}	Low Level Input Current	3		78		uA
I_I	Input Current	4	-2		2	mA
I_{QH}	High Level Quiescent Current	5		1		uA
I_{QL}	Low Level Quiescent Current	5		1		uA
AC						
t_{I1}	Input Time Delay to 1	6		35		uS
t_{I0}	Input Time Delay to 0	7		117		uS

Note 1: $(V_{IH} + V_{IL}) / 2$	v1.03
Note 2: at V_{IH}	cell v6.01

Note 3: at V_{IL}

Note 4: Over Maximum Input Voltage range

Note 5: From PowerCore Supply5

Note 6: Time delay from a cell pin input voltage change from 0 to 10 Volts to see V_{IH} at the cell output

Note 7: Time delay from a cell pin input voltage change from 10 to 0 Volts to see V_{IL} at the cell output

4.3.10.4 Contact Input

This section presents the circuit schematic and the characteristics for the contact input.

A contact input is designed to be connected to a switch. It has a longer time constant than the other digital inputs to quiet the noise generated by a switch closure. And it already has the pullup allowing simple wiring of the switch between the input and ground. Basically, it is a hardware switch debouncer, though it may be still be neccesary to use software to fully debounce the switch.

Figure 4-14 Contact Input Circuit

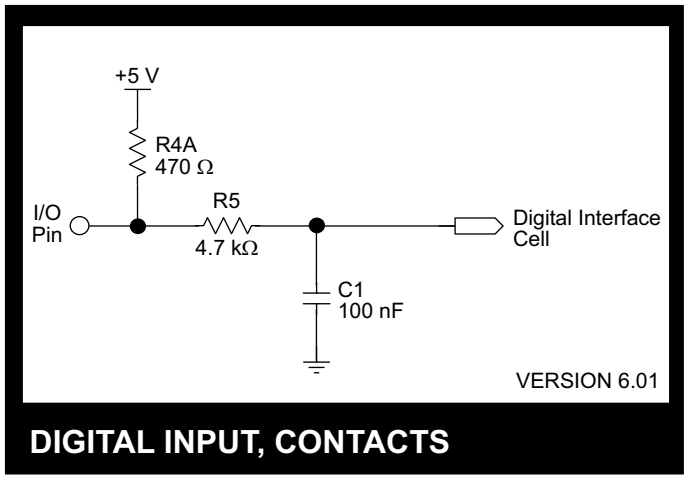


Table 4-11. Contact Input Characteristics

Symbol	Parameter	Note	Min	Typical	Max	Unit
DC						
V_{IW}	Working Input Voltage		0		5	V
V_{IM}	Maximum Input Voltage		-0.5		10.5	V
V_{IH}	High Level Input Voltage		2.0			V
V_{IL}	Low Level Input Voltage				0.8	V
V_T	Trigger Threshold Voltage	1		1.4		V
I_{IH}	High Level Input Current	2		-6.4		mA
I_{IL}	Low Level Input Current	3		-8.9		mA
I_I	Input Current	4	-13		13	mA
I_{QH}	High Level Quiescent Current	2, 5, 6		6.4		mA
I_{QL}	Low Level Quiescent Current	3, 5, 6		8.9		mA
AC						
t_{I1}	Input Time Delay to 1	7		240		uS
t_{I0}	Input Time Delay to 0	8		780		uS

Note 1: $(V_{IH} + V_{IL}) / 2$ — v1.03

Note 2: at V_{IH} — cell v6.01

Note 3: at V_{IL}

Note 4: Over Maximum Input Voltage range

Note 5: From PowerCore Supply5

Note 6: $I_Q = ((5 - V_{IW}) * 2.13)$ mA over Working Input Voltage Range

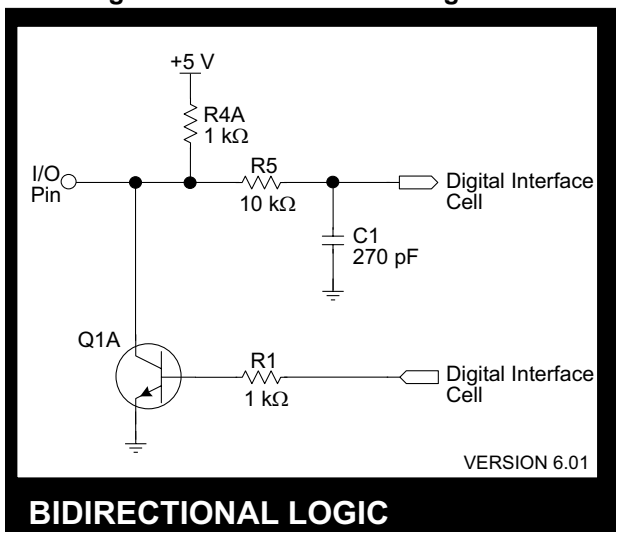
Note 7: Time delay from contact opening to see V_{IH} at the cell output

Note 8: Time delay from contact closing to see V_{IL} at the cell output

4.3.10.5 Bidirectional Logic

This section presents the circuit schematic and the characteristics for the bidirectional logic. A birdirectional logic circuit is useful when talking to an offboard device that communicates using the same line for transmit and receive, such as an I^2C device.

Figure 4-15 Bidirectional Logic Circuit



Power On/Reset State: Input

Table 4-12. Bidirectional Logic Characteristics

Symbol	Parameter	Note	Min	Typical	Max	Unit
DC						
INPUT						
V_{IW}	Working Input Voltage		0		5	V
V_{IM}	Maximum Input Voltage		-3		13	V
V_{IH}	High Level Input Voltage		2.0			V
V_{IL}	Low Level Input Voltage				0.8	V
V_T	Trigger Threshold Voltage	1		1.4		V
I_{IH}	High Level Input Current	2		-3		mA
I_{IL}	Low Level Input Current	3		-4.2		mA
I_I	Input Current	4	-27.5		10	mA
I_{QH}	High Level Quiescent Current	2, 5, 6		3		mA
I_{QL}	Low Level Quiescent Current	3, 5, 6		4.2		mA
OUTPUT						
V_S	Supply Voltage	5	4.75	5.0	5.25	V
V_{OH}	High Level Output Voltage		2.0			V
V_{OL}	Low Level Output Voltage				0.8	V
I_{OH}	High Level Output Current	2	-2.6			mA
I_{OL}	Low Level Output Current	3	25			mA
I_{QH}	High Level Quiescent Current	2, 5, 6		3		mA
I_{QL}	Low Level Quiescent Current	5		9		mA
AC						
INPUT						
t_{I1}	Input Time Delay to 1	7		1.5		uS
t_{I0}	Input Time Delay to 0	8		6		uS
OUTPUT						
t_{O1}	Output Time Delay to 1	9		750		nS
t_{O0}	Output Time Delay to 0	10		20		nS

Note 1: $(V_{IH} + V_{IL}) / 2$ v1.06

Note 2: at V_{IH} for input, V_{OH} for output cell v6.01

Note 3: at V_{IL} for input, V_{OL} for output

Note 4: Over Maximum Input Voltage range

Note 5: From PowerCore Supply5

Note 6: $I_Q = (5 - V_{IW})$ mA over Working Input Voltage Range.

Note 7: Time delay from a cell pin input voltage change from 0 to 5 Volts to see V_{IH} at the cell output

Note 8: Time delay from a cell pin input voltage change from 5 to 0 Volts to see V_{IL} at the cell output

Note 9: Time delay from a cell input voltage change from logical 1 to 0 to see V_{IH} at the output pin at no load

Note 10: Time delay from a cell input voltage change from logical 0 to 1 to see V_{IL} at the output pin At I_{OL} Max

4.3.11 Digital Outputs

There are several digital output options on a RabbitFLEX BL300F.They are listed in Table 4-13. Note that not all options are available on all five connectors.

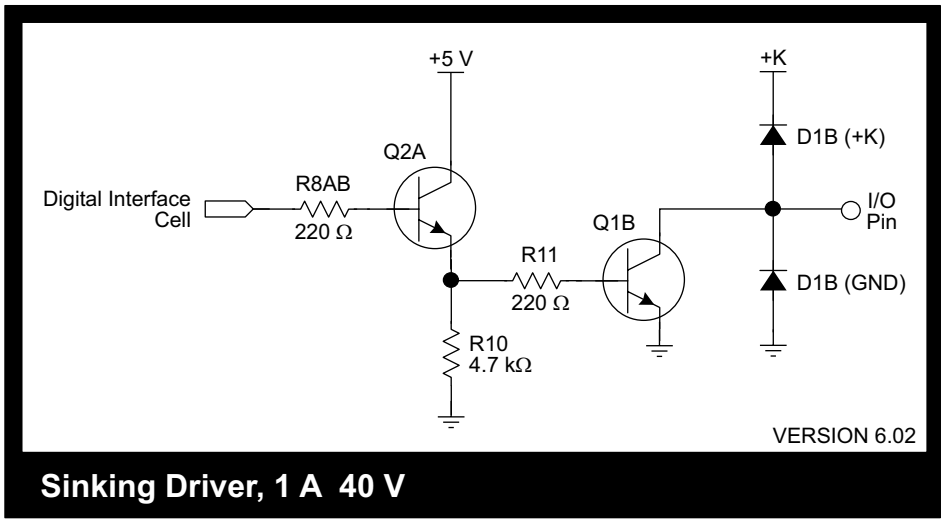
Table 4-13. Digital Output Specifications

RabbitFLEX BL300F Option	Connector Availability	Diode Protection
Sinking Driver, 1 A	1 - 3	Yes
Sinking Driver, 100 mA	1 - 5	No
Sourcing Driver, +K V, 400 mA	1 - 3	No
Sourcing Driver, 5 V, 50 mA (short circuit current limited)	1 - 3	Yes
Line Driver, 100 Ohm, 5 V	1 - 5	

4.3.11.1 Sinking Driver 1 A

This section presents the circuit schematic and the characteristics for the 1 A sinking driver. See Section 5.4.3 for more information on protection diodes.

Figure 4-16 Sinking Driver 1 A Circuit Schematic



Power On/Reset State: Off, high impendance

Table 4-14. SInking Driver 1 A Characteristics

Symbol	Parameter	Note	Min	Typical	Max	Unit
DC						
V_S	Supply Voltage		0		40	V
V_{ON}	On Voltage	1			0.4	V
I_L	Leakage Current				10	uA
I_O	Output Current				1	A
I_{QH}	High Level Quiescent Current	4		1		uA
I_{QL}	Low Level Quiescent Current	4		17		mA
AC						
t_{ON}	Turn On Time	2		50		nS
t_{OFF}	Turn Off Time	3		14.5		uS

Note 1: At I_O Max — v1.03

Note 2: From V_S Max to 10% of V_S Max at I_O Max with resistive load — cell v6.02

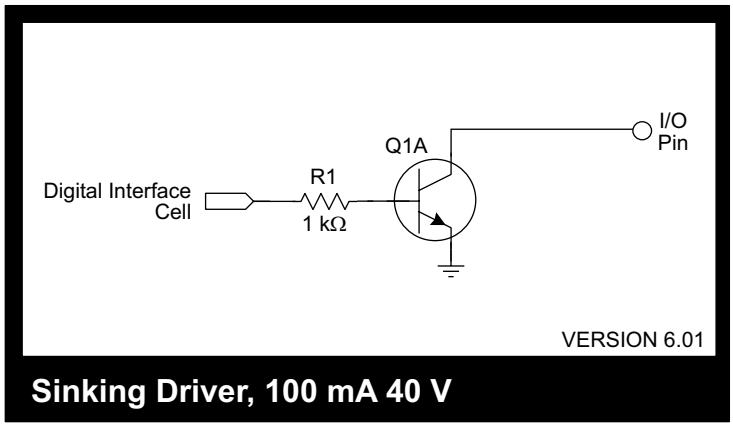
Note 3: From 0 V to 90% of V_S Max at 10 mA with resistive load

Note 4: From PowerCore Supply5

4.3.11.2 Sinking Driver 100 mA

This section presents the circuit schematic and the characteristics for the 100 mA sinking driver. The 100 mA sinking driver does not have the protection diodes that are in the 1 A circuit.

Figure 4-17 Sinking Driver 100 mA Circuit Schematic



Power On/Reset State: Off, high impendance

Table 4-15. Sinking Driver 100 mA Characteristics

Symbol	Parameter	Note	Min	Typical	Max	Unit
DC						
V_S	Supply Voltage		0		40	V
V_{ON}	On Voltage	1			0.4	V
I_L	Leakage Current				10	uA
I_O	Output Current				100	mA
I_{QH}	High Level Quiescent Current	4		1		uA
I_{QL}	Low Level Quiescent Current	4		4		mA
AC						
t_{ON}	Turn On Time	2		46		nS
t_{OFF}	Turn Off Time	3		1.1		uS

Note 1: At I_O Max — v1.02

Note 2: From V_S Max to 10% of V_S Max at I_O Max with resistive load — cell v6.01

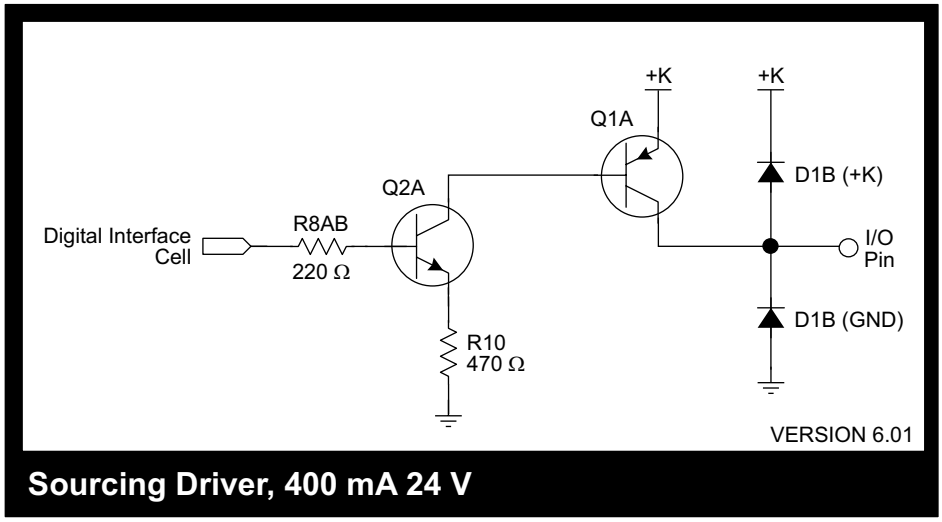
Note 3: From 0 V to 90% of V_S Max at 10 mA with resistive load

Note 4: From PowerCore Supply5

4.3.11.3 Sourcing Driver 400 mA

This section presents the circuit schematic and the characteristics for the 400 mA sourcing driver. Note that it has diode protection. This driver sources from the user-supplied +K pin and can handle up to 24 V.

Figure 4-18 Sourcing Driver 400 mA Circuit Schematic



Power On/Reset State: Off, high impendance

Table 4-16. Sourcing Driver 400 mA Characteristics

Symbol	Parameter	Note	Min	Typical	Max	Unit
DC						
V_S	Supply Voltage		0		24	V
V_{ON}	On Voltage	1	V_S - 0.4			V
I_L	Leakage Current				-100	uA
I_O	Output Current				-0.4	A
I_{QH}	High Level Quiescent Current	4, 5		20		uA
I_{QL}	Low Level Quiescent Current	4		1		uA
AC						
t_{ON}	Turn On Time	2		244		nS
t_{OFF}	Turn Off Time	3		82		uS

Note 1: At I_O Max

Note 2: From 0 V to 90% of V_S Max at I_O Max with resistive load

Note 3: From V_S Max to 10% of V_S Max at 10 mA with resistive load

Note 4: From PowerCore Supply5

Note 5: External +K source supplies additional 10 mA quiescent current

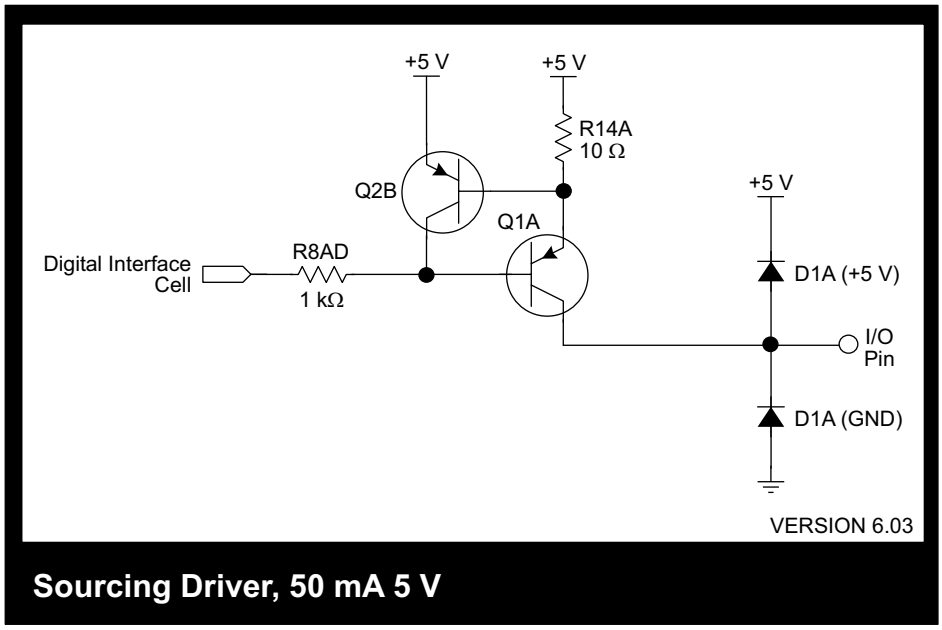
v1.04

cell v6.01

4.3.11.4 Sourcing Driver 50 mA

This section presents the circuit schematic and the characteristics for the 50 mA sourcing driver. Note that it has diode protection. This sourcing driver is short circuit current limited to about 65 mA, making it ideal for vending machine applications requiring a short circuit proof driver.

Figure 4-19 Sourcing Driver 50 mA Circuit Schematic



Power On/Reset State: On, 5 V

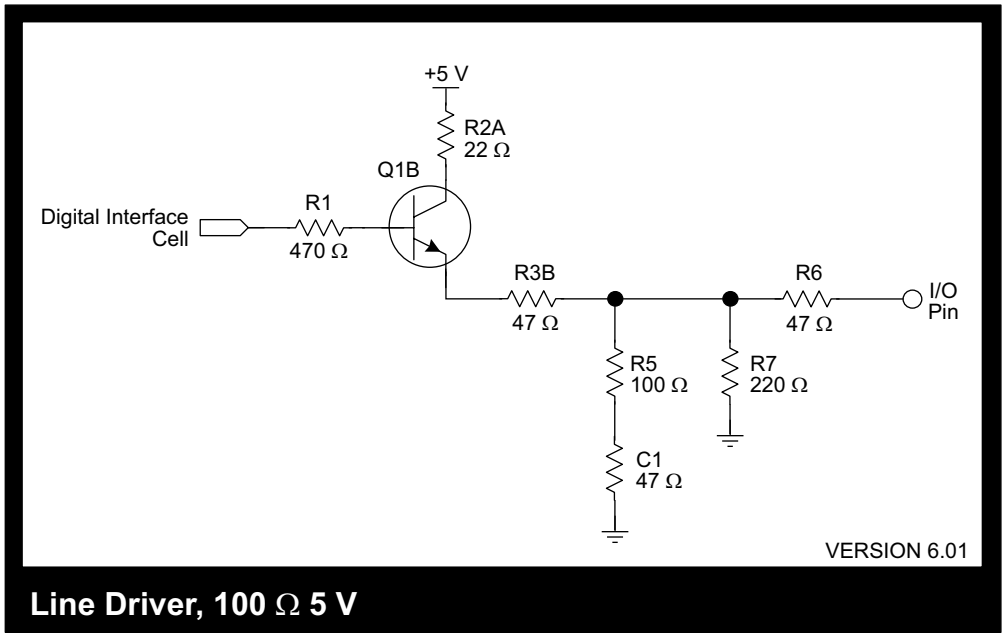
Table 4-17. Sourcing Driver 50 mA Characteristics

Symbol	Parameter	Note	Min	Typical	Max	Unit
DC						
V_S	Supply Voltage	1	4.75	5.0	5.25	V
V_{OH}	High Level Output Voltage	2	4.0			V
I_{OH}	High Level Output Current				-50	mA
I_L	Leakage Current				-10	uA
I_{OLIM}	Output Current Limit	3		-65		mA
I_{QH}	High Level Quiescent Current	1		4		mA
I_{QL}	Low Level Quiescent Current	1		1		uA
AC						
t_{ON}	Turn On Time	4		30		nS
t_{OFF}	Turn Off Time	5		490		nS
Note 1: From PowerCore Supply5						v1.04
Note 2: At I_{OH} Max						cell v6.03
Note 3: Extended time in the current limit mode may damage circuit						
Note 4: From Off to V_{OH} with 100 Ω load						
Note 5: From On to 10% of V_O with 100 Ω load						

4.3.11.5 Line Driver, 100 Ω, 5 V

This section presents the circuit schematic and the characteristics for the line driver. Line drivers are available on J1 through J5, pins 1 through 8.

Figure 4-20 Line Driver 100 Ω Circuit Schematic



Power On/Reset State: Off, 0 V

Table 4-18. Line Driver Characteristics

Symbol	Parameter	Note	Min	Typical	Max	Unit
DC						
V_S	Supply Voltage	1	4.75	5.0	5.25	V
V_{OH}	High Level Output Voltage		2.0			V
V_{OL}	Low Level Output Voltage				0.8	V
I_{OH}	High Level Output Current		-6.7			mA
I_{OL}	Low Level Output Current				5.5	mA
I_{QH}	High Level Quiescent Current	1		31		mA
I_{QL}	Low Level Quiescent Current	1		1		uA
AC						
t_{O1}	Output Time Delay to 1	2		35		nS
t_{O0}	Output Time Delay to 0	3		28		nS

Note 1: From PowerCore Supply5

Note 2: Time delay from a cell input voltage change from logical 0 to 1 to see V_{OH} at the output pin at I_{OH}

Note 3: Time delay from a cell input voltage change from logical 1 to 0 to see V_{OL} at the output pin at I_{OH}

v1.03

cell v6.01

4.3.12 Digital-to-Analog Converters

The RabbitFLEX BL300F can have up to two DAC channels. The analog output from channel #0 comes out on J2, pin 7or can be used to drive the optional speaker output (See Section 4.3.12.3). Analog output channel #1 comes out on J2, pin 8. There are two voltage output range options:

- 0-3 V
- 0-10 V

There are two things to notice in the circuit schematics for the DAC channels (Figure 4-21 and Figure 4-22). First, the 0-10 V DAC channel uses the op amp to increase the PWM voltage to the desired range. Second, the +K voltage is user-supplied and must be at least +12 V, and no more than 30 V when either DAC channel is set to the 0-10 voltage range.

> **NOTE:** The DACs are powered by the same voltage source, which means that a 0-3 V DAC will be powered by +K on J2 if the 0-10 V DAC also exists. If there is no 0-10 V DAC, any 0-3 V DACs will be powered by +5 V.

4.3.12.1 DAC 0-3 V

This section presents the circuit schematic and the characteristics for the 0-3 V DAC.

Figure 4-21 DAC 0-3 V Circuit Schematic

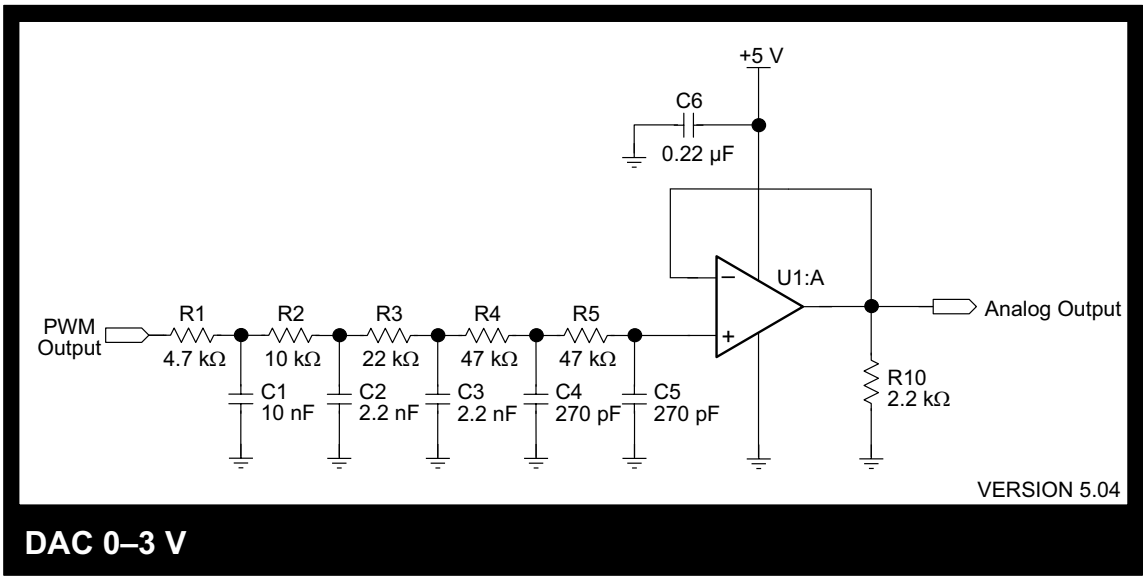


Table 4-19. DAC 0-3 V Characteristics

Symbol	Parameter	Note	Min	Typical	Max	Unit
DC						
V_S	Supply Voltage	1	5		30	V
V_{FS}	Full Scale Output Range			0 to 3		V
I_{OM}	Maximum Output Current	2		-5		mA
RES	Resolution			10		Bits
LIN	Linearity Error	3		±0.5		LSB
MON	Monotonicity			10		Bits
LSB	LSB Value			3.36		mV
AC						
RATE	Conversion Speed	4	900			Samples/S
T_S	Settling Time	5		1.1		mS
BW_{3dB}	3dB Bandwidth			DC to 1.1		kHz

Note 1: Uses Supply5 when 10V DAC not present. Uses the 10V DAC external supply otherwise. v1.02

Note 2: At Maximum V_{FS} cell v5.04

Note 3: Best Fit Method

Note 4: Based on V_{FS} Settling Time. Higher rates possible with reduced voltage changes

Note 5: Full Scale Change to ½ LSB

4.3.12.2 DAC 0-10 V

This section presents the circuit schematic and the characteristics for the 0-10 V DAC.

Figure 4-22 DAC 0-10 V Circuit Schematic

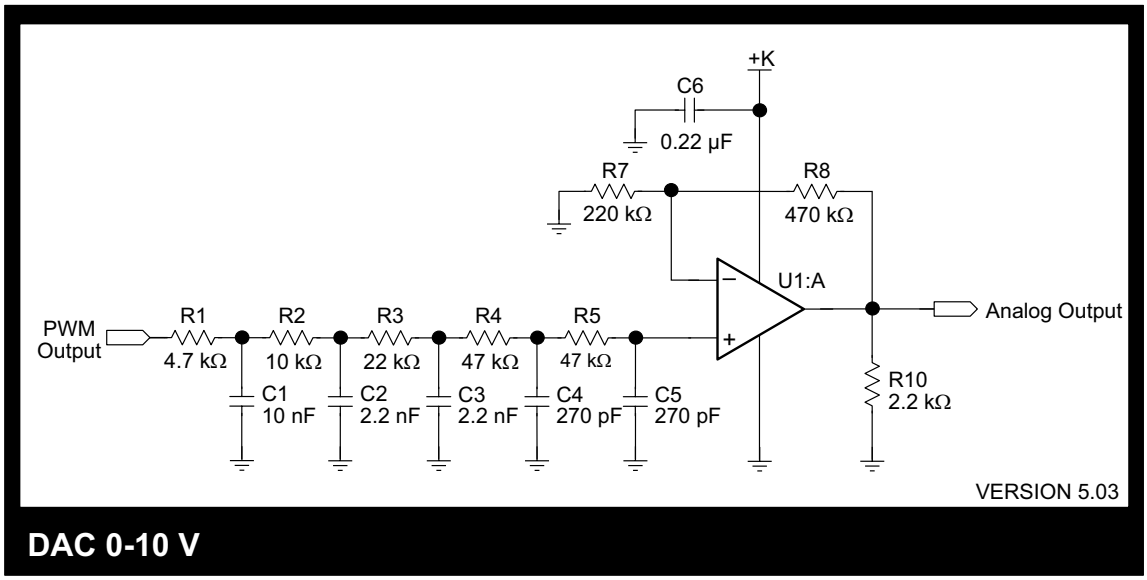


Table 4-20. DAC 0-10 V Characteristics

Symbol	Parameter	Note	Min	Typical	Max	Unit
DC						
V_S	Supply Voltage	1	12		30	V
V_{FS}	Full Scale Output Range			0 to 10		V
I_{OM}	Maximum Output Current	2		-5		mA
RES	Resolution			10		Bits
LIN	Linearity Error	3		±0.5		LSB
MON	Monotonicity			10		Bits
LSB	LSB Value			10.4		mV
AC						
RATE	Conversion Speed	4	900			Samples/S
T_S	Settling Time	5		1.1		mS
BW_{3dB}	3dB Bandwidth			DC to 1.1		kHz

Note 1: Requires external supply — v1.02

Note 2: At Maximum V_{FS} — cell v5.03

Note 3: Best Fit Method

Note 4: Based on V_{FS} Settling Time. Higher rates possible with reduced voltage changes

Note 5: Full Scale Change to ½ LSB

4.3.12.3 Speaker

DAC channel #0 can be configured as an input to a speaker. A pulse width modulated (PWM) signal goes through a low-pass filter, changing the high speed digital signal from the Rabbit processor into a smoothly varying analog signal. This signal is not strong enough to drive the speaker, so it goes through a low voltage audio power amplifier before connecting to the speaker line. Figure 4-23 is a circuit schematic of the filter/amplifier. The output to the speaker is on J6.

Figure 4-23 Loudspeaker Circuit Schematic

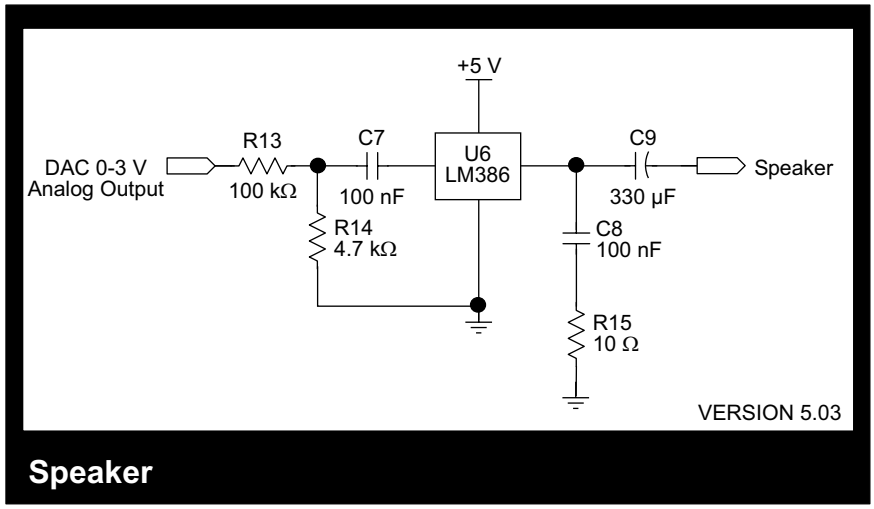


Figure 4-24 Speaker Characteristics

Symbol	Parameter	Note	Min	Typical	Max	Unit
Temperature						
T	Operating Temperature		0		70	°C
DC						
V_S	Supply Voltage		4.75	5.0	5.25	V
I_Q	Quiescent Current	1		4		mA
P_{OUT}	Output Power	2		275		mW
AC						
BW_{3dB}	3dB Bandwidth			.05 - 1.15		kHz
Note 1: From PowerCore Supply5						v1.00
Note2: 8 Ω Load, 10% THD						cell v 5.03

NOTE: The operating temperature range for the speaker differs from the operating temperature range (–40°C to +70°C) of the rest of the RabbitFLEX BL300F components.

A sample WAV files is provided in `/Samples/RabbitFlex_BL300F/audio/` or in `/Samples/RabbitFlex_SBC40/audio/` depending on your version of Dynamic C.

4.3.13 Analog-to-Digital Converters

The RabbitFLEX BL300F has a ramp-compare ADC. Up to 16 channels are available. The analog inputs can be on any combination of pins 1 through 8 on both J3 and J4. The design of the ADCs allow you to choose a current or voltage input on these pins. The measurement range options are:

- 0-3 V
- 0-10 V
- 4-20 mA

The ADCs work with single-ended unipolar voltage inputs and industry standard 4-20 mA current loops.

The digital result has an 8-bit resolution and the sampling frequency is determined by the following equation.

$$\text{Sampling Frequency for ADCs} = \frac{1000 \text{ ms}}{2.35 \text{ ms} \times \#\text{ A/D channels}}$$

The ADC channels are sampled round-robin, so the sampling frequency depends on the number of analog-to-digital channels that exist on the RabbitFLEX BL300F. If there is only one ADC, the sampling frequency is about 425 samples per second. If the maximum of 16 ADCs exist, the sampling frequency is about 25 samples per second.

The ADCs are not meant for applications that require very fast data acquisition rates. They are suitable for measuring analog voltages that change slowly.

4.3.13.1 Analog Input 0-3 V

This section presents the circuit schematic and the characteristics for the 0-3 V analog input.

Figure 4-25 Analog Input 0-3 V Threshold Circuit

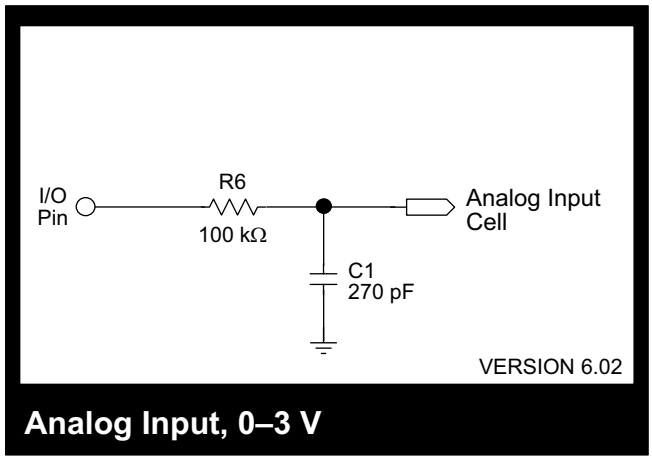


Table 4-21. ADC 0-3 V Characteristics

Symbol	Parameter	Note	Min	Typical	Max	Unit
DC						
V_{IW}	Working Input Voltage			0 to 3		V
V_{IM}	Maximum Input Voltage		-40		40	V
I_I	Input Current	1	-0.5		0.25	mA
R_I	Input Resistance	2		10		MΩ
RES	Resolution			9	12	Bits
LIN	Linearity Error	3, 4		±1.25		LSB
MON	Monotonicity			9		Bits
REP	Repeatability	3		±1		LSB
LSB	LSB Value	3		6		mV
AC						
RATE	Conversion Speed	5		425		Samples/S
T_S	Settling Time	6		200		uS
BW_{3dB}	3dB Bandwidth			DC to 5.2		kHz
Note 1: Over Maximum Input Voltage range						v1.04
Note 2: Over Working Input Voltage range						cell v6.02
Note 3: At stated typical resolution						
Note 4: Best Fit Method						
Note 5: This is the aggregate sampling rate. The sampling rate per channel is (RATE / # Channels)						
Note 6: Full Scale Change to ½ LSB						

4.3.13.2 Analog Input 0-10 V

This section presents the circuit schematic and the characteristics for the 0-10 V analog input.

Figure 4-26 Analog Input 0-10 V Circuit Schematic

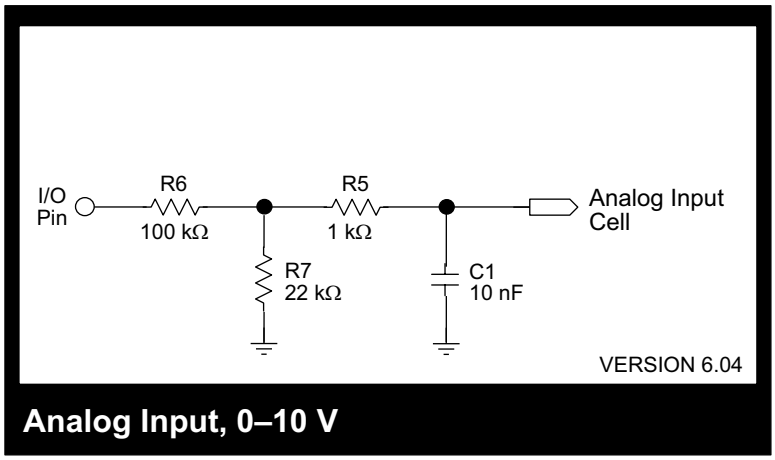


Table 4-22. ADC 0-10 V Characteristics

Symbol	Parameter	Note	Min	Typical	Max	Unit
DC						
V_{IW}	Working Input Voltage			0 to 10		V
V_{IM}	Maximum Input Voltage		-40		40	V
I_I	Input Current	1	-0.5		0.5	mA
R_I	Input Resistance	2		122		KΩ
RES	Resolution			9	12	Bits
LIN	Linearity Error	3, 4		±1		LSB
MON	Monotonicity			9		Bits
REP	Repeatability	3		±1		LSB
LSB	LSB Value	3		33		mV
AC						
RATE	Conversion Speed	5		425		Samples/S
T_S	Settling Time	6		1.1		mS
BW_{3dB}	3dB Bandwidth			DC to 900		Hz
Note 1: Over Maximum Input Voltage range						v1.04
Note 2: Over Working Input Voltage range						cell v6.04
Note 3: At stated typical resolution						
Note 4: Best Fit Method						
Note 5: This is the aggregate sampling rate. The sampling rate per channel is (RATE / # Channels)						
Note 6: Full Scale Change to ½ LSB						

4.3.13.3 Analog Input 4-20 mA

This section presents the circuit schematic and the characteristics for the 4-20 mA analog input.

This analog input provides a simple interface for industry standard 4-20 mA devices. It is a passive input and the device must either supply the loop current or be wired in series with a supply to operate correctly. The loop is formed by using the input pin as the SIG+ line and the ground pin as the SIG- or signal return.

A design consideration for choosing the appropriate ADC circuit for your application is the fact that current is preferred over voltage in noisy industrial/control environments where the transmission of low-amplitude, low-frequency signals must travel significant distances. This is because the current stays constant over the length of the cable.

Figure 4-27 Analog Input 4-20 mA Circuit Schematic

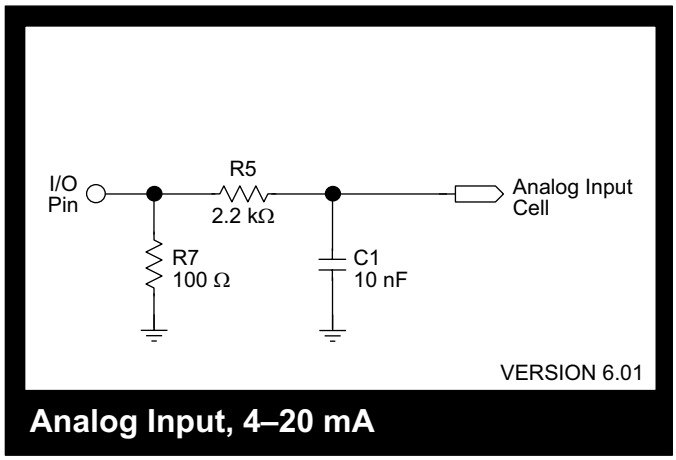


Table 4-23. ADC 4-20 mA Characteristics

Symbol	Parameter	Note	Min	Typical	Max	Unit
DC						
I_{IW}	Working Input Current			4 to 20		mA
I_{IM}	Maximum Input Current		-27.5		27.5	mA
R_I	Input Resistance			100		Ω
RES	Resolution			9	12	Bits
LIN	Linearity Error	1, 2		±1.25		LSB
MON	Monotonicity			9		Bits
REP	Repeatability	1		±1		LSB
LSB	LSB Value	1		60		uA
AC						
RATE	Conversion Speed	3		425		Samples/S
T_S	Settling Time	4		125		uS
BW_{3dB}	3dB Bandwidth			DC to 7.6		kHz
Note 1: At stated typical resolution						v1.02
Note 2: Best Fit Method						cell v6.01
Note 3: This is the aggregate sampling rate. The sampling rate per channel is (RATE / # Channels)						
Note 4: Full Scale Change to ½ LSB						

4.3.14 Power Routing

You can bring power supply pins out on any of the pin-configurable connectors (J1-J5). All five connectors accommodate 5 V pins, but only J1-J3 have 3.45 V. Keep in mind that no matter how many power supply pins are configured, there is a limit to their simultaneous use. The power budget for your application must stay within the parameters of the total power for the system.

Table 4-24. 5 V Power Supply Characteristics

Symbol	Parameter	Note		Min	Typical	Max	Unit
DC							
V_S	Supply Voltage			4.75	5.0	5.25	V
I_{S5}	Supply Current	1, 2	3	1.05		2 - I_Q - I_{S345}	A
			4	0.05		1 - I_Q - I_{S345}	
Note 1: This current is shared by all Supply5 cells and depends on the current used by any SUPPLY345 cells.							v1.04
Note 2: I_Q = Current consumed by the PowerCore at +3.45 Volts, typically between 150 and 400 mA.							cell v 6.01
Note 3: With PowerCore 3800. Please see the PowerCore User's manual for more information.							
Note 4: With PowerCore 3810. Please see the PowerCore User's manual for more information.							

Table 4-25. 3.45 V Power Supply Characteristics

Symbol	Parameter	Note	Min	Typical	Max	Unit
DC						
V_S	Supply Voltage		3.30	3.45	3.55	V
I_{S345}	Supply Current	1, 2	0		0.55	A
Note 1: This current is shared by all Supply345 cells.						v1.03
Note 2: This is dependent upon the installed PowerCore Module configuration and the available current from the 5V supply. Please see the PowerCore User's manual for more information.						cell v6.01

5. MORE INFORMATION ON RABBITFLEX BL300F OPTIONS

This chapter provides links, references and additional information about the RabbitFLEX BL300F options. All Rabbit documentation can be accessed online at rabbit.com or from the Dynamic C CD that came in the RabbitFLEX Tool Kit.

5.1 PowerCore Module Options

For more information on the core module options, see the following manuals:

- *PowerCore FLEX User's Manual* - Describes the PowerCore 3800 and PowerCore 3810
- *Rabbit 3000 Microprocessor User's Manual* - Describes the Rabbit chip

5.1.1 Ethernet

For more information regarding the software necessary for Ethernet communication, see the following manuals:

- *Dynamic C TCP/IP User's Manual, Vols 1 and 2* - Describes the Dynamic C TCP/IP stack and its higher-level protocols
- *An Introduction to TCP/IP* - A beginner's guide to networking

5.1.2 Serial Flash

One of the benefits of having a serial flash is the ability to use the Dynamic C FAT module. The small footprint of this well-defined industry-standard file system makes it ideal for embedded systems. The Dynamic C implementation of the FAT file system has a directory structure that can be accessed with either Unix or DOS style paths. The standard directory structure allows for monitoring, logging, Web browsing, and FTP updates of files.

For more information, see:

- *FAT File System User's Manual* - Describes the FAT structure, operations and functions available.

5.2 Serial Communication

Dynamic C provides drivers for RS-232 and RS-485. For a full discussion of these drivers, see the Technical Note, TN213: "Rabbit Serial Port Software."

5.2.1 Comparison of RS-232 and RS-485

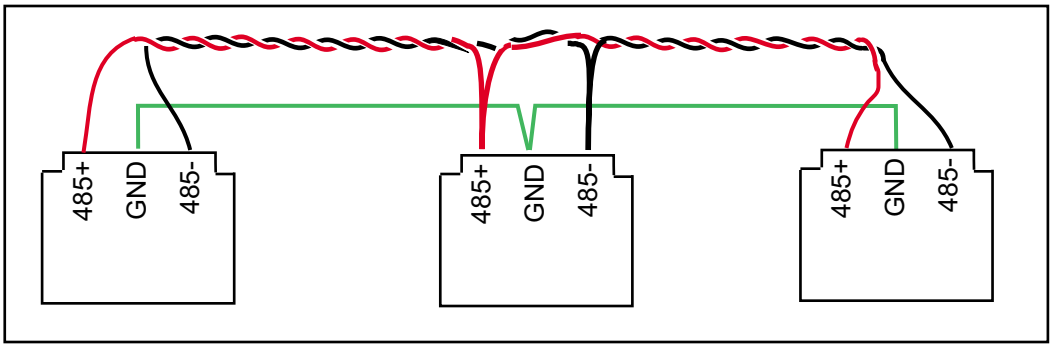
Both RS-232 and RS-485 are popular and well-established serial communication protocols that are used by many different devices. There are a number of considerations when choosing which serial protocols are right for a particular application. The following table looks at both protocols and compares these considerations.

Table 5-3. Serial Communication Parameters

Parameter	RS-232	RS-485
Max number of drivers	1	32
Mode of operation	Full-duplex	Half-duplex
Topology	End-to-end	Multidrop, with node-to-node communication
Max distance	15 m	1200 m
Max slew rate	30 V/µs	n/a

RS-485 was developed to overcome some of the limitations of RS-232, such as the distance between transmitter and receiver. RS-485 also has a more open topology, allowing the RabbitFLEX BL300F to be part of a multidrop network of serially connected devices.

Figure 5.1 Multidrop Network Wiring



5.3.1 Termination Resistors

You can design the RS-485 circuit to include termination resistors or not. A third option is to have selectable termination. For best performance in a multidrop network, termination resistors are enabled only on the end nodes and are disabled on intervening nodes.

At the time of this writing, more information on the termination of transmission lines can be found at:

www.bb-elec.com/tech_articles/rs422_485_app_note

5.4 Digital Outputs

There are several types of options available for digital outputs. There are sinking and sourcing drivers, as well as line drivers. No matter which digital output type you have, the circuit is activated by a call to `flexDigOut()` or `flexDigOutGroup16()`.

5.4.1 Sinking and Sourcing Drivers

One approach to boost the power of DC control signals is to use a sinking output. There are two choices for sinking outputs on a RabbitFLEX BL300F: a sinking output that can sink up to 1 A and one that can sink up to 100 mA. Another advantage of a sinking output driver is that it can control large voltage—up to the rating of the transistor used. The 1 A sinking driver can control up 40 V.

A sinking output uses an NPN transistor. The transistor's *emitter* is connected to ground potential. When on (the control signal, called the *base*, is high), the output terminal, called the *collector*, is connected to ground.

Another type of digital output driver is the sourcing output driver. It is the complement of the sinking output and uses a PNP transistor. The emitter is connected to a positive supply. When the transistor is on (a low voltage on the control signal), the collector is connected to the positive supply.

As with the sinking driver, there are two choices for sourcing outputs on a RabbitFLEX BL300F: a sourcing ouput that can source up to 400 mA and one that can source up to 50 mA.

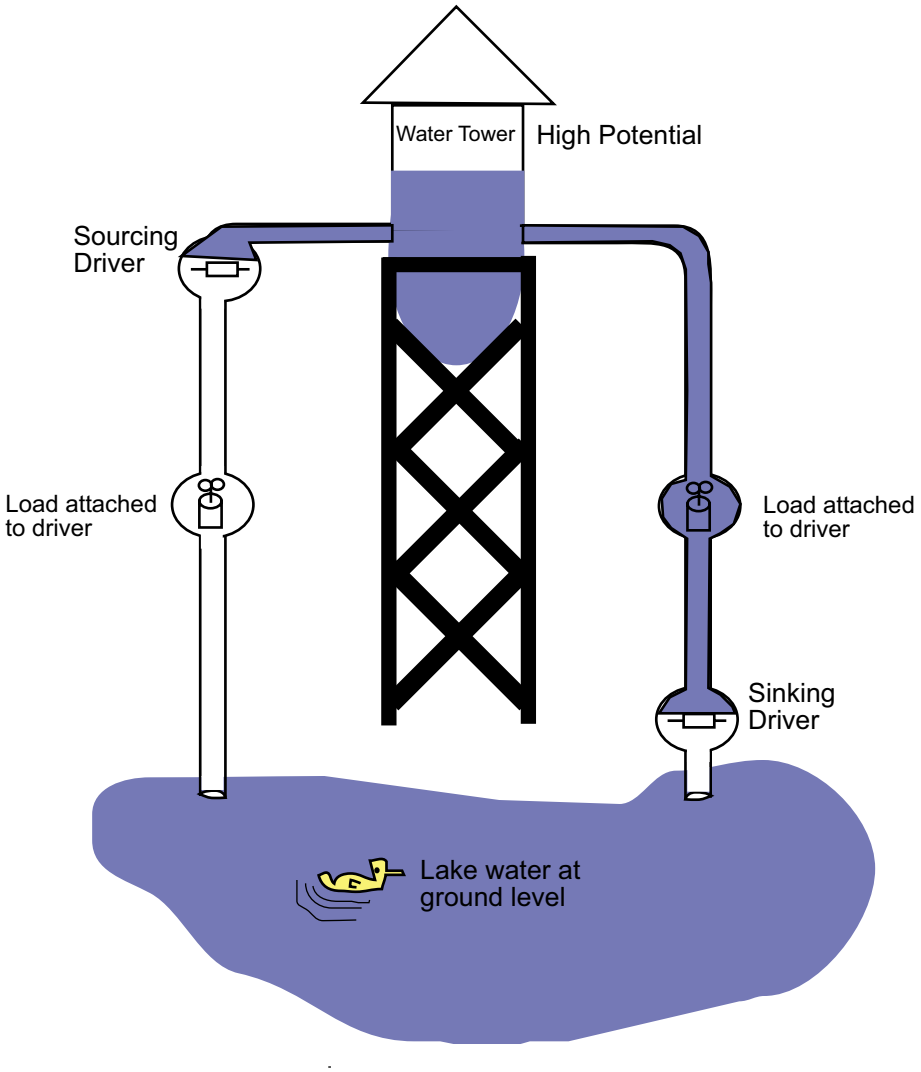
At first, sourcing drivers seem to make a little more sense than sinking drivers, but they are not as efficient as sinking drivers. PNP transistors typically do not have current ratings as high as their NPN counterparts. Sourcing drivers can, however, source current to devices that are connected to negative power supplies.

One way to "see" the difference between sourcing and sinking power is to expand on the water analogy that is often used to understand the different terms that describe electricity, such as voltage and current.

Using a water tower, Figure 5.2 illustrates that the transistor type used in the sourcing driver is connected to the power supply (i.e., the water in the tower), thus sourcing power for the attached load when the transistor's control line is activated. The transistor type used in a sinking driver is connected to ground (i.e., the water at ground level) thus causing current to flow in the attached load by sinking the current to ground when the transistor's control line is activated.

When selecting a driver, make sure to consider the type of load you will be driving. If driving an inductive load, such as a relay, solenoid or electric motor, select a driver that has diode protection to prevent burning out the driver transistor. See Section 5.4.3 for more on protection diodes.

Figure 5.2 Water Tower Analogy of Sinking and Sourcing Power



5.4.2 Line Drivers

Line drivers are used to increase current, thereby enhancing transmission reliability.

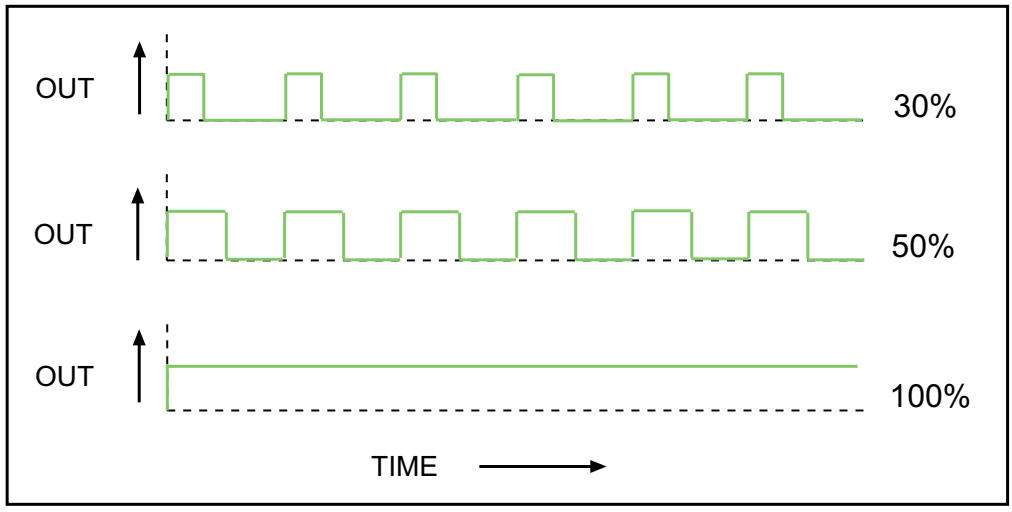
5.4.3 Protection Diodes

A protection diode gives a path for high voltages to follow when driving an inductive load. Just as you cannot stop a freight train instantaneously, you cannot stop current in a magnetic coil instantaneously. When the transistor is turned on, current runs through the transistor, then through the coil. The diode has no current running through it at all because it is back biased (current only runs in the direction of the diode arrow.) For a very short time immediately after the transistor turns off, current is still moving through the coil. It must go somewhere. The protection diode provides a path for that current to go around in a loop until the coil's resistance eventually stops the current (typically takes less than 5 ms). A very high voltage would develop at the transistor's collector without the protection diode, which could zap and destroy the transistor. Typical inductive loades include the magnetic coils in relays, soleniods and electric motors.

5.5 DACs

To produce an analog signal for the DAC channel outputs, the Rabbit generates precisely timed pulses using PWM (pulse width modulation). The digital signal, which is either 0 V or 5 V, is a train of pulses. This means that if the signal is taken to be usually at 0 V (or ground), there will be 5 V pulses. The voltage will be 0 V for a given time, then jump to 5 V for a given time, then back to ground for a given time, then back to 5 V, and so on. The digital signal is a value between 0 and 1024. This value is used to set the duty cycle of the PWM channel by passing it to the function `pwm_set()`.

Figure 5.3 Duty Cycle Examples



Thus, the software only needs to vary the time the signal is at 5 V with respect to the time the signal is at 0 V to achieve any desired voltage between 0 V and 5 V.

The quality of the analog signal is determined largely by the settling time, the slew rate and the output resolution of the DAC channel.

- Settling Time – This is the time it takes for the analog signal to achieve a full-scale change in voltage.
- Slew Rate – This is the maximum rate at which the DAC channel can change the voltage level of the analog signal.
- Output Resolution – This is the number of bits in the digital input that results in the analog output.

5.6 Speaker

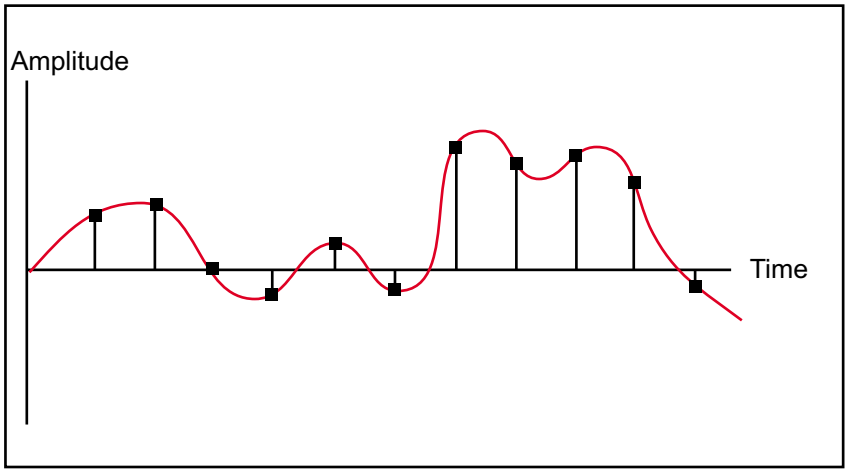
DAC Channel #0 can be configured as an input to a speaker. The audio hardware on the RabbitFLEX board has two sections, the filter and the amplifier. The first part, the filter, converts the high speed digital pulses that come from the Rabbit processor into a smoothly varying analog signal. It does this through *low pass filtering*. According to its name, low pass filtering allows low frequency signals to pass through it while blocking higher frequency signals. For a complex signal like digital pulses, this has the effect of averaging them out to produce an analog signal. In this way, the low pass filter converts a pulse width modulated (PWM) signal into a variable voltage signal. The filter is composed of resistors and capacitors arranged so that the high frequency signals are bypassed to ground through the capacitors.

After the audio signal is filtered it must be amplified in order to be powerful enough to drive a speaker. This is done with an LM386 audio amplifier IC. The signal from the filter must be amplified because it is a *high impedance* signal. Simply put, a high impedance signal is one that is degraded if it is used to directly power an electrical load. So instead of directly connecting the filter output to the speaker, it is connected to the amplifier input. The amplifier input senses the signal from the output and generates a matching low impedance signal that has enough power to drive the speaker effectively.

When dealing with digital to analog conversion and WAV files (a popular standard for digitized audio information) one of the most important questions is does it sound like the original analog signal that was used to create the WAV file?

The answer depends on two things: the sampling rate and the sampling precision used when creating the digital audio file. Digital audio samples are simply a sequence of values, each value representing the amplitude of an audio signal at a given point in time.

Figure 5.4 Sampling of a Sound Waveform



The sampling rate is the number of samples taken in one second. Looking at the figure above, you can see that if you connect the dots from left to right with straight lines, you lose a lot of the information represented by the curved line. By increasing the sampling rate, the closer the straight lines match the curved line, meaning less information is lost.

The sampling precision is the number of gradations possible for each sample point. Each time a sample is taken, the sampling precision determines the digital value that is assigned. For example, if the precision is 10 the software can assign a sample one of 10 discrete numbers. By increasing the sampling precision, just like with the sampling rate, you can make the sample points more closely resemble the curved line of the analog signal.

As of the date of this manual, a really good description of sampling rate and precision (among other things) can be found at:

www.howstuffworks.com/analog-digital3.htm

5.7 ADCs

To measure the analog input signal, the ramp-compare ADC circuits uses the ramp generator on the PowerCore module. The ramp generator produces a saw-tooth signal that ramps up, then quickly falls to zero. The input capture timer starts counting when the ramping up starts. When the ramp voltage matches the analog input signal, a comparator fires, and the timer's value is recorded.

The RAMP_OUT pin is the output from the core module's ramp generator.

The calibration of the ramp is tied to an onboard 2.5 V voltage reference. The 400 Hz ramp has a linear rise time from 0 to 3.1 V of approximately 1.9 ms, and ramps down in approximately 0.45 ms. (The ramp actually starts at a slightly negative voltage of approximately -0.05 V.) The ramp output has a linearity of about 0.1%.

For more information on the ramp generator, see the *PowerCore FLEX User's Manual*.

6. APPLICATIONS PROGRAMMING

Programming your customized RabbitFlex board has been made both adaptable and as simple as possible. The complexity of dealing with multiple hardware configurations has been hidden in the use of data structures in the Flex libraries. A number of sample programs are provided to illustrate the new data structures and supporting software as well as to use for templates for your own application development.

In Section 6.4 a software walk-through is provided for the sample program `speakertone.c` with modifications to increase its functionality.

6.1 RabbitFLEX Sample Programs

This section summarizes sample programs specific to the RabbitFLEX BL300F. They are located in the `/Samples/RabbitFlex_BL300F` folder (formerly named `/Samples/RabbitFlex_SBC40`) where you installed Dynamic C.

Table 6-1. RabbitFLEX BL300F Sample Programs

Program Name	Description
anain_avg_simple.c	Continuously samples one user-specified analog input pin, displaying in the Stdio window both the raw data from the analog signal and also "averaged" raw data, meaning that noise and error are removed using a weighted value. Both the raw data and the averaged raw data are converted to volts. These values are also displayed.
anain_calibrate.c	Displays all analog inputs, prompting you to choose one of them to calibrate, along with its low and high voltages. This sample overwrites the factory calibration.
anain_simple.c	Continuously reads user-specified analog input pin, displaying both its raw value and its converted voltage value.
anain_4_20_simple.c	Continuously reads a user-specified 4-20 mA analog input pin and displays both its raw value and its converted amperage value.
anaout_calibrate.c	Calibrate analog out pins. Requires you to measure output voltages. This sample overwrites the factory calibration.
anaout_simple.c	Accepts a raw value or voltage from the user and outputs it to the analog out pin selected at compile time.
digin_simple.c	Display the value of a digital input.
digin_groups.c	Demonstrates software grouping capability. Displays all digital inputs, then prompts for which inputs to group together.
digout_simple.c	Display the value of a digital output.

Table 6-1. RabbitFLEX BL300F Sample Programs

Program Name	Description
digout_groups.c	Demonstrates software grouping capability. Displays all digital outputs, then prompts for which inputs to group together. After the group is created, each successive digital output channel is set to "1" while the others are set to "0."
keypad.c	You must have a keypad configured for your RabbitFLEX board to run this program. It will automatically configure the keys on your keypad to display to Stdio when pressed. It will also allow you to set whether the keys should repeat when held down
lcd.c	You must have an LCD display configured for your RabbitFLEX board to run this program. It demonstrates how to use each of the RabbitFLEX LCD functions. First, select the desired command from the menu. Then, if necessary, enter the parameter (a character, a string, etc.) for the selected command.
master.c	You must have the RS485 option installed to use this sample program. This program demonstrates a simple RS485 transmission of lower case letters to a slave controller. The slave will send converted upper case letters back to the master controller to be displayed in the Stdio window. Use `slave.c` to program the slave controller.
parity.c	Demonstrates the use of parity checking in the RS232 driver. You must have 2 available serial ports to use this driver.
RabbitFLEX_BL300F_Board_Options.c (a.k.a., RabbitFLEX_SBC40_Board_Options.c)	This program is intended to support targetless compilation. It displays a list of macro definitions to the Stdio window that must be used for targetless compilation to the attached board.
serial_number.c	Displays design number and serial number (unique for each unit) of the RabbitFLEX BL300F.
simple3wire.c	Demonstrates the use of a 3-wire serial port. You must have 1 available serial port to use this driver.
simple5wire.c	Demonstrates the use of a 5-wire (hardware handshaking) serial port. You must have 2 serial ports available to use this driver—one port will be used for the hardware handshaking lines.
sflash_pattern_inspect.c	This program demonstrates the use of a serial flash by writing a pattern to the first 100 pages which then can be inspected or cleared by the user.
sflash_test.c	Tests that an installed serial flash device is operating correctly.
slave.c	You must have the RS485 option installed to use this sample program. This program demonstrates a simple RS485 transmission of characters to a master controller. The slave will send converted upper case letters back to the master controller to be displayed in the Stdio window. Use `master.c` to program the master controller.

Table 6-1. RabbitFLEX BL300F Sample Programs

Program Name	Description
speaker_audio.c	You must have a loudspeaker configured for your RabbitFLEX board to run this program. It demonstrates the use of the RabbitFLEX audio driver. This program loads a .wav file and plays it on an attached loudspeaker.
speaker_tone.c	You must have a loudspeaker configured for your RabbitFLEX board to run this program. It demonstrates the use of the RabbitFLEX tone driver. It allows you to select a tone to play from a menu. The tone can be played for a brief period of time, or indefinitely.
thermistor.c	Continuously reads the onboard thermistor and displays the calculated temperature, an integer value (0-4095) corresponding to the voltage on the analog input channel, and a float value (0-3 V) corresponding to the voltage on the analog input channel. You can apply heat or cold to the area around the thermistor and see the change it makes in the displayed values.
thermOffset.c	This sample allows you to calibrate the onboard thermistor.

6.2 RabbitFLEX BL300F Files

There are two files that you must download from the RabbitFLEX Configurator after you have ordered a board. One is a library file that is necessary for using the board. The other one is a design file that you are responsible for safeguarding. Both will be described here.

6.2.1 RabbitFLEX BL300F Libraries

The general library that will be used by all RabbitFLEX BL300F board configurations is `RabbitFlex_BL300F.lib` (formerly named `RabbitFlex_SBC40.lib`.) It is located in the `lib/RabbitFlex_BL300F` folder (formerly named the `lib/RabbitFlex_SBC40` folder) where you installed Dynamic C.

The library that is specific to your board design must be downloaded from the website using the Rabbit-FLEX Configurator. This design-specific generated library is named FAxxxxxx.lib, where FAxxxxxx is the board's design number. The design number is assigned automatically.

Once this design-specific generated library has been downloaded, it must be placed in the Dynamic C "\lib" folder. A good location for this is in "\lib\RabbitFLEX_BL300F" or "\lib\RabbitFLEX_SBC40," depending on your version of Dynamic C. Note that if you edit the `LIB.DIR` file, it is possible to place the design-specific library in any location—see the *Dynamic C User's Manual* for more information. Although you have flexibility on where to place the design-specific library, you must not change the name of the library. Dynamic C queries your attached RabbitFLEX board to determine its design number, and then uses that to automatically include the correct design-specific library.

6.2.2 RabbitFLEX BL300F Design File

The design number is also used as the name of an .xml file and an html file, both of which summarize the board's connector and pin assignments. The html file is provided as a convenient way for you to review the

connector/pin assignments. The .xml file is more important. It is used when you want to reorder a particular design. A board design cannot be ordered without its .xml file. Although your design file may still be available through your RabbitFLEX account, **it is your responsibility to maintain a copy of the design .xml file**.

6.3 RabbitFLEX BL300F Software Concepts

There are several concepts that must be understood to program the RabbitFLEX BL300F. This section will describe these concepts.

6.3.1 Board Initialization

To use much of your RabbitFLEX BL300F I/O functionality, you must first call the `brdInit()` function. This must be called before you call any other RabbitFLEX functions.

6.3.2 Software Pin Names

Because the RabbitFLEX BL300F has many configurable I/O pins, we need some mechanism for referring to these pins within the software. To accomplish this, each of the pins can be given a software name at design time (within the RabbitFLEX Configurator). All software names will begin with "flex_". This software name is actually the name of a C data structure that represents all information that the system needs to know about that particular I/O pin. Note that each pin will be given a software name whether you provide one or not—your design summary file will display your software names.

Many RabbitFLEX functions accept a `Flex_IOPin*` as the first parameter. For example, look at the prototype declaration for the `flexDigIn()` function:

```
int flexDigOut(Flex_IOPin *pin, int state);
```

So, if you have a digital output with a software name of "flex_led1", then this will correspond to a `Flex_IOPin` data structure named "flex_led1". To change the state of this digital output, you might do the following:

```
flexDigOut(&flex_led1, 1);
```

Note that you must precede the software name with an '&' character—this passes a pointer to the data structure to the `flexDigOut()` function. More explanation of software names and how to use them is given in the software walk-through section later in this chapter.

The LCD, keypad, and speaker also have software names, but they are predefined and not changeable by the user. These devices, if they exist on your RabbitFLEX BL300F design, must be referenced as follows:

Device	Software Name
LCD	flex_lcd
Keypad	flex_keypad
Speaker	flex_speaker

So, to get a keypress from your keypad, you would do the following:

```
ch = flexKeyGet(&flex_keypad);
```

If you would like to change your pin software names after you have designed your board, then it is possible to modify your design-specific generated library. You must search the library for all references to the name that you wish to change. Note, however, that if you do modify your design-specific library, you will be responsible for maintaining that library. The version in your RabbitFLEX account will no longer be synchronized with your modified version. A better way of renaming may be to use a macro define, like this:

```
#define flex_greenled flex_c1p3
```

6.3.3 Pin Groups

One useful feature in the digital input and output software support is the concept of pin groups. A pin group can be used to read a number of inputs or set a number of outputs with one function call.

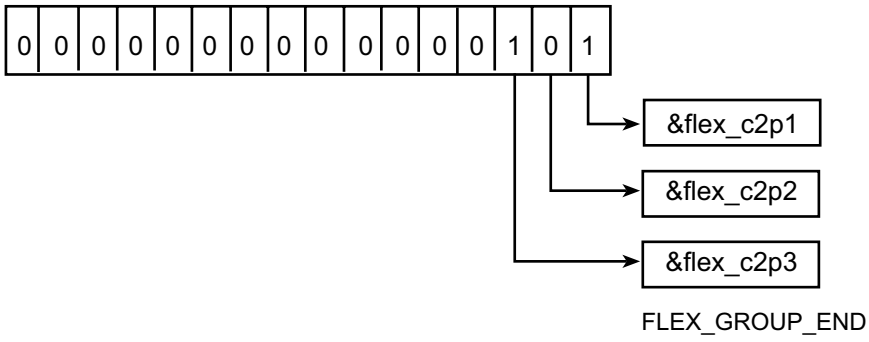
First, the pin group must be defined. A pin group is an array of `Flex_IOPin` pointers (`Flex_IOPin *`). The group must be terminated by an element called `FLEX_GROUP_END`. For example:

```
static const Flex_IOPin *leds[] = {
   &flex_c2p1,
   &flex_c2p2,
   &flex_c2p3,
   FLEX_GROUP_END
};
```

This example creates a pin group of 3 digital outputs. To set these outputs, we could do the following:

```
flexDigOutGroup16(leds, 0x05);
```

The first parameter is the pin group, and the second parameter contains the values with which to set the digital outputs. The least significant bit is written to the first pin in the pin group; the next least significant bit is written to the second pin in the pin group; etc. The following diagram demonstrates this process:



A digital input group works similarly:

```
flexDigInGroup16(switches, &result);
```

In this case, the state of the first pin in the pin group is placed in the least significant bit of result; the state of the second pin is placed in the next least significant bit of the result; etc.

Some automatically generated pin groups are created in your design-specific library. These groups are listed below:

Pin Group	Pin Group Members
_flex_pins_digin[]	All digital inputs
_flex_pins_digout[]	All digital outputs
_flex_pins_diginout[]	All bidirectional pins
_flex_pins_anain[]	All analog inputs
_flex_pins_anaout[]	All analog outputs

Some of these groups are used in sample programs to make them operate on any given RabbitFLEX BL300F design.

6.3.4 Analog Input and Output

Note that your analog inputs and outputs have been factory calibrated. To modify your calibration, use the `flexAnaInCalib()` and `flexAnaOutCalib()` functions. These functions are demonstrated in the `anain_calibrate.c` and `anaout_calibrate.c` sample programs.

To achieve greater accuracy with analog inputs, you can use the analog input averaging functionality. The tradeoff for greater accuracy is that the analog input channels will be less responsive to change. The following formula is used to compute the running averages:

```
avgnew = (1-α)avgold + α * anain
```

The alpha value must be between 0.0 and 1.0, and determines the accuracy/responsiveness tradeoff. A low alpha value will result in higher accuracy but slower responsiveness, whereas a high alpha value will result in lower accuracy but faster responsiveness.

To set the alpha value for a specific analog input, use the `flexAnaInAverageSetting()` function. Then use the `flexAnaInAverage()` and `flexAnaInVoltsAverage()` or `flexAnaInmAmpsAverage()` functions to read the averaged values.

6.3.5 RS-232

Use the `serMode()` function to set up your serial ports before using them. `serMode()` takes a parameter: 0 means to use 3-wire serial ports, and 1 means to use a 5-wire serial port. Note that your design must support your selected serial mode. `serMode()` should be called after your `serXopen()` function(s), where X represents the particular serial port you are using.

6.3.6 Keypad

All keypad functions begin with “flexKey.”

To use the keypad software support, you must first initialize the keypad. Use the `flexKeyInit()` function. Here is an example:

```
flexKeyInit(&flex_keypad);
```

Note that the keypad's software name is always flex_keypad.

Next, you should configure the keys on your keypad. This is what the flexKeyConfig() function does. When a key is pressed, the flexKeyGet() function will get a specific key code before returning which key was pressed. The keycode is calculated as follows:

```
[output #] * [num outputs] + [input #]
```

The flexKeyConfig() function takes the key code as a parameter to determine which key you are configuring. The other parameters let you configure the actual char value that flexKeyGet() will report when the corresponding key is pressed, as well as whether or not a key should repeat if held down.

To drive the keypad driver, you must continuously call the flexKeyProcess() function.

6.3.7 LCD

All LCD functions begin with "flexDisp." To use the LCD software support, you must first initialize the LCD. Use the flexDispInit() function. Here is an example:

```
flexDispInit(&flex_lcd, NULL);
```

Note that the LCD's software name is always flex_lcd.

Also note the second parameter in flexDispInit(). Your LCD display may have different properties from the standard LCDs that we support. The second parameter lets you configure some of the parameters for your attached LCD. This initialization parameter is of type Flex_LCDConf. Here is the structure definition for Flex_LCDConf:

```
typedef struct {
   char num_cols;              // Number of columns on the display
   char num_rows;              // Number of rows on the display
   char row_addresses[4];      // Starting address of each row (up to 4 rows)
   float anaout_contrast_min; // Minimum voltage for contrast control
   float anaout_contrast_max; // Maximum voltage for contrast control
   int pwm_contrast_min;       // Minimum duty cycle for contrast control
   int pwm_contrast_max;       // Maximum duty cycle for contrast control
   char def_contrast;          // Default scaled value for contrast control (0-255)
} Flex_LCDConf;
```

If your display has a specific number of columns and rows, then you can set that with the num_cols and num_rows parameters. The documentation for your LCD module should indicate at which address each row begins—these can be specified with the row_addresses array. The anaout_contrast_min and anaout_contrast_max parameters can be ignored—contrast on the RabbitFLEX BL300F is currently set via a PWM output. pwm_contrast_min and pwm_contrast_max can be used to restrict the range of the contrast settings, whereas def_contrast can be used to set the default contrast value.

So, for a 2x20 display, you might initialize it as follows:

```
Flex_LCDConf my_lcd;

// Number of columns on the display
my_lcd.num_cols = 20;
```

```
// Number of rows on the display
my_lcd.num_rows = 2;

// Starting address of each row (up to 4 rows)
my_lcd.row_addresses[0] = 0x80;
my_lcd.row_addresses[1] = 0xc0;

// Minimum duty cycle for contrast control
my_lcd.pwm_contrast_min = 0;

// Maximum duty cycle for contrast control
my_lcd.pwm_contrast_max = 1024;

// Default scaled value for contrast control (0-255)
my_lcd.def_contrast = 0;
```

Note that the RabbitFLEX BL300F LCD driver is intended to support modules based on the Hitachi HD44780 chipset, but we cannot guarantee compatibility will all LCD modules based on this chipset. To ensure compatibility, use the LCD module that we provide for the RabbitFLEX BL300F.

After the display has been initialized, there are a number of functions to display text, manipulate the cursor, change the contrast, turn the backlight on and off, etc. See the `flexDisp*()` functions for more information.

6.3.8 Speaker

The loudspeaker option can be used with two drivers: the tone driver and the audio driver. The tone driver is intended to play a repeated tone for a specific number of repeats (or indefinitely). The audio driver is intended to play a single audio sample. Tone driver functions begin with flexTone, and audio driver functions begin with flexAudio.

To begin using the tone or audio drivers, you must first activate them with `flexToneActivate()` or `flexAudioActivate()`, respectively. When activated, the drivers will consume about 10-20% of your CPU. Thus, after you have played a tone or sample, you may want to shutdown the driver(s) with `flexToneShutdown()` or `flexAudioShutdown()`.

See the `speaker_tone.c` and `speaker_audio.c` for examples on using these drivers.

6.3.9 Thermistor

This is a PowerCoreFLEX feature—please see the *PowerCoreFLEX User's Manual* for information on using the thermistor.

6.3.10 Serial Flash

This is a PowerCoreFLEX feature—please see the *PowerCoreFLEX User's Manual* for information on using the serial flash.

6.4 Software Walk-Through

In this section, we will begin with the speaker_tone.c sample program and modify it to demonstrate more RabbitFLEX functionality. Note that to make the modifications and run the sample program, you will need the speaker option, appropriate digital inputs and outputs, and the demo board that is included with the RabbitFLEX tool kit. However, even if you do not have these items, you should still be able to read along to gain more understanding of how to use the RabbitFLEX software.

6.4.1 Studying speaker_tone.c

The speaker_tone.c sample program demonstrates how to use the tone driver. For the purposes of RabbitFLEX, it is not necessary to understand all of the details of this sample program, but essentially it builds waveforms at different frequencies to be used with the tone driver. The refwave_organ() and refwave_saw() functions build the waveforms; the buildwave() function adapts the organ and saw waveforms to different frequencies.

In the following walk-through, we will only be making changes to the main() function, so it is probably worth studying main() to understand it.

```
void main(void)
{
   char ch;                  // Holds the user command
   char repeat;              // Indicates whether or not the tone should repeat
   int request_exit;         // Indicates if user has requested to exit the program
```

The above section of code simply declares some variables for later use in main().

```
   // This function must be called to initialize the RabbitFLEX board
   brdInit();
```

The brdInit() function must be called at the beginning of every RabbitFLEX BL300F program. It sets up internal data structures, initializes analog capabilities, sets up CPU registers appropriately, etc. If this function is not called, then much of the RabbitFLEX I/O capability will be unavailable.

```
   // This function sets the correct PWM channel to use for the speaker.
   // Simply pass a pointer to the speaker structure.
   flexSpeakerPWM(&MY_SPEAKER);
```

The above function is used to set the PWM (Pulse Width Modulation) channel that the speaker uses. This introduces the idea of using data structures to pass information into the RabbitFLEX functions. MY_SPEAKER is a macro that has been defined at the top of the program to flex_speaker. So, without the macro definition, this line would be:

```
flexSpeakerPWM(&flex_speaker);
```

flex_speaker is a data structure of type Flex_IOPin. These data structures contain all of the information that the RabbitFLEX functions need to know about the given pin. In this case, the structure contains the correct PWM channel. Note that instead of passing the data structure itself, we pass a pointer to the data structure. We will see this again and again in other RabbitFLEX functions.

```
// Activate the tone system
flexToneActivate();
```

This function simply activiates the tone driver. Once enabled, it consumes about 10-20% of the system's processor time while running. Therefore, in a real program, you might want to activate and shutdown (using `flexToneShutdown()`) the tone driver when needed.

The next section of code simply builds the waveforms for the five different tones and then displays a menu of choices to the user. We will skip down to the next RabbitFLEX-related section of code.

```
// Continue until the user requests to exit the program
while (!request_exit) {
   ch = getchar();
   // Perform the user command
   switch (ch) {
   case '0':
      // Play the raspy tone
      flexToneLoad(tones[0], tone_len[0], repeat);
      break;
```

This section of code begins with a while loop that checks for user input. Note in particular the `flexToneLoad()` call—this loads the given tone and begins playing it. The "repeat" parameter is used to indicate how many times the tone will be repeated (≥128 means to repeat indefinitely). Following the above section are a number of other "case" statements that load other tones and implement other commands. We will skip them.

```
// Turns off the tone system. This frees up the processor for other
// tasks, but in this case, we're just exiting the program.
flexToneShutdown();
```

Finally, after the user has chosen to exit the program, the `flexToneShutdown()` function is called. This disables the tone driver and frees the processor for handling other tasks.

6.4.2 Extending speaker_tone.c

`speaker_tone.c` shows how to use the tone driver, but can we combine it with some of the other RabbitFLEX I/O functionality? In this section, we will add the ability to select the tone to play via a digital input switch, and light up a corresponding digital output LED when we select a tone.

To implement this sample, we need appropriate digital inputs and outputs. We will use the demo board from the RabbitFLEX toolkit. We will use each of the four switches and LEDs. We will connect the switches directly to 1.4V TTL Threshold digital inputs, and the LEDs directly to 100mA sinking driver, 40V max digital outputs.

Each of the changes that are shown below are made to the `main()` function of `speaker_tone.c`.

```
Flex_IOPin *switch_pressed;
Flex_IOPin *led;
int switchnum;
```

First of all, at the top of `main()`, we need to add a few variables. The variable `*switch_pressed` will point to the switch that has been pressed. The variable `*led` will point to the LED that we should light up. And the variable `switchnum` indicates the number of the switch (1-4) that was pressed. We will see how these will be used later in this walk-through.

We will now skip farther down over some code that does not need to change.

```
// Display a menu of choices to the user
printf("Press a switch on the demo board to play a sound!\n");
```

The above code should simply replace the menu of options that are presented to the user. Instead of using the Stdio window to select the tone to play, we will be using switches on the demo board.

Again, we will skip down further, to just before the main while() loop.

```
// Initialize the output LEDs to turn off
flexDigOut(&flex_digout32, 1);
flexDigOut(&flex_digout34, 1);
flexDigOut(&flex_digout36, 1);
flexDigOut(&flex_digout38, 1);
```

The lines of code above will turn off the attached LEDs. Note that to turn off the LEDs on the demo board, we must actually turn on the connected digital output. This is because the Flex library takes into account drivers that invert the signal. All `flexDigOut()` calls always keep the logic uninverted. In the case of a sinking driver, when `flexDigOut()` is called with a logic 1, the library turns the sinking transistor off so that the output can go high and the logical 1 is available at the pin. So when using sinking drivers, the transistor is turned on by calling `flexDigOut()` with a logic 0.

`flexDigOut()` is the function that manipulates a single digital output. Note that, like `flexSpeakerPWM()`, this function takes a pointer to a `Flex_IOPin` structure. This first call to `flexDigOut()` manipulates the `flex_digout32` digital output. Note that “flex_digout32” should correspond to how you have named your digital output. When you design your RabbitFLEX board, you have the option of setting your own software name for each pin. If you designed your RabbitFLEX board to have a digital output attach to an LED, then you might want to call that output “flex_led”. It the case above, we used a different naming scheme. “digout” indicates that this is a digital output. “32” indicates that this digital output is located on connector J3, pin 2. Again, when you design your board, you can name your RabbitFLEX pins in whatever way makes the most sense for your application.

The rest of this section will detail the main changes that have been made to this program. In fact, you should delete the entire main `while` loop in speaker_tone.c, since we will be replacing it. For convenience, here is the complete listing for the new `while` loop.

```
while (1) {
   costate {
      switch_pressed = NULL;
      while (!switch_pressed) {
         // Check for a switch pressed. When we detect one, then save off the pin structure
         // for the switch pressed, the pin structure for the corresponding LED, and the number
         // of the switch pressed.
         if (flexDigIn(&flex_digin31)) {
            switch_pressed = &flex_digin31;
            led = &flex_digout32;
            switchnum = 1;
         }

         else if (flexDigIn(&flex_digin33)) {
            switch_pressed = &flex_digin33;
            led = &flex_digout34;
            switchnum = 2;
         }
         else if (flexDigIn(&flex_digin35)) {
            switch_pressed = &flex_digin35;
            led = &flex_digout36;
            switchnum = 3;
         }
         else if (flexDigIn(&flex_digin37)) {
            switch_pressed = &flex_digin37;
            led = &flex_digout38;
            switchnum = 4;
         }
      }
      // Wait 50 ms to make sure the switch stays pressed (debouncing)
      waitfor(DelayMs(50));

      // Check if the switch is still pressed
      if (flexDigIn(switch_pressed) == 1) {
         // Light the corresponding output LED
         flexDigOut(led, 0);
         // Play the tone
         flexToneLoad(tones[switchnum], tone_len[switchnum],
                      REPEAT_TONE);
```

```
            while (1) {
               // Wait for the switch to be released
               waitfor(flexDigIn(switch_pressed) == 0);
               // Wait additional 200 ms
               waitfor(DelayMs(200));
               // If the switch is still released, then break out of the while loop
               if (flexDigIn(switch_pressed) == 0) {
                  // Turn the LED back off
                  flexDigOut(led, 1);
                  break;
               } // end if statement
            } // end while loop
         } // end if statement
      } // end costate
   } // end main while loop
```

Now we will go through each part of the while loop step-by-step.

```
   while (1) {
      costate {
         switch_pressed = NULL;
         while (!switch_pressed) {
```

Note that we are using a costate in this sample. The costate will allow us to more easily delay for implementing switch debouncing. Also note that we set `switch_pressed` to NULL. Recall that `switch_pressed` is of the type `Flex_IOPin*`. It will be used to point to a switch (or digital input) that has been engaged. We initialize this to NULL, and will remain in the loop until a switch has been pressed.

```
            if (flexDigIn(&flex_digin31)) {
               switch_pressed = &flex_digin31;
               led = &flex_digout32;
               switchnum = 1;
            }
```

We have already seen that `flexDigOut()` manipulates digital outputs. Therefore, it is not surprising that `flexDigIn()` manipulates digital inputs. In this case, `flex_digin31` indicates the digital input that the program is reading. Always remember that you must pass a pointer to the RabbitFLEX I/O structures, and not the structure itself.

If `flexDigIn()` indicates that `flex_digin31` has been pressed, then we save off the switch that has been pressed, the LED that we should turn on, and the number of the switch that has been pressed.

```
            else if (flexDigIn(&flex_digin33)) {
               switch_pressed = &flex_digin33;
               led = &flex_digout34;
               switchnum = 2;
            }
```

```
      else if (flexDigIn(&flex_digin35)) {
         switch_pressed = &flex_digin35;
         led = &flex_digout36;
         switchnum = 3;
      }
      else if (flexDigIn(&flex_digin37)) {
         switch_pressed = &flex_digin37;
         led = &flex_digout38;
         switchnum = 4;
      }
   } // end while loop
```

Similarly, we check each of the other three switches.

```
// Wait 50 ms to make sure the switch stays pressed (debouncing)
waitfor(DelayMs(50));
// Check if the switch is still pressed
if (flexDigIn(switch_pressed) == 1) {
```

To make sure that the signal from the switch has settled, we wait 50 ms and recheck the digital input. Note that there is no '&' before `switch_pressed`. This is because `switch_pressed` is already a pointer—it is used to point to which switch has been pressed.

If this debouncing check fails, then we will reach the end of the costatement, which means that the costatement starts over again from the top. Thus, we once again start monitoring all of the switches.

```
// Light the corresponding output LED
flexDigOut(led, 0);
```

We have now accepted the switch press. Recall that `led` is also already a pointer, and thus it is also not preceded by an "&." This line simply turns on the LED to indicate that the switch has been pressed.

```
// Play the tone
flexToneLoad(tones[switchnum], tone_len[switchnum], REPEAT_TONE);
```

Since we have accepted the switch press and turned on the corresponding LED, then we might as well play the tone. We use the switchnum variable that we saved earlier to choose the tone to play.

```
            while (1) {
               // Wait for the switch to be released
               waitfor(flexDigIn(switch_pressed) == 0);
               // Wait additional 200 ms
               waitfor(DelayMs(200));
               // If the switch is still released, then break out of the while loop
               if (flexDigIn(switch_pressed) == 0) {
                  // Turn the LED back off
                  flexDigOut(led, 1);
                  break;
               }
            }
         }
      }
   }
```

This last section implements debouncing on switch release. We wait until the switch is released, and then check the switch again 200 ms later. If the switch is still released, then we turn off the corresponding LED and we break out of the while loop. The costatement starts over again, which means that we again monitor all of the switches.

This concludes the main while() loop. With these changes, we can now use the switches on the demo board to play a tone and light a corresponding LED.

6.4.3 Extending speaker_tone.c with I/O Grouping

In this section, we will make further changes to speaker_tone.c to use RabbitFLEX I/O grouping. This will make the resulting code somewhat more flexible, and will demonstrate how to use I/O groups. First, we need to declare some variables.

```
Flex_IOPin **switch_pressed;
Flex_IOPin **led;
int switchnum;
unsigned int switch_values;
unsigned int value;
```

Note that switch_pressed and led are now double pointers (Flex_IOPin **). This is because we will be using these variables to iterate through a group (or array) of Flex_IOPin* pointers. This will become more apparent as we walk through this sample.

We have also created switch_values and value variables, which we will explain later.

```
static const Flex_IOPin *switches[] = {
   &flex_digin31,
   &flex_digin33,
   &flex_digin35,
   &flex_digin37,
   FLEX_GROUP_END
};
```

```
static const Flex_IOPin *leds[] = {
   &flex_digout32,
   &flex_digout34,
   &flex_digout36,
   &flex_digout38,
   FLEX_GROUP_END
};
```

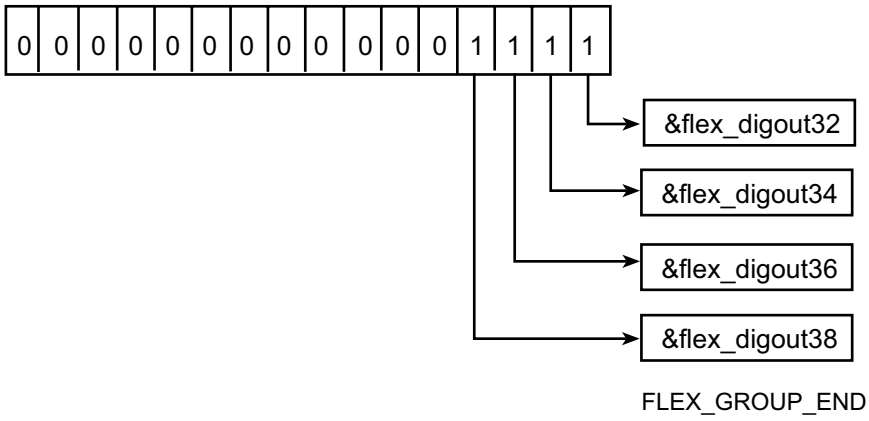
To use I/O groups, we must first create the groups. The `switches[]` and `leds[]` arrays contain pointers to each of the `Flex_IOPin` structures for the switches and the LEDs, respectively. Note that these are arrays of `Flex_IOPin` pointers. Also note that each group is terminated by `FLEX_GROUP_END`; this macro allows the I/O group API functions to know when they have reached the end of the I/O group.

Note the flexibility that I/O groups allow. To change which inputs you want to use, you can simply change the members of the switches I/O group. Once this change has been made, it is not necessary to change any other code in the sample.

We will now skip through the sample to just before the main `while()` loop.

```
// Initialize the output LEDs to turn off
flexDigOutGroup16(leds, 0x0f);
```

Remember how we individually initialized each of the LEDs in the previous section? With the "leds" I/O group, we can initialize the LEDs in a single statement. Both the `flexDigOutGroup16()` and `flexDigInGroup16()` functions expect an array of `Flex_IOPin` pointers as the first argument. The second parameter in `flexDigOutGroup16()` is an unsigned integer. This integer provides the values that will be output to the pins in the group. The least significant bit in the integer provides the value for the first output in the group; the next least significant bit provides the value for the second output in the group, etc:



Next is the main `while` loop. The entire `while` loop will be replaced, so here is the code listing for convenience:

```
while (1) {
   costate {
      // Check if a switch has been pressed
      do {
         flexDigInGroup16(switches, &switch_values);
      } while (switch_values == 0x00);
      // Determine which switch was pressed
      value = 1;
      switch_pressed = switches;
      led = leds;
      switchnum = 1;
      while (!(value | switch_values)) {
         value <<= 1;
         switch_pressed++;
         led++;
         switchnum++;
      }

      // switch_pressed now indicates which pin was pressed
      // Wait 50 ms to make sure the switch stays pressed (debouncing)
      waitfor(DelayMs(50));

      // Check if the switch is still pressed
      if (flexDigIn(*switch_pressed) == 1) {
         // Light the corresponding output LED
         flexDigOut(*led, 0);
         // Play the tone
         flexToneLoad(tones[switchnum], tone_len[switchnum],
                      REPEAT_TONE);
         while (1) {
            // Wait for the switch to be released
            waitfor(flexDigIn(*switch_pressed) == 0);
            // Wait additional 200 ms
            waitfor(DelayMs(200));
            // If the switch is still released, then break out of the while loop
            if (flexDigIn(*switch_pressed) == 0) {
               // Turn the LED back off
               flexDigOut(*led, 1);
               break;
            }
         }
      }
   }
}
```

We will now go through each part of this new while loop in detail:

```
while (1) {
   costate {
      // Check if a switch has been pressed
      do {
         flexDigInGroup16(switches, &switch_values);
      } while (switch_values == 0x00);
```

Like before, we use a costatement to make implementing debouncing easier. Now notice the `flexDigInGroup16()` function call. Like `flexDigOutGroup16()`, this function takes an array of `Flex_IOPin` pointers. However, the second parameter is a pointer to an unsigned integer. `flexDigInGroup16()` places the values read from the digital inputs into this unsigned integer (`switch_values`, in this case). This is done in much the same way that the outputs are used in `flexDigOutGroup16()`. That is, the value read from the first input in the group is placed in the least significant bit in the unsigned integer; the second input value is placed in the next least significant bit; etc.

Note that having a do/while loop in a costatement is not an efficient design for cooperative multitasking, since the costatement would not yield until a switch has been pressed. In a real-world application, busy waiting such as this should be replaced with a `waitfor` statement and a function call to code that would check for the switch press and return.

Notice that we are reading the switch inputs until one of the switches has been pressed. Because we read the values in a group, we can check all switches simultaneously.

```
      // Determine which switch was pressed
      value = 1;
      switch_pressed = switches;
      led = leds;
      switchnum = 1;
      while (!(value | switch_values)) {
         value <<= 1;
         switch_pressed++;
         led++;
         switchnum++;
      }
      // switch_pressed now indicates which pin was pressed
```

Next, we must inspect the `switch_values` unsigned integer to determine which switch was pressed. To do this, we use the `value` variable to check against each bit of `switch_values`. That is, we change the bit that is set within the variable `value` in each iteration of the loop. By ORing the `value` variable against `switch_values`, we can determine if the corresponding switch has been pressed.

Note that as we iterate through the loop, we keep track of the switch, LED, and switch number that we are on. Incrementing the `switch_pressed` and `led` pointers will make them point to the next member in the respective group.

```
        // Wait 50 ms to make sure the switch stays pressed (debouncing)
        waitfor(DelayMs(50));
        // Check if the switch is still pressed
        if (flexDigIn(*switch_pressed) == 1) {
```

Now we begin switch debouncing. We wait 50 ms, and then check if the switch has been pressed again. Note that `switch_pressed` indicates which pin has been pressed—this is one of the results from the loop above. However, `switch_pressed` is a double pointer (`Flex_IOPin **`), whereas `flexDigIn()` expects a single pointer (`Flex_IOPin *`). Therefore, we need to dereference `switch_pressed` in the call to `flexDigIn()`.

```
            // Light the corresponding output LED
            flexDigOut(*led, 0);
            // Play the tone
            flexToneLoad(tones[switchnum], tone_len[switchnum], REPEAT_TONE);

            while (1) {
               // Wait for the switch to be released
               waitfor(flexDigIn(*switch_pressed) == 0);
               // Wait additional 200 ms
               waitfor(DelayMs(200));
               // If the switch is still released, then break out of the while loop
               if (flexDigIn(*switch_pressed) == 0) {
                  // Turn the LED back off
                  flexDigOut(*led, 1);
                  break;
               }
            } // end of while loop
         }
      }
   }
```

At this point, the code here is very similar to the debouncing code in the previous section. The only differences are the references to `switch_pressed` and `led`. Because those variables are now both double pointers, we must dereference them when making calls to `flexDigIn()` and `flexDigOut()`.

This section demonstrated how to use the grouping functionality in the RabbitFLEX API. The techniques described in this chapter should now have given you a running start to implementing your own RabbitFLEX applications.

6.5 API Functions

This section describes the programming interface for the RabbitFLEX BL300F. Table 6-6 lists the functions by category. Each function name is a link to the function's full description.

Table 6-6. API Functions

Function Category	Function Name	
Board Initialization	brdInit	
Pin Names	flexPinName	
Digital Inputs	flexDigIn flexDigOutGroup16	
Digital Outputs	flexDigOut flexDigShadow flexDigOutGroup16	
Analog Inputs	flexAnaInCalib flexAnaIn flexAnaInVolts flexAnaInmAmps flexAnaInAverageSetting	flexAnaInAverage flexAnaInVoltsAverage flexAnaInmAmpsAverage flexAnaInNewValue
Analog Outputs	flexAnaOutCalib flexAnaOut flexAnaOutVolts	
LCD	flexDispInit flexDispCursor flexDispGoto flexDispGetPosition flexDispGetDimensions flexDispClear	flexDispPutc flexDispPrintf flexDispOnoff flexDispBacklight flexDispContrast flexDispGetContrast
Keypad	flexKeyProcess flexKeyInit flexKeyConfig	flexKeyUnget flexKeyGet
Speaker	flexSpeakerPWM flexToneActivate flexToneLoad flexToneStop flexToneShutdown flexAudioActivate	flexAudioPlay flexAudioPlaying flexAudioSetRate flexAudioLoad flexAudioShutdown flexAudioStop
Serial Communication	ser485Tx ser485Rx	serMode

6.6.1 Board Initialization

brdInit

```
void brdInit(void);
```

DESCRIPTION

Call this function at the beginning of your application to initialize the controller's I/O ports. This function sets up all of the I/O available on your RabbitFLEX board, including Ethernet, digital I/O, and analog I/O.

RETURN VALUE

None.

6.6.2 Pin Names

flexPinName

```
char *flexPinName(Flex_IOPin *pin);
```

DESCRIPTION

Returns the character string that represents the name of the given I/O pin. Note that the returned string cannot be modified.

PARAMETER

pin	Pointer to the information structure for the selected I/O pin.

RETURN VALUE

A string representing the name of the selected I/O pin.

6.6.3 Digital Inputs

flexDigIn

```
int flexDigIn(Flex_IOPin *pin);
```

DESCRIPTION

Reads the state of a digital input channel. This function is non-reentrant.

PARAMETER

pin	Pointer to the information structure for the selected I/O pin.

RETURN VALUE

0: logic low
1: logic high
< 0: error (not a valid digital input?)

SEE ALSO

flexDigShadow, flexDigOut, flexDigInGroup16, flexDigOutGroup16

flexDigInGroup16

```
int flexDigInGroup16(Flex_IOPin *digin_pins[],
   unsigned int *result);
```

DESCRIPTION

This function reads the state of a set of digital input channels into the result parameter. The first parameter must be an array of pointers to `Flex_IOPin` structures.

Essentially, this is an array of pins.The last entry in the array must be `FLEX_GROUP_END`. This indicates the end of the pins in the group. The second parameter must be a pointer to an `unsigned int`. The values for each of the inputs will be placed into the variable pointed to by this pointer. The values are placed in the following manner — the input reading from the first pin in the pin group is placed in the first bit (lsb) of the result; the second pin is placed in the second bit; etc.

This function is limited to reading 16 digital inputs at a time. Any unused bits in the result field will be filled with zeroes.

PARAMETERS

digin_pins — array of pointers to `Flex_IOPin` structures (the information structure for each I/O pin), terminated by `FLEX_GROUP_END`. These are the input pins.

result — the results of the digital reads, with the i'th bit in the result field corresponding to the i'th I/O pin in the `digin_pins` array.

RETURN VALUE

`0`: success

`<0`: error (not a valid digital input, or more than 16 digital inputs?)

SEE ALSO

`flexDigShadow`, `flexDigIn`, `flexDigOut`, `flexDigOutGroup16`

6.6.4 Digital Outputs

flexDigOut

```
int flexDigOut(Flex_IOPin *pin, int state);
```

DESCRIPTION

Sets the state of a digital output channel and is non-reentrant.

Note that this function takes into account any inversion by the circuit on the given pin so that the called state is always reflected on the output pin.

PARAMETERS

pin	Pointer to the information structure for the selected I/O pin.
state	Output logic value: 0 = logic low 1 = logic high 2 = tristated (valid for push-pull outputs)

RETURN VALUE

0: success
< 0: error (not a valid digital output?)

SEE ALSO

flexDigShadow, flexDigIn, flexDigInGroup16, flexDigOutGroup16

flexDigShadow

```
int flexDigShadow(Flex_IOPin *pin);
```

DESCRIPTION

Reads the shadow (last value) of the given digital output channel.

PARAMETER

pin Pointer to the information structure for the selected I/O pin.

RETURN VALUE

≥0: value of the shadow
0xFF: no shadow value (digital channel not yet used)
<0: error (not a valid digital input or output?)

SEE ALSO

flexDigIn, flexDigOut, flexDigInGroup16, flexDigOutGroup16

flexDigOutGroup16

```
int flexDigOutGroup16(Flex_IOPin *digout_pins[],
   unsigned int values);
```

DESCRIPTION

This function sets the state of a set of digital output channels based on the `values` parameter. The first parameter must be an array of pointers to `Flex_IOPin` structures.

Essentially, this is an array of pins. The last entry in the array must be `FLEX_GROUP_END`. This indicates the end of the pins in the group. The second parameter must be an `unsigned int`. This function will set the state of the first pin in the pin group from the value of the first bit (lsb) of the values variable; the second pin is set from the value of the second bit; etc.

This function is limited to setting 16 digital outputs at a time.

PARAMETERS

digout_pins Array of pointers to `Flex_IOPin` structures (the information structure for each I/O pin), terminated by `FLEX_GROUP_END`. These are the output pins.

values The values with which to set the digital outputs. The i'th bit in the values field will be used to set the i'th pin in the `digout_pins` array.

RETURN VALUE

`0`: success

`<0`: error (not a valid digital output, or more than 16 digital outputs?)

SEE ALSO

`flexDigShadow`, `flexDigIn`, `flexDigInGroup16`, `flexDigOut`

6.6.5 Analog Inputs

flexAnaInCalib

```
int flexAnaInCalib(Flex_IOPin *pin, int value1, float volts1, int
   value2, float volts2);
```

DESCRIPTION

Calibrates the response of the selected A/D converter channel as a linear function using the two conversion points provided. Gain and offset constants are calculated and placed into a global table. This function is non-reentrant.

PARAMETERS

pin	Pointer to the information structure for this I/O pin.
value1	The first A/D converter value.
volts1	The voltage corresponding to the first A/D converter value.
value2	The second A/D converter value.
volts2	The voltage corresponding to the second A/D converter value.

RETURN VALUE

0: success

< 0: error (not an analog input?)

SEE ALSO

flexAnaIn, flexAnaInVolts

flexAnaIn

```
int flexAnaIn( Flex_IOPin *pin );
```

DESCRIPTION

Reads the state of the external analog channel. Note that this is a raw value that has not been converted to a voltage. The raw values are useful for calibration.

Also note that the value retrieved here is not read when requested; rather, the latest value that has been calculated by the analog input interrupt service request is returned. The update rate is 2.5 ms times the number of analog input channels.

This function is non-reentrant.

PARAMETER

pin Pointer to the information structure for the selected pin.

RETURN VALUE

≥0: a value corresponding to the voltage on the analog input channel (0–4095)
-1: overflow or out of range
-2: error (not an analog input?)

SEE ALSO

flexAnaInVolts, flexAnaInmAmps, flexAnaInCalib

flexAnaInVolts

```
float flexAnaInVolts(Flex_IOPin *pin);
```

DESCRIPTION

Reads the state of the external analog input channel and uses the previously set calibration constants to convert it to volts. This function is non-reentrant.

Also note that the value retrieved here is not read when requested; rather, the latest value that has been calculated by the analog input interrupt service request is returned. The update rate is 2.5 ms times the number of analog input channels.

PARAMETER

pin Pointer to the information structure for the selected pin.

RETURN VALUE

≥0: a voltage value corresponding to the voltage on the analog input channel
FLEX_ANAIN_ERROR: overflow, out-of-range, or not an analog input

SEE ALSO

flexAnaIn, flexAnaInmAmps, flexAnaInCalib

flexAnaInmAmps

```
float flexAnaInmAmps(Flex_IOPin *pin);
```

DESCRIPTION

Reads the state of the external analog input channel and uses the previously set calibration constants to convert it to milliamps. This function is non-reentrant.

Also note that the value retrieved here is not read when requested; rather, the latest value that has been calculated by the analog input interrupt service request is returned. The update rate is 2.5 ms times the number of analog input channels.

PARAMETER

pin Pointer to the information structure for the selected pin.

RETURN VALUE

≥ 0: a milliamp value corresponding to the current on the analog input channel
FLEX_ANAIN_ERROR: overflow, out-of-range, or not an analog input

SEE ALSO

flexAnaIn, flexAnaInVolts, flexAnaInCalib

flexAnaInAverageSetting

```
int flexAnaInAverageSetting(Flex_IOPin *pin, float alpha);
```

DESCRIPTION

Sets the averaging alpha value (weight value) for a given ADC channel to read averaged ADC data. To read the average ADC data use the following functions:

- `flexAnaInAverage()` - read averaged raw data
- `flexAnaInVoltsAverage()` - read averaged voltage
- `flexAnaInmAmpsAverage()` - read averaged current

If you set the alpha value to 1 for a given A/D channel, then you will get the actual unaltered raw data value.

This function is non-reentrant.

PARAMETERS

pin	Pointer to the information structure for the selected pin.
alpha	Value for averaging the given analog input channel; the recommended range is 0.001–1.0.

RETURN VALUE

`0`: success
`-1`: invalid alpha value
`-2`: error (not an analog input?)

SEE ALSO

`flexAnaInAverage`, `flexAnaInVoltsAverage`, `flexAnaInmAmpsAverage`

flexAnaInAverage

```
int flexAnaInAverage(Flex_IOPin *pin);
```

DESCRIPTION

This function is like `flexAnaIn()`, except that it reports the most recent discount averaged value instead of the raw value.

To set the discount average alpha value, execute the `flexAnaInAverageSetting()` function before calling this function.

This function is non-reentrant.

PARAMETER

pin Pointer to the information structure for the selected analog input pin.

RETURN VALUE

≥0: A value corresponding to the raw voltage (0–4095) on the analog input channel
-1: overflow or out of range
-2: error (not an analog input?)

SEE ALSO

`flexAnaInAverageSetting`, `flexAnaInVoltsAverage`, `flexAnaInmAmpsAverage`

flexAnaInVoltsAverage

```
float flexAnaInVoltsAverage(Flex_IOPin *pin);
```

DESCRIPTION

This function is like `flexAnaInVolts()`, except that it reports the most recent discount averaged voltage value instead of the raw voltage value.

To set the discount average alpha value, execute `flexAnaInAverageSetting()` before calling this function.

This function is non-reentrant.

PARAMETER

pin	Pointer to the information structure for the selected analog input pin.

RETURN VALUE

≥0: A value corresponding to the voltage on the analog input channel
FLEX_ANAIN_ERROR: overflow, out of range, or not an analog input

SEE ALSO

flexAnaInAverageSetting, flexAnaInAverage, flexAnaInmAmpsAverage

flexAnaInmAmpsAverage

```
float flexAnaInmAmpsAverage(Flex_IOPin *pin);
```

DESCRIPTION

This function is like `flexAnaInmAmps()`, except that it reports the most recent discount averaged current reading instead of the raw current reading.

To set the discount average alpha value, execute `flexAnaInAverageSetting()` before calling this function.

This function is non-reentrant.

PARAMETER

pin	Pointer to the information structure for the selected analog input pin.

RETURN VALUE

≥0: A milliamp value corresponding to the current on the analog input channel
FLEX_ANAIN_ERROR: overflow, out of range, or not an analog input

SEE ALSO

flexAnaInAverageSetting, flexAnaInAverage, flexAnaInVoltsAverage

flexAnaInNewValue

```
int flexAnaInNewValue(Flex_IOPin *pin, int clear);
```

DESCRIPTION

Reports whether an updated value is available for the given analog input pin. If the `clear` parameter is 1, then a call to this function will clear the result that a new value is available, i.e., assuming that another new value has been processed and made available, the `flexAnaInNewValue()` function will indicate that a new value is not available the next time it is called.

PARAMETER

pin	Pointer to the information structure for the requested pin.
clear	Should be !0 (e.g., 1) to indicate that this new value should subsequently be considered old. 0 indicates that this value will still be considered new on a subsequent call.

RETURN VALUE

`>0`: a new value is available for this analog input
`0`: this analog input has not been updated
`<0`: error (not an analog output?)

6.6.6 Analog Outputs

flexAnaOutCalib

```
int flexAnaOutCalib(Flex_IOPin *pin, int value1, float volts1, int
   value2, float volts2);
```

DESCRIPTION

Calibrates the response of the selected D/A converter channel as a linear function using the two conversion points provided. Gain and offset constants are calculated and placed into a global table. This function is non-reentrant.

PARAMETERS

pin	Pointer to the information structure for this I/O pin.
value1	The first D/A converter value (0–1024).
volts1	The voltage corresponding to the first D/A converter value.
value2	The second D/A converter value (0–1024).
volts2	The voltage corresponding to the second D/A converter value.

RETURN VALUE

0: success

< 0: error (not an analog output?)

SEE ALSO

flexAnaOut, flexAnaOutVolts

flexAnaOut

```
int flexAnaOut(Flex_IOPin *pin, int rawvalue);
```

DESCRIPTION

Sets the voltage on a given analog output pin and a raw value. This function is particularly useful for calibration. This function is non-reentrant.

PARAMETERS

pin	Pointer to the information structure for the selected analog output pin.
rawvalue	Value corresponding to the voltage on the analog output (valid range is 0–1024)

RETURN VALUE

0: success
-1: error (not an analog output?)

SEE ALSO

flexAnaOutVolts, flexAnaOutCalib

flexAnaOutVolts

```
int flexAnaOutVolts(Flex_IOPin *pin, float voltage);
```

DESCRIPTION

Sets the voltage on a given analog output pin by using the previously set calibration constants to calculate the correct data values. This function is non-reentrant.

PARAMETERS

pin	Pointer to the information structure for the selected analog output pin.
voltage	Voltage desired on the output pin.

RETURN VALUE

0: success
-1: error (not an analog output?)

SEE ALSO

flexAnaOut, flexAnaOutCalib

6.6.7 LCD

flexDispInit

```
int flexDispInit(Flex_LCD *lcd, Flex_LCDConf *user_conf);
```

DESCRIPTION

Initializes the given LCD, optionally using the given user configuration structure. This function defaults the cursor to off, the backlight to off, and the contrast control to the display's default value.

PARAMETERS

lcd Pointer to an LCD data structure.

user_conf Pointer to a user-supplied configuration structure for this LCD, or NULL to use the defaults supplied in the RabbitFLEX board's configuration library. The structure definition is given as follows.

```
typedef struct {
   char num_cols;          //Columns on the display
   char num_rows;          //Rows on the display
   //Start address of each row (up to 4 rows)
   char row_addresses[4];
   //  Minimum voltage for contrast control
   float anaout_contrast_min;
   //  Maximum voltage for contrast control
   float anaout_contrast_max;
   //  Minimum duty cycle for contrast control
   int pwm_contrast_min;
   //  Maximum duty cycle for contrast control
   int pwm_contrast_max;
   //  Default scaled value for contrast control (0-255)
   char def_contrast;
} Flex_LCDConf;
```

RETURN VALUE

None.

SEE ALSO

flexDispCursor, flexDispGoto, flexDispClear, flexDispPutc, flexDispPrintf, flexDispOnoff, flexDispBacklight, flexDispContrast

flexDispCursor

```
void flexDispCursor(Flex_LCD *lcd, unsigned int style);
```

DESCRIPTION

Sets the cursor type to be on, off, or blink.

PARAMETERS

lcd	Pointer to an LCD data structure.
style	One of the following cursor macros: DISP_CUROFF: cursor off DISP_CURON: cursor on DISP_CURBLINK: cursor blink

RETURN VALUE

None.

SEE ALSO

flexDispInit, flexDispGoto, flexDispClear, flexDispPutc, flexDispPrintf, flexDispOnoff, flexDispBacklight, flexDispContrast

flexDispGoto

```
int flexDispGoto(Flex_LCD *lcd, unsigned int col, unsigned int row);
```

DESCRIPTION

Places the cursor at the given column and row.

PARAMETERS

lcd	Pointer to an LCD data structure.
col	Column position from top left corner.
row	Row position from top left corner.

RETURN VALUE

0: success
-1: error (column or row not in allowed range for this display?)

SEE ALSO

flexDispInit, flexDispCursor, flexDispClear, flexDispPutc, flexDispPrintf, flexDispOnoff, flexDispBacklight, flexDispContrast

flexDispGetPosition

```
int flexDispGetPosition(Flex_LCD *lcd, unsigned int *col, unsigned
   int *row);
```

DESCRIPTION

Gets the current cursor position based on the column and row parameters.

PARAMETERS

lcd	Pointer to an LCD data structure.
col	Pointer to an `unsigned int` (used to store column position).
row	Pointer to an `unsigned int` (used to store row position).

RETURN VALUE

None.

SEE ALSO

`flexDispGetDimensions`, `flexDispGetContrast`

flexDispGetDimensions

```
void flexDispGetDimensions(Flex_LCD *lcd, unsigned int *num_cols,
   unsigned int *num_rows);
```

DESCRIPTION

Gets the number of columns and rows for the given display.

PARAMETERS

lcd	Pointer to an LCD data structure.
col	Pointer to an `unsigned int` (used to store number of columns).
row	Pointer to an `unsigned int` (used to store number of rows).

RETURN VALUE

None.

SEE ALSO

`flexDispGetPosition`, `flexDispGetContrast`

flexDispClear

```
void flexDispClear(Flex_LCD *lcd);
```

DESCRIPTION

Clears the display and returns the cursor to the home position.

PARAMETER

`lcd`	Pointer to an LCD data structure.

RETURN VALUE

None.

SEE ALSO

flexDispInit, flexDispCursor, flexDispGoto, flexDispPutc, flexDispPrintf, flexDispOnoff, flexDispBacklight, flexDispContrast

flexDispPutc

```
void flexDispPutc(Flex_LCD *lcd, char ch);
```

DESCRIPTION

Puts the given character on the display at the current cursor position.

PARAMETERS

lcd	Pointer to an LCD data structure.
ch	The character to write to the display.

RETURN VALUE

None.

SEE ALSO

flexDispInit, flexDispCursor, flexDispGoto, flexDispClear, flexDispPrintf, flexDispOnoff, flexDispBacklight, flexDispContrast

flexDispPrintf

```
int flexDispPrintf(Flex_LCD *lcd, char *format, ...);
```

DESCRIPTION

Prints the formatted string to the display starting at the current cursor position. The format specifier and additional parameters are specified just like in `printf()`.

PARAMETERS

lcd	Pointer to an LCD data structure.
format	`printf()`-style format specifier string.

RETURN VALUE

The number of characters displayed.

SEE ALSO

flexDispInit, flexDispCursor, flexDispGoto, flexDispClear, flexDispPutc, flexDispOnoff, flexDispBacklight, flexDispContrast

flexDispOnoff

```
void flexDispOnoff(Flex_LCD *lcd, int onOff);
```

DESCRIPTION

Turns the display on or off.

PARAMETERS

lcd	Pointer to an LCD data structure.
onOff	1 = turn the display on, 0 = turn the display off.

RETURN VALUE

None.

SEE ALSO

flexDispInit, flexDispCursor, flexDispGoto, flexDispClear, flexDispPutc, flexDispPrintf, flexDispBacklight, flexDispContrast

flexDispBacklight

```
int flexDispBacklight(Flex_LCD *lcd, int onOff);
```

DESCRIPTION

Turns the display's backlight on or off.

PARAMETERS

lcd	Pointer to an LCD data structure.
onOff	1 = turn the backlight on, 0 = turn the backlight off.

RETURN VALUE

0: success

-1: error (no backlight control on this display)

SEE ALSO

flexDispInit, flexDispCursor, flexDispGoto, flexDispClear, flexDispPutc, flexDispPrintf, flexDispOnoff, flexDispContrast

flexDispContrast

```
int flexDispContrast(Flex_LCD *lcd, char vcontrast);
```

DESCRIPTION

Sets the contrast for the display. The `vcontrast` value range is from 0–255, and is scaled to a contrast-control voltage.

PARAMETERS

lcd	Pointer to an LCD data structure.
vcontrast	Contrast setting (0–255) that is scaled to produce a contrast-control voltage (lower values produce higher contrast).

RETURN VALUE

`0`: success
`-1`: error (no contrast control on this display)

SEE ALSO

`flexDispInit`, `flexDispCursor`, `flexDispGoto`, `flexDispClear`, `flexDispPutc`, `flexDispPrintf`, `flexDispOnoff`, `flexDispBacklight`

flexDispGetContrast

```
char flexDispGetContrast(FLEX_LCD *lcd);
```

DESCRIPTION

Returns the current contrast setting (0–255).

PARAMETER

`lcd`	Pointer to an LCD data structure.

RETURN VALUE

The current contrast setting (0–255).

SEE ALSO

flexDispContrast, flexDispGetDimensions, flexDispGetPosition

6.6.8 Keypad

flexKeyProcess

```
void flexKeyProcess(Flex_Keypad *keypad);
```

DESCRIPTION

For the given keypad, this function scans and processes keypad data for keypresses. It also handles debouncing, releases, and repeats.

PARAMETER

keypad Pointer to the keypad to process.

RETURN VALUE

None.

SEE ALSO

flexKeyInit, flexKeyConfig, flexKeyGet, flexKeyUnget

flexKeyInit

```
void flexKeyInit(Flex_Keypad *keypad);
```

DESCRIPTION

Initializes the given keypad. The `flexKeyInit()` function must be called before any other keypad function for this particular keypad.

PARAMETER

keypad Pointer to the keypad to initialize.

RETURN VALUE

None.

SEE ALSO

flexKeyConfig, flexKeyProcess, flexKeyGet, flexKeyUnget

flexKeyConfig

```
void flexKeyConfig(Flex_Keypad *keypad, char cRaw, char cPress, char
   cRelease, char cCntHold, char cSpdLo, char cCntLo, char cSpdHi);
```

DESCRIPTION

Assigns each key with key press and release codes, hold and repeat ticks for auto repeat, and de-bouncing for a specific keypad.

PARAMETERS

keypad Pointer to the keypad to configure.

cRaw *Raw Key Code Index* Each key has a raw key code. The raw key codes start from 0 and count up. Each crosswired keypad has *m* outputs and *n* inputs, which may or may not correspond to rows and columns.

The raw key code can be calculated by the following formula.

```
[output #] * [num outputs] + [input #]
```

where the output # and input # start from 0. If these correspond to rows and columns, then the formula would be as follows.

```
[row #] * [num rows] + [column #]
```

cPress *Key Press Code* An 8-bit value returned when a key is pressed;
0 = unused.

cRelease *Key Release Code* An 8-bit value returned when a key is released;
0 = unused.

cCntHold *Hold Ticks* How long to hold before repeating;
0 = No Repeat.

cSpdLo *Low-Speed Repeat Ticks* How many times to repeat;
0 = none.

cCntLo *Low-Speed Hold Ticks* How long to hold before going to high-speed repeat;
0 = slow only.

cSpdHi *High-Speed Repeat Ticks* How many times to repeat after low-speed repeat;
0 = none.

RETURN VALUE

None.

SEE ALSO

flexKeyInit, flexKeyProcess, flexKeyGet, flexKeyUnget

flexKeyUnget

```
int flexKeyUnget(Flex_Keypad *keypad, char cKey);
```

DESCRIPTION

Pushes the key press (cKey) on the specified keypad back onto the top of the input queue.

PARAMETERS

keypad	Pointer to the keypad to use.
cKey	Key code to put back onto the top of the queue.

RETURN VALUE

>0: key code successfully put back onto the queue
0: failure (keypad queue full?)

SEE ALSO

flexKeyInit, flexKeyConfig, flexKeyProcess, flexKeyGet

flexKeyGet

```
char flexKeyGet(Flex_Keypad *keypad);
```

DESCRIPTION

Gets the next key press for the selected kepad from the keypad input queue.

PARAMETER

keypad Pointer to the keypad to use.

RETURN VALUE

>0: next key press
0: no key presses available

SEE ALSO

flexKeyInit, flexKeyConfig, flexKeyProcess, flexKeyUnget

6.6.9 Speaker

flexSpeakerPWM

```
int flexSpeakerPWM(Flex_IOPin *pin);
```

DESCRIPTION

Selects the PWM channel to be used for the audio and tone speaker drivers.

PARAMETER

pin Pointer to the PWM pin to use for the speaker.

RETURN VALUE

0: success
-1: error (not a PWM pin?)

SEE ALSO

flexToneActivate, flexAudioActivate

flexToneActivate

```
unsigned int flexToneActivate();
```

DESCRIPTION

Enables the tone driver so that samples can be loaded and played.

The tone driver consumes 10–20% of the processor's time while it is running. Use `flexToneShutdown()` to disable the tone driver while not in use.

Do not use the tone and audio drivers simultaneously. You may switch between them within the same program.

RETURN VALUE

The frequency of PWM pulses output by the tone driver.

SEE ALSO

flexSpeakerPWM, flexToneShutdown, flexToneLoad, flexToneStop

flexToneLoad

```
int flexToneLoad(char *buffer, int bufsize, char repeat);
```

DESCRIPTION

Loads tone samples to be played into the tone driver buffer. This sample will be played repeatedly for the specified number of repeats. If the number of repeats is negative, the clip will be repeated indefinitely until `flexToneStop()` is called.

PARAMETERS

buffer	Pointer to an array of bytes to be loaded.
bufsize	Number of bytes in the buffer.
repeat	Number of times the buffer contents will be repeated. A value >128 will repeat the buffer indefinitely. A value of 0 or 1 will play the tone once.

RETURN VALUE

The number of bytes actually loaded into the driver. This number may be smaller than the requested `bufsize` or it may even be 0 because of limits on the size of the audio driver's internal buffer.

SEE ALSO

flexSpeakerPWM, flexToneActivate, flexToneShutdown, flexToneStop

flexToneStop

```
void flexToneStop();
```

DESCRIPTION

Immediately stops any tone that is playing.

RETURN VALUE

None.

SEE ALSO

flexSpeakerPWM, flexToneActivate, flexToneShutdown, flexToneLoad

flexToneShutdown

```
void flexToneShutdown();
```

DESCRIPTION

Turns off the tone driver. Useful to preserve processor resources.

RETURN VALUE

None.

SEE ALSO

flexSpeakerPWM, flexToneActivate, flexToneLoad, flexToneStop

flexAudioActivate

```
unsigned int flexAudioActivate();
```

DESCRIPTION

Enables the audio driver so that samples can be loaded and played.

The audio driver consumes 10–20 of the processor's time while it is running. Use `flexAudioShutdown()` to disable the audio driver while not in use.

Do not use the tone and audio drivers simultaneously. You may switch between them within the same program.

RETURN VALUE

The frequency of PWM pulses output by the audio driver.

SEE ALSO

flexSpeakerPWM, flexAudioShutdown, flexAudioPlay, flexAudioStop, flexAudioPlaying, flexAudioSetRate, flexAudioLoad

flexAudioPlay

```
void flexAudioPlay();
```

DESCRIPTION

Plays samples that have been loaded into the driver buffer, see `flexAudioLoad()`. The driver will continue to play samples until its buffer is empty.

RETURN VALUE

None.

SEE ALSO

flexSpeakerPWM, flexAudioActivate, flexAudioShutdown, flexAudioStop, flexAudioPlaying, flexAudioSetRate, flexAudioLoad

flexAudioPlaying

```
int flexAudioPlaying();
```

DESCRIPTION

Determines whether the audio driver is currently playing samples.

RETURN VALUE

`1`: if the driver is playing audio samples,
`0`: if the driver is not playing audio samples.

SEE ALSO

flexSpeakerPWM, flexAudioActivate, flexAudioShutdown, flexAudioPlay, flexAudioStop, flexAudioSetRate, flexAudioLoad

flexAudioSetRate

```
void flexAudioSetRate(unsigned int sample_rate);
```

DESCRIPTION

Tells the driver to treat the incoming data as audio sampled at this given rate. This tells the driver how fast the data should be played through.

PARAMETER

sample_rate Sampling rate of the input data (1–65,535 Hz).

RETURN VALUE

None.

SEE ALSO

flexSpeakerPWM, flexAudioActivate, flexAudioShutdown, flexAudioStop, flexAudioPlaying, flexAudioPlay, flexAudioLoad

flexAudioLoad

```
int flexAudioLoad(char *buffer, int bufsize);
```

DESCRIPTION

Loads audio samples to be played into the audio driver buffer. These samples will not actually play unless `flexAudioPlay()` is called.

> You only need to call `flexAudioPlay()` once when starting. Data in subsequent calls will also be played as long as the audio buffer does not become empty.

PARAMETER

buffer	Pointer to an array of bytes to be loaded.
bufsize	Number of bytes in the buffer.

RETURN VALUE

The number of bytes actually loaded into the driver. This number may be smaller than the requested `bufsize` or it may even be 0 because of limits on the size of the audio driver's internal buffer.

SEE ALSO

`flexSpeakerPWM`, `flexAudioActivate`, `flexAudioShutdown`, `flexAudioPlay`, `flexAudioStop`, `flexAudioPlaying`, `flexAudioSetRate`

flexAudioShutdown

```
void flexAudioShutdown();
```

DESCRIPTION

Turns off the audio driver. Useful to preserve processor resources.

RETURN VALUE

None.

SEE ALSO

flexSpeakerPWM, flexAudioActivate, flexAudioPlay, flexAudioStop, flexAudioPlaying, flexAudioSetRate, flexAudioLoad

flexAudioStop

```
void flexAudioStop();
```

DESCRIPTION

Immediately stops any sound that is playing and empties the sample buffer. Any samples loaded after calling `flexAudioStop()` will not play until `flexAudioPlay()` is called.

RETURN VALUE

None.

SEE ALSO

flexSpeakerPWM, flexAudioActivate, flexAudioShutdown, flexAudioPlay, flexAudioPlaying, flexAudioSetRate, flexAudioLoad

6.6.10 Serial Communication

serMode

```
int serMode(int mode);
```

DESCRIPTION

Sets up user interface for the serial communication lines. This function must be called *after* executing `serXOpen()`. Whether you are opening one or multiple serial ports, this function must be executed after executing the last `serXOpen()` AND before you start using any of the serial ports.

This function is non-reentrant.

PARAMETER

`mode` Defined serial port configuration if devices are installed

Table 6-7.

Mode	Serial Port	
	F	D or E
0	RS-232, 3-wire	RS-232, 3-wire (if present)
1	RS-232, 5-wire	CTS/RTS

RETURN VALUE

`0`: Valid mode
`1`: Invalid mode

SEE ALSO

API functions in `RS232.lib`. Documention available in "Function Lookup" in the Dynamic C pulldown help menu; function documentation also in the *Dynamic C Function Reference Manual*.

ser485Tx

```
void ser485Tx(void);
```

DESCRIPTION

Enables the RS-485 transmitter. The `brdInit()` function must be executed before running this function. This function is non-reentrant.

After the byte or block of data has been transmitted, the RS-485 transmitter must be idle and its transmit buffer empty before an application can disable the transmitter and thus receive data from the RS-485 interface. There are two methods for identifying when to disable the transmitter.

1. Poll the receive data buffer for the transmitted data that is echoed back to it. This is the easier method. The following code snippet illustrates disabling the transmitter after one byte has been transmitted using serial port C. You can also send a block of data before disabling the transmitter.

```
ser485Tx();                     // enable transmitter
serCputc ( 0x55 );              // send byte
while (serCgetc() == -1);       // wait for echo
ser485Rx();                     // disable transmitter
```

2. Read SxSR, the status register of the serial port you are using. Bits 2 and 3 report on the RS-485 transmitter and its buffer. The following code snippet illustrates how to check these bits if you are using serial port C.

```
ser485Tx();
serCputc(0x55);
// Wait for Tx Data buffer to become empty
while(serCwrFree() != COUTBUFSIZE);
// Wait for Tx buffer and shift register on uP to become empty
while(RdPortI(SCSR)&0x08 || RdPortI(SCSR)&0x04);
ser485Rx();
```

RETURN VALUE

None

SEE ALSO

brdInit, ser485Rx

ser485Rx

```
void ser485Rx(void);
```

DESCRIPTION

Disables the RS-485 transmitter. This puts you in a listen mode, which allows you to receive data from the RS-485 interface. The `brdInit()` function must be executed before running this function. This function is non-reentrant.

RETURN VALUE

None

SEE ALSO

brdInit, ser485Tx

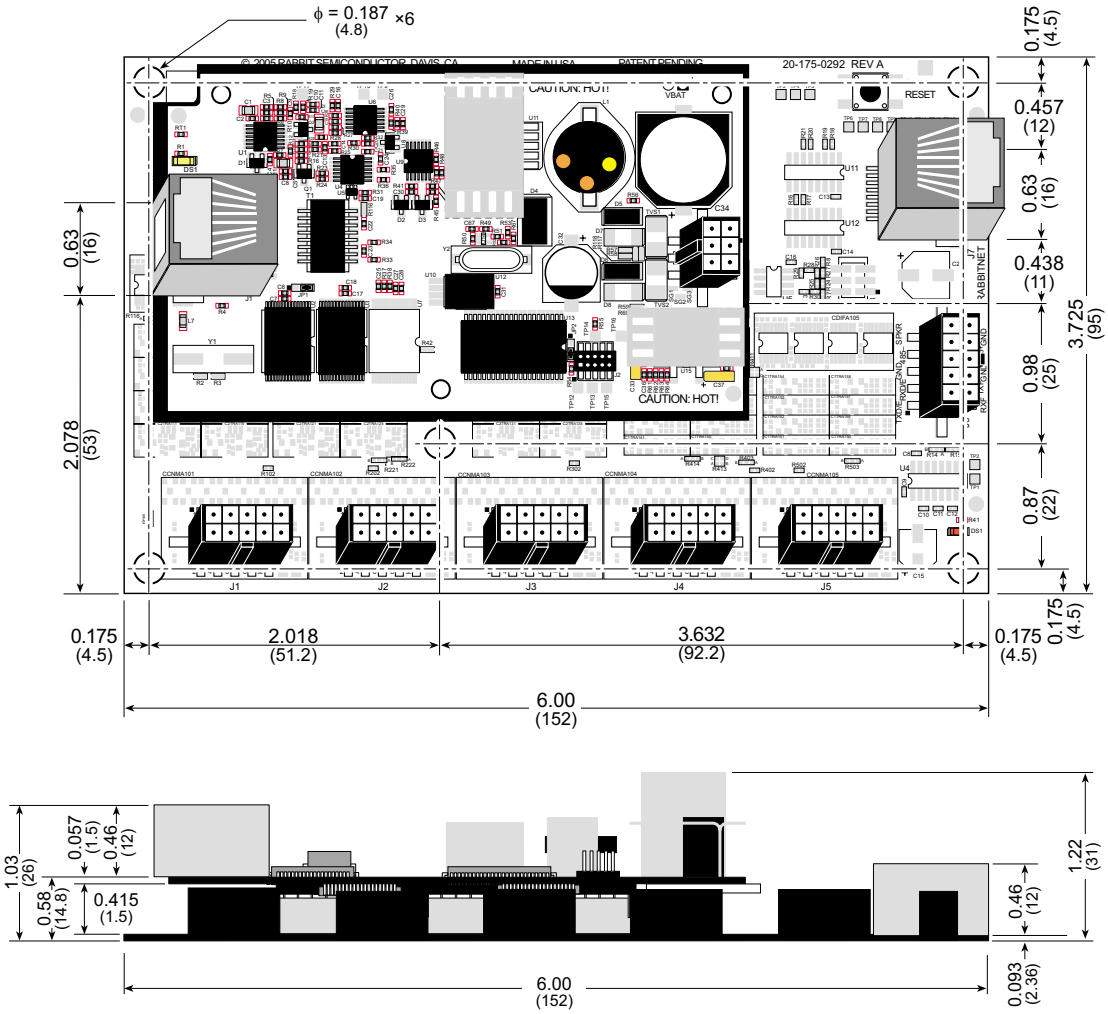
APPENDIX A. RABBITFLEX BL300F SPECIFICATIONS

This appendix provides the board specifications for the RabbitFLEX BL300F.

A.1 Electrical and Mechanical Characteristics

Figure A.1 shows the mechanical dimensions for the RabbitFLEX BL300F.

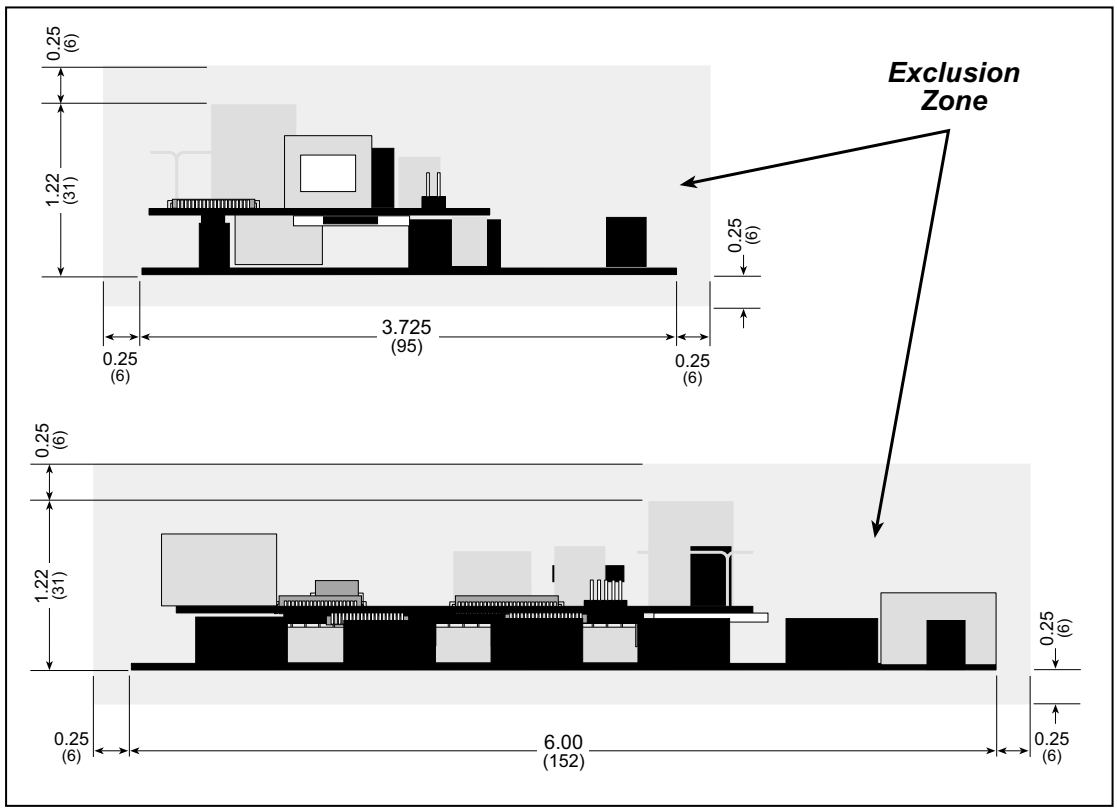
Figure A.1 RabbitFLEX BL300F Dimensions



NOTE: All measurements are in inches followed by millimeters enclosed in parentheses. All dimensions have a manufacturing tolerance of ±0.01" (0.25 mm). There are 6 mounting holes, each with diameter 0.187 inches (4.8 mm).

Rabbit recommends an exclusion zone of 0.25" (6 mm) around the RabbitFLEX BL300F board in all directions when the RabbitFLEX BL300F board is incorporated into an assembly that includes other components. Figure A.2 shows this exclusion zone.

Figure A.2 RabbitFLEX BL300F Exclusion Zone



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

The RabbitFLEX boards were tested for heat dissipation over the specified operating temperature range, and normal heat dissipation by convection was found to be adequate. If you plan to use the board in a tightly enclosed space, additional forced-air cooling will likely be needed.

Table A-1 lists the electrical, mechanical, and environmental specifications for the RabbitFLEX BL300F.

Table A-1. RabbitFLEX BL300F Specifications

Parameter	With PowerCore 3800	With PowerCore 3810
Microprocessor	Rabbit 3000® at 51.6 MHz	Rabbit 3000® at 25.8 MHz
EMI Reduction	Spectrum spreader for reduced EMI (ElectroMagnetic Interference)	
Ethernet Port	10Base-T interface, RJ-45, 2 LEDs	—
SRAM	512K program 512K data	256K data
Flash Memory (program)	512K	512K
Serial Flash Memory	1 Mbyte	—
Backup Battery	3 V lithium coin type 2032, 220 mA·h (to support RTC and data SRAM)	
Configurable I/O	40 individually configurable I/O pins: • all 40 configurable as digital inputs, sinking or sourcing digital drivers, line drivers, bidirectional logic, or as +5 V DC power points • up to twenty-four configurable as +3.45 V DC power points • up to 16 configurable as analog voltage or current inputs	
Analog Output	Up to two analog outputs available, one of which can be configured to drive an 8 Ω speaker	
Serial Ports	4 serial ports: • two RS-232 or one RS-232 (with CTS/RTS) • one RS-485 (terminated or unterminated) or one RS-422 RabbitNet™ SPI master port • one serial port dedicated for programming/debug	
Serial Rate	Max. asynchronous rate = CLK/8, Max. synchronous rate = CLK/2	
Connectors	RJ-45 connectors: one Ethernet and one RabbitNet™ (if options selected) Friction-lock connectors: up to six polarized 2 × 5 terminals with 3 mm pitch one 2 × 3 terminal with 3 mm pitch Programming port: 2 × 5 IDC, 1.27 mm pitch	
Real-Time Clock	Yes	
Timers	Ten 8-bit timers (6 cascadable, 3 reserved for internal peripherals), one 10-bit timer with 2 match registers	
Watchdog/Supervisor	Yes	

Table A-1. RabbitFLEX BL300F Specifications (Continued)

Parameter		With PowerCore 3800	With PowerCore 3810
Input Power Options	DC	Unregulated 8-43 V DC (draws 13.3 W)	Unregulated 8-40 V DC (draws 6.7 W)
		24–60 V AC with center-tapped transformer (draws 13.3 W)	19–57 V AC with center-tapped transformer (draws 6.7 W)
	AC	12-36 V AC with untapped standard transformer (draws 13.3 W)	10-29 V AC with untapped standard transformer (draws 6.7 W)
Operating Temperature		–40°C to +70°C	
Humidity		5% to 95%, noncondensing	
Standoffs/Spacers		Provision for 6	
Board Size (without wiring harness)		3.725" × 6.000" × 1.22" (95 mm × 152 mm × 31 mm)	

A.2 Conformal Coating

The areas around the crystal oscillator and the battery backup circuit on the PowerCore module have had the Dow Corning silicone-based 1-2620 conformal coating applied. The conformal coating protects these high-impedance circuits from the effects of moisture and contaminants over time.

The conformally coated area on the PowerCore module is on its bottom side. This area is inaccessible when the module is attached to the RabbitFLEX base board. Under normal circumstances, there is no need to access it. Any components in the conformally coated area may be replaced using standard soldering procedures for surface-mounted components. A new conformal coating should then be applied to offer continuing protection against the effects of moisture and contaminants.

The *PowerCore FLEX User's Manual* contains further information and an illustration of the conformally coated area.

> **NOTE:** For more information on conformal coatings, refer to Rabbit Technical Note 303, "Conformal Coatings."

APPENDIX B. POWER SUPPLY

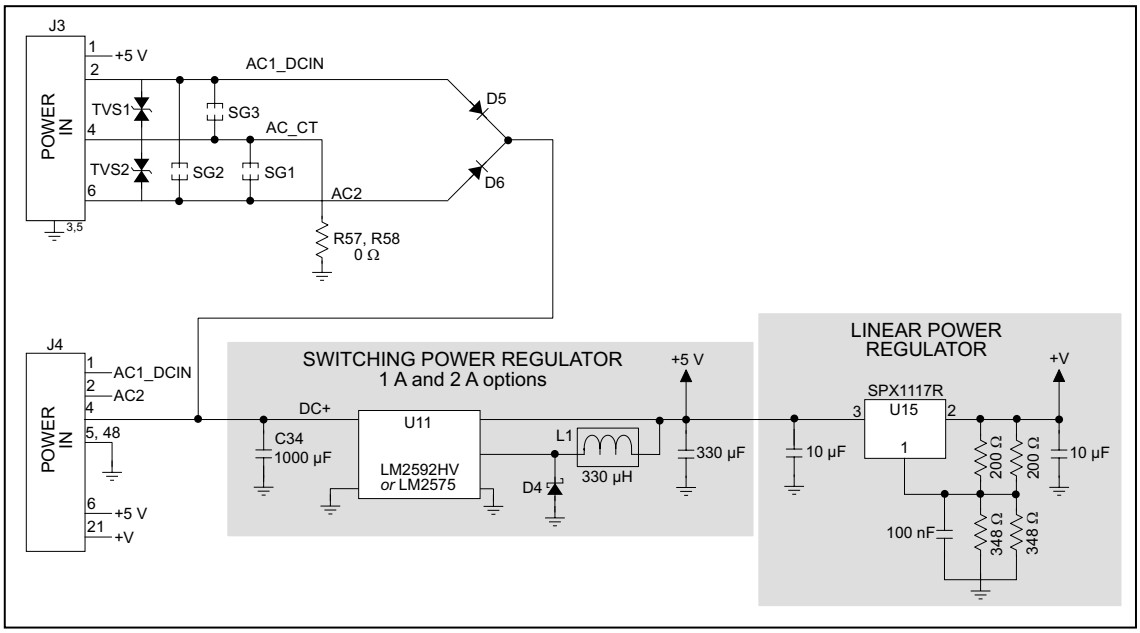
This appendix provides information for the power supplies on the PowerCore modules used on the RabbitFLEX BL300F board. Information on the reset generator used in power management is also included.

B.1 Power Supplies

The PowerCore module has a 6-pin locking connector on the top side at J3 designed to accept a wire harness that brings in power. The J3 label on the PowerCore module is difficult to see, but luckily the 6-pin locking connector is easy to find.

The PowerCore module receives power in the form of unregulated AC. There is an onboard +5 V DC switching regulator that is available in 1 A (PowerCore 3810) and 2 A (PowerCore 3800). The +5 V DC is regulated down to 3.45 V by a linear regulator on the PowerCore module.

Figure B.1 PowerCore Module Power Supplies



The PowerCore 3800 and the PowerCore 3810 are built for use with a full-wave rectifier and center-tapped AC transformer.

B.2 Battery-Backup Circuits

The data SRAM and the real-time clock on the PowerCore module have battery backup. Power to the SRAM and the real-time clock (VRAM) is provided by two different sources, depending on whether the PowerCore module is powered or not. When the PowerCore module is powered normally, and the +5 V supply is within operating limits, the SRAM and the real-time clock are powered from the +5 V supply. If power to the board is lost or falls below 4.38 V, the VRAM and real-time clock power will come from the battery. The reset generator circuit controls the source of power by way of its /RESET output signal.

A replaceable 220 mA·h lithium battery provides power to the real-time clock and SRAM when external power is removed from the circuit board. The drain on the battery is typically less than 6 µA when there is no external power applied to the PowerCore module, and so the expected shelf life of the battery is:

$$\frac{220\ \text{mA} \cdot \text{h}}{6\ \mu\text{A}} = 4.2\ \text{years}$$

Note that the shelf life of a lithium ion battery is ultimately 10 years. The PowerCore module does not drain the battery while it is powered up normally.

B.2.1 Replacing the Backup Battery

The battery is user-replaceable, and is fitted in a battery holder. To replace the battery, slide out the old battery. Use only a 2032 or equivalent replacement lithium battery, and insert it into the battery holder with the + side facing away from the PowerCore module.

NOTE: The SRAM contents and the real-time clock settings will be lost if the battery is replaced with no power applied to the PowerCore module. Exercise care if you replace the battery while external power is applied to the PowerCore module.

CAUTION: Be careful when replacing the battery with external AC power applied to the PowerCore module. AC voltages up to 60 V may be present on locking connector J3.

CAUTION: There is an explosion danger if the battery is short-circuited, recharged, or replaced incorrectly. Replace the battery only with the same type or an equivalent type recommended by the battery manufacturer. Dispose of used batteries according to the battery manufacturer's instructions.

B.3 Reset Generator

The core module uses a reset generator to reset the Rabbit 3000 microprocessor when the voltage drops below the voltage necessary for reliable operation. The reset typically occurs at 4.38 V.

B.4 Power On / Reset State

The table below shows the circuits that are affected by power on or reset.

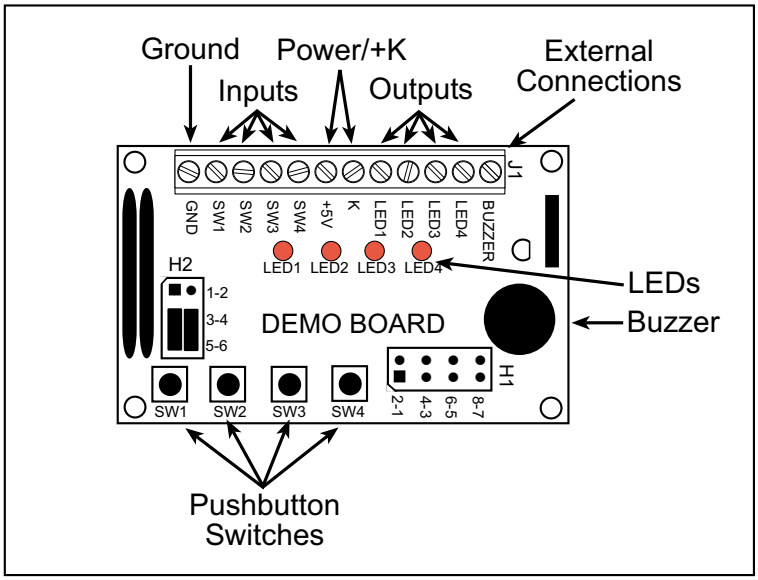
Table B-1.

Circuit	Power On / Reset State
Bidirectional Logic Lines	Input
Keypad Outputs	Off, High Impedance
Sinking Drivers, 1 A 40 V	Off, High Impedance
Sinking Drivers, 100 mA 40 V	Off, High Impedance
Sourcing Drivers, 400 mA 40 V	Off, High Impedance
Sourcing Drivers, 50 mA 5 V	On, 5 V
Line Drivers, 100 Ω 5 V	Off, 0 V

APPENDIX C. DEMONSTRATION BOARD

The demonstration board allows you to exercise the digital I/O on a RabbitFLEX board. The demonstration board, shown in Figure C.1, has four connections for digital inputs that create logic highs while the corresponding pushbutton switches on the demonstration board are pressed. There are four connections for sinking digital outputs that light up corresponding LEDs while a digital output on the RabbitFLEX board is activated.

Figure C.1 Demonstration Board



C.1 Demonstration Board Connections

Before running digital I/O sample programs that can use the demonstration board, connect the demonstration board to the RabbitFLEX SBC40 with one of the cable assemblies included with the RabbitFLEX SBC40 Tool Kit.

C.1.1 Power and Ground Connections

There are two power connections on the demo board: +5 V and K. If you selected the 5 V supply option on the RabbitFLEX SBC40 connector you will be using, you can attach its corresponding green wire from the cable assembly to +5 V on screw terminal header J1 on the demo board. Otherwise you will need to attach an external +5 V power supply to J1.

Connect the black wire from the cable assembly to GND on screw terminal header J1. If you are using an external power supply, connect its ground to GND on J1 as well.

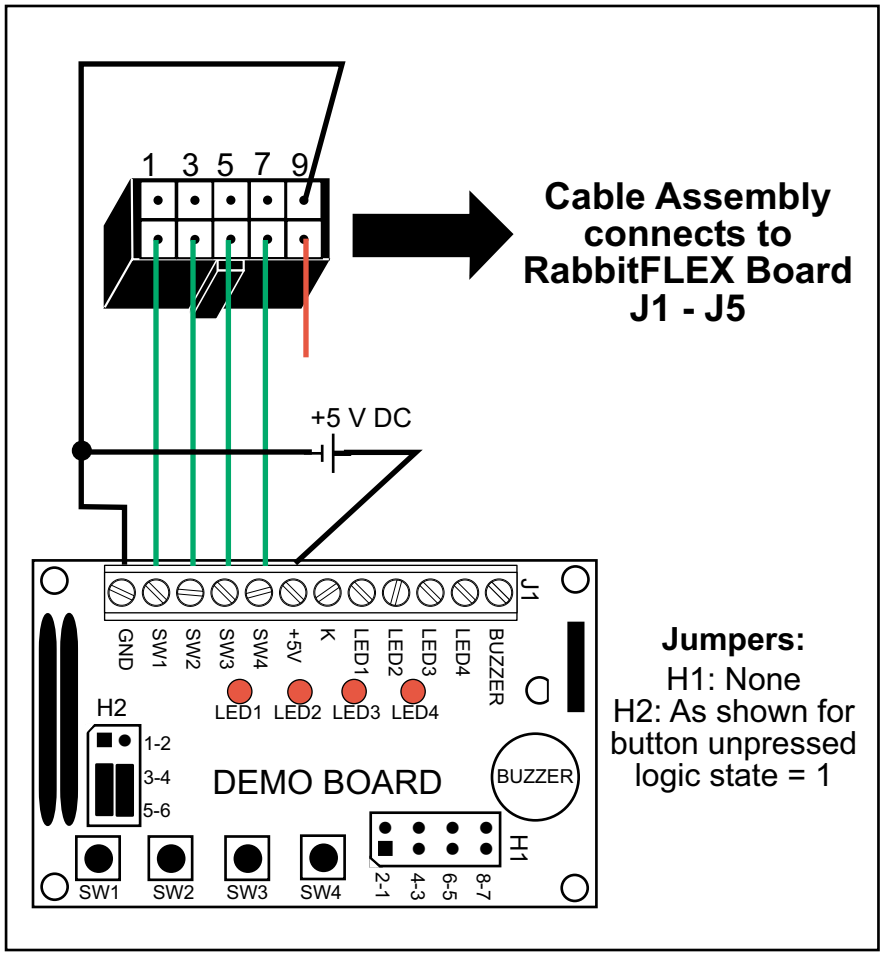
C.1.2 Digital Input Connections

To run the digital input sample programs, you must connect at least one demo board switch to a digital input on the RabbitFLEX board. There are four demo board switches, so you may connect up to four digital inputs.

The 4.4 V threshold digital inputs may or may not work reliably since the switching threshold it so close to 5 V. The bidirectional logic must be in an input state, which is one of the things that is done during the board initialization function (`brdInit()`).

Connect the green wire of the cable assembly that corresponds to the selected digital input to SW1, SW2, SW3 or SW4 on screw terminal header J1 on the demo board. Connect the polarized connector of the cable assembly to the connector that has the digital input(s) you selected on the RabbitFLEX board. Figure C.2 shows the demo board pushbutton switches connected to pins 2, 4, 6 and 8 of the cable assembly. In this case, the polarized connector of the cable assembly should be connected to one of the connectors on the RabbitFLEX SBC40 that has digital inputs on pins 2, 4, 6 and 8.

Figure C.2 Demo Board Connected to Pins 2, 4, 6, 8 on Cable Assembly

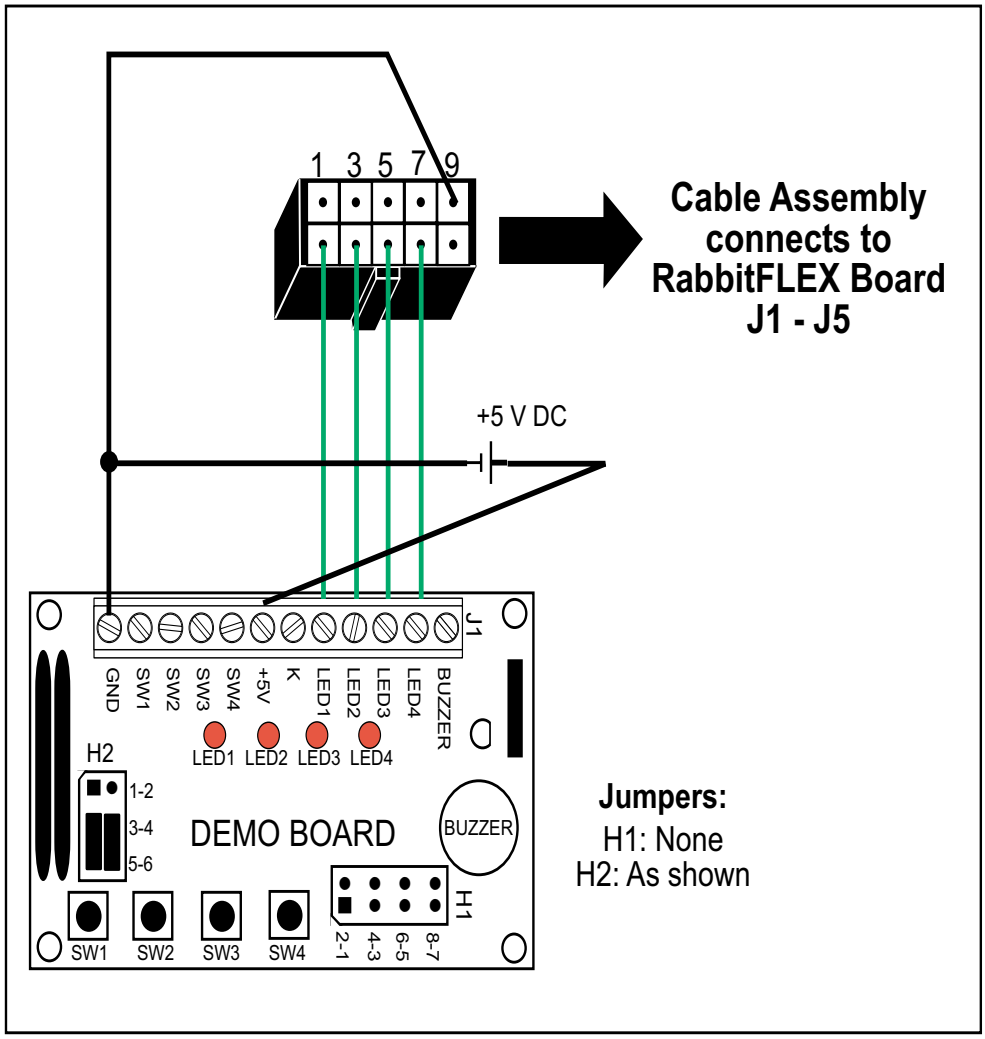


C.1.3 Digital Output Connections

To run the digital output sample programs, you must connect at least one demo board LED to a 100 mA sinking driver on the RabbitFLEX board. There are four demo board LEDs, so you may connect up to four 100 mA sinking drivers.

Connect the green wire of the cable assembly that corresponds to the selected digital output to LED1, LED2, LED3 or LED4 on screw terminal header J1 on the demo board. Connect the polarized connector of the cable assembly to the connector that has the digital output(s) you selected on the RabbitFLEX board. Figure C.2 shows the demo board LEDs connected to pins 2, 4, 6 and 8 of the cable assembly. In this case, the polarized connector of the cable assembly should be connected to one of the connectors on the RabbitFLEX SBC40 that has 100 mA sinking drivers on pins 2, 4, 6 and 8.

Figure C.3 Demo Board Connected to Pins 2, 4, 6, 8 on Cable Assembly



C.1.4 Run Sample Programs

Make sure the programming cable and transformer are connected as described in Chapter 2, "Getting Started."

Now you can run your application or a sample program. To exercise digital inputs, you can run either `digin_simple.c` or `digin_groups.c`. To exercise the digital outputs, you can run either `digout_simple.c` or `digout_groups.c`.

APPENDIX D. RABBITFLEX KEYPAD/DISPLAY KIT

Rabbit offers a RabbitFLEX Keypad/Display Kit for the RabbitFLEX board that provides the hardware components required to run the sample programs and to demonstrate the functionality of your keypad/display design. Table D-1 lists the items in the RabbitFLEX Keypad/Display Kit along with their part numbers.

Table D-1. RabbitFLEX Keypad/Display Kit Parts

Description	Quantity	Supplier	Part Number
4 × 20 Character Display	1	Rabbit	535-0026
5 Ω, 0.5 W Current-Limiting Resistor	1	Rabbit	202-0051
2 × 6 Keypad	1	Rabbit	505-0027
Keypad Connector	1	Rabbit	498-0013
Cable Assembly	2	Rabbit	151-0153

D.1 Keypad

To make the hardware connections between the keypad and the RabbitFLEX board, you will first need:

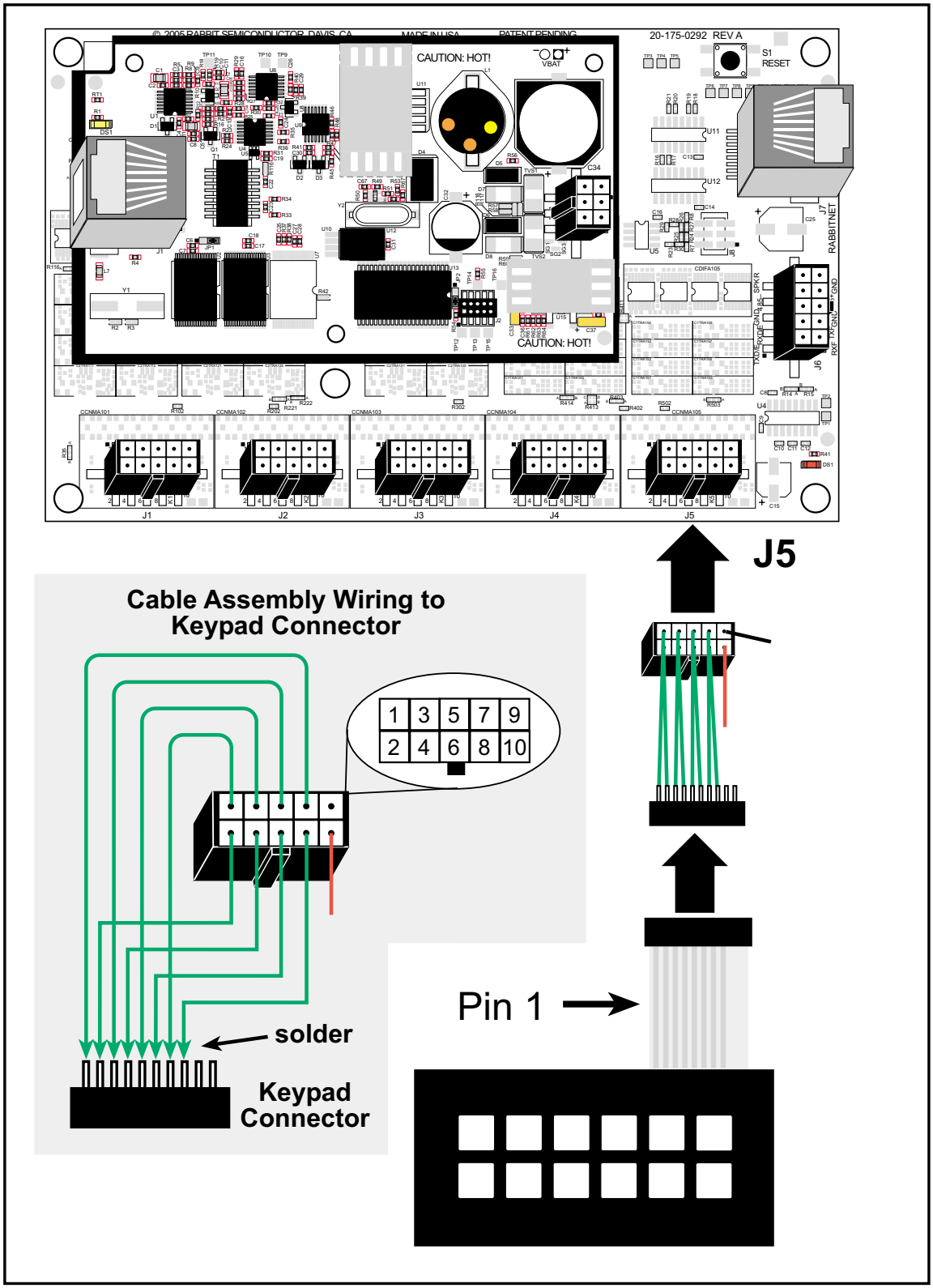
- One of the cable assemblies from the RabbitFLEX BL300F Tool Kit
- The keypad and the keypad connector from the RabbitFLEX Keypad/Display Kit.

Solder the green wires from the cable assembly to the keypad connector as shown in Figure D.1. Slide the flexible cable from the keypad into the keypad connector as shown in the diagram and connect the friction-lock connector of the cable assembly to J5 on the RabbitFLEX board.

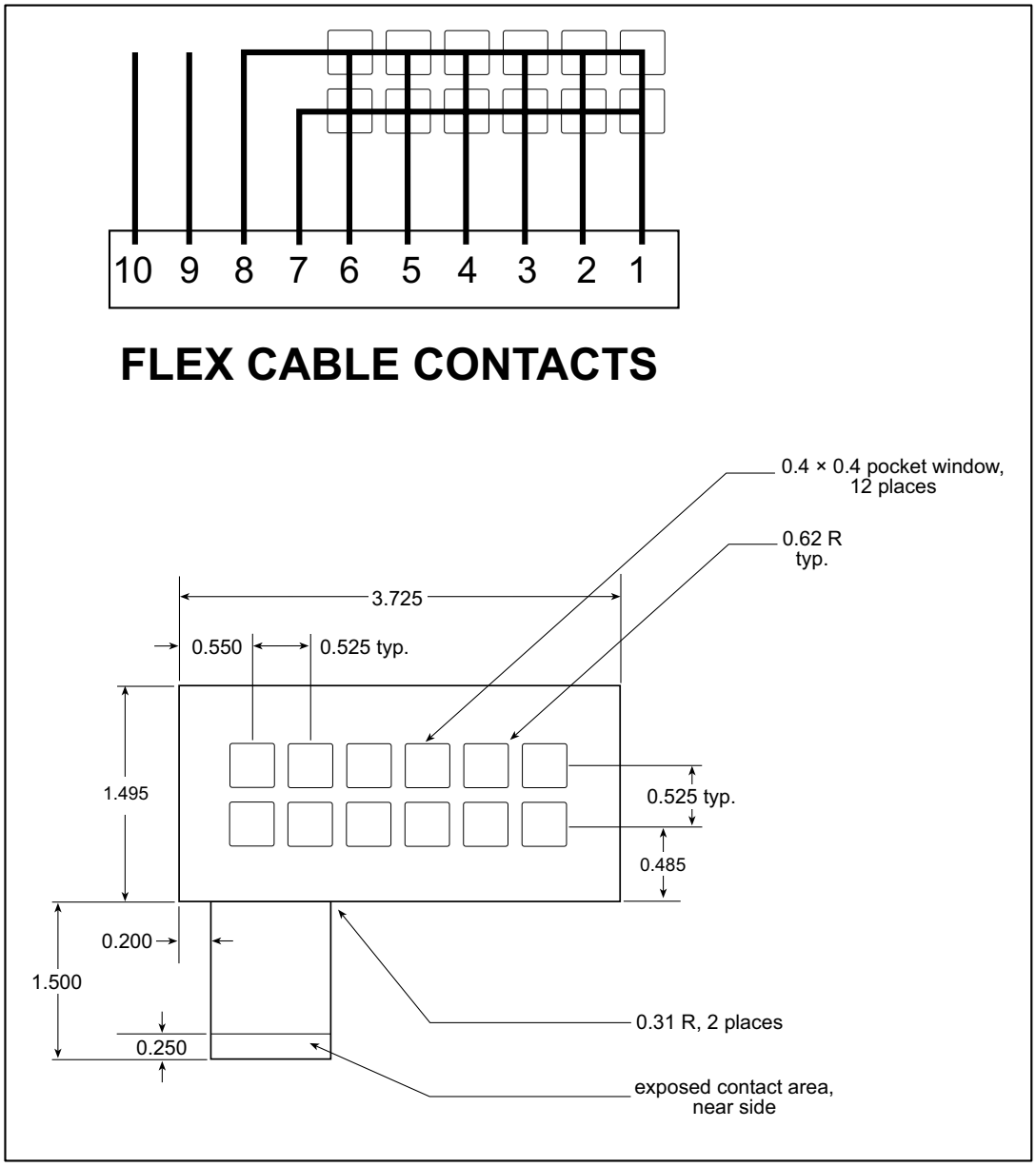
D.2 Keypad Template

If you want to label the keypad keys, a piece of paper can be inserted between the two layers of the keypad.

Figure D.1 Connect Keypad to Keypad Connector J5



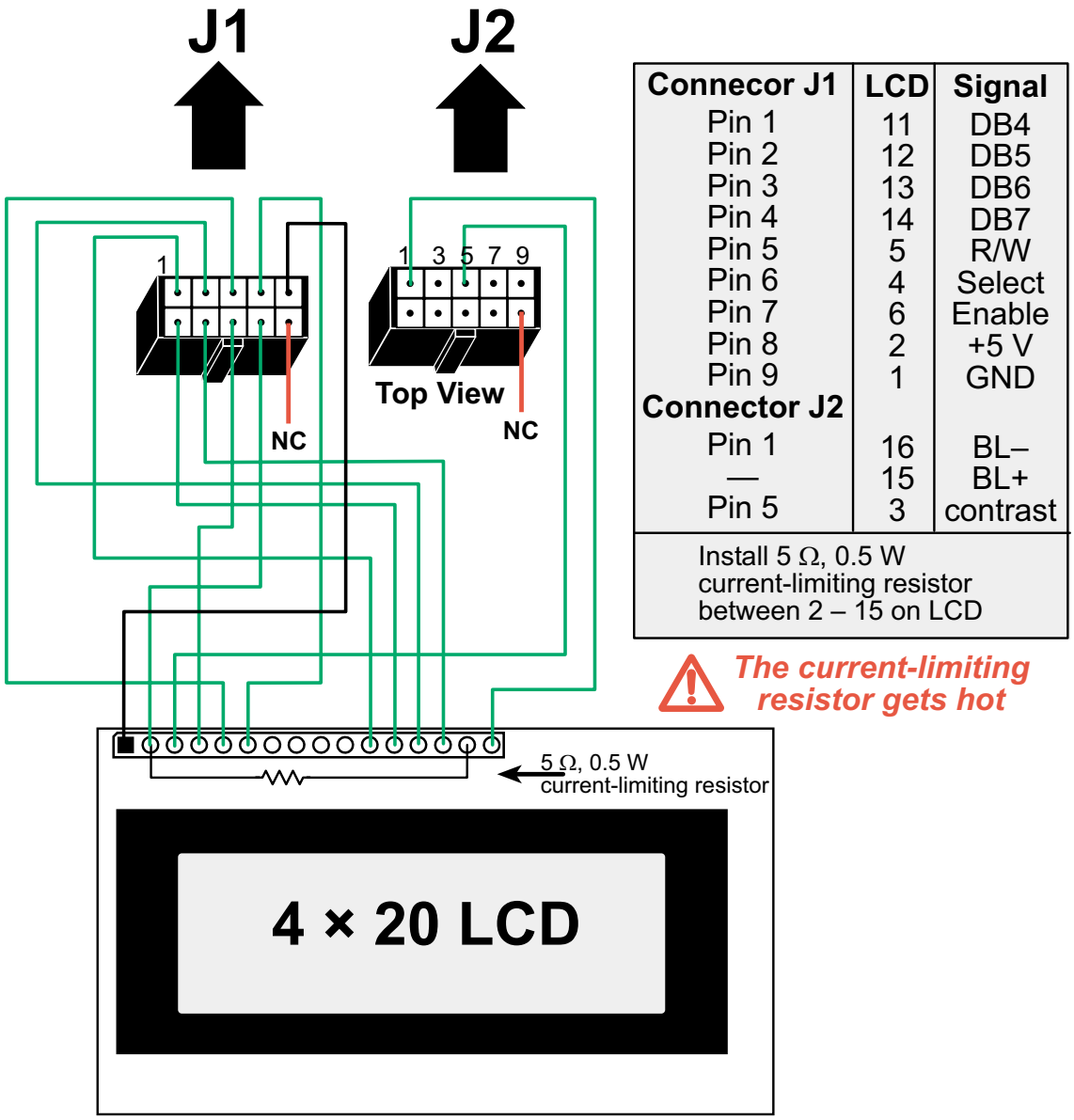
D.3 2 × 6 Keypad Datasheet



D.4 LCD Module

The LCD supplied with the RabbitFLEX Keypad/Display Kit does not have any connectors attached. Use two cable assemblies from the RabbitFLEX BL300F Tool Kit to connect the LCD display to connectors J1 and J2 on the RabbitFLEX board as shown below in Figure D.2. Remember to add the 5 Ω, 0.5 W current-limiting resistor.

Figure D.2 Connecting LCD Display to RabbitFLEX Board Connectors J1 and J2



Connecor J1	LCD	Signal
Pin 1	11	DB4
Pin 2	12	DB5
Pin 3	13	DB6
Pin 4	14	DB7
Pin 5	5	R/W
Pin 6	4	Select
Pin 7	6	Enable
Pin 8	2	+5 V
Pin 9	1	GND
Connector J2		
Pin 1	16	BL–
—	15	BL+
Pin 5	3	contrast

Install 5 Ω, 0.5 W current-limiting resistor between 2 – 15 on LCD

The current-limiting resistor gets hot

D.5 4 × 20 Character LCD Datasheet

EVER CATCHER GROUP
WE CATCH THE BEST TECH. FOREVER

EVERBOUQUET/WAYTON

GENERAL SPECIFICATIONS FOR CHARACTER LCD MODULE

PIN ASSIGNMENT

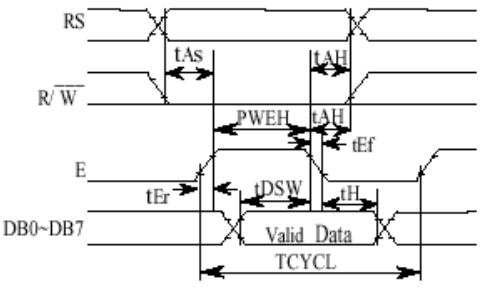
PIN NO	SYMBOL	LEVEL	FUNCTION		PIN NO	SYMBOL	LEVEL	FUNCTION
1	Vss	—	Power Supply	0V (GND)	5	R/$\overline{W}$	H/L	H:Data Read (Module→MPU)
2	Vcc	—		+5V				L:Data Write (Module←MPU)
3	Vo	—		for LCD Drive	6	E	H,H→	Enable Signal
4	RS	H/L	Register Select Signal Register H: Data Input Select L: Instruction Input		7 | 14	DB0 | DB7	H/L	Data Bus Line

* Interface between Data Bus line and 4- bit or 8-bit MPU is available. Data transfer are made in twice in case of 4-bit MPU. and once in case of 8-bit MPU.

TIMING CHART

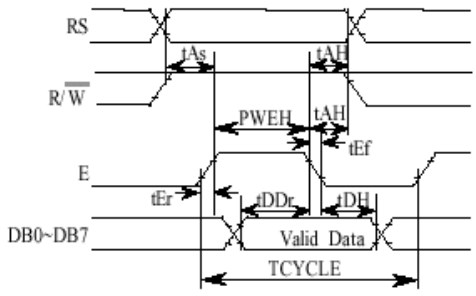
ITEM	SYMBOL	Measuring Condition	STANDARD VALUE			UNIT
			MIN.	TYP.	MAX.	
Enable Cycle Time	t_{cycE}	Figs. 1,2	1000	—	—	ns
Enable Pulse Width, High Level	PW_{EH}	Figs. 1,2	450	—	—	ns
Enable Rise and Decay Time	t_{Er}, t_{Ef}	Figs. 1,2	—	—	25	ns
Address Setup Time, RS, R/W-E	t_{AS}	Figs. 1,2	140	—	—	ns
Data Delay Time	t_{DDR}	Figs. 2	—	—	320	ns
Data Setup Time	t_{DSW}	Figs. 1	195	—	—	ns
Data Hold Time (Write Operation)	t_H	Figs. 1	10	—	—	ns
Data Hold Time (Read Operation)	t_{DHR}	Figs. 2	20	—	—	ns
Address Hold Time	t_{AH}	Figs. 1,2	10	—	—	ns

FIG 1. WRITE OPERATION



(WRITE DATA FROM MPU TO MODULE)

FIG 2. READ OPERATION



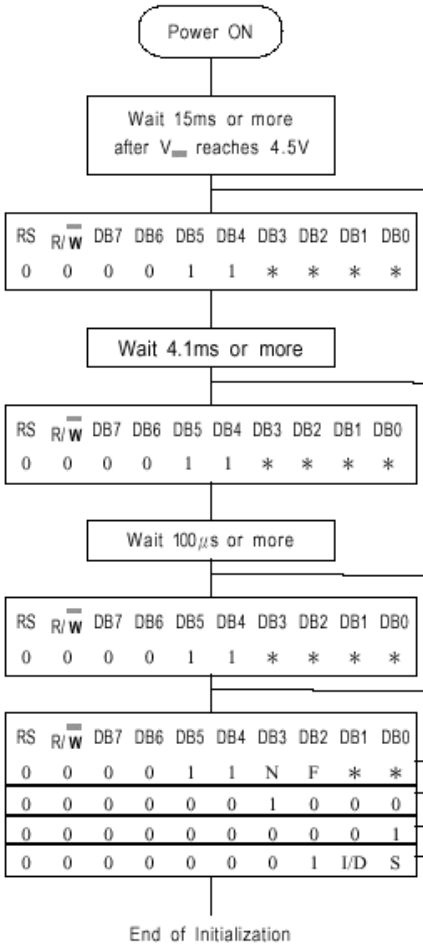
(READ DATA FROM MODULE TO MPU)

Instruction	Code										Description	Execution Time (max) (when fcp or fosc is 250KHz)
	RS	R/$\overline{W}$	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	Clears entire display and sets DD RAM address 0 in address counter.	1.64 ms
Return Home	0	0	0	0	0	0	0	0	1	*	Sets DD RAM address 0 in address counter. Also returns display being shifted to original position. DD RAM contents remain unchanged.	1.64 ms
Entry Mode Set	0	0	0	0	0	0	0	1	1/D	S	Sets cursor move direction and specifies shift of display. These operations are performed during data write and read.	40 μs
Display On / Off Control	0	0	0	0	0	0	1	D	C	B	Sets ON/OFF of entire display (D), Cursor ON/OFF (C),and blink of cursor position character (B).	40 μs
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	*	*	Moves cursor & shifts display without changing DD RAM contents.	40 μs
Function Set	0	0	0	0	1	DL	N	F	*	*	Sets interface data length (DL), number of display lines (L) and character fonts (F).	40 μs
Set CG RAM Address	0	0	0	1	ACG						Sets CG RAM address. CG RAM data is sent and received after this setting.	40 μs
Set DD RAM Address	0	0	1	ADD							Sets DD RAM address. CG RAM data is sent and received after this setting.	40 μs
Read Busy Flag and Address	0	1	BF	AC							Reads Busy flag (BF) indicating internal operation is being performed and reads address counter contents.	0 μs
Write Data to CG or DD RAM	0	0	Write Data								Writes data into DD RAM or CG RAM	40 μs
Read Data from CG or DD RAM	0	0	Read Data								Reads data into DD ram or CG RAM	40 μs
	I/D = 1:Increment I/D = 0:Decrement S = 1:Accompanies display shift S/C = 1:Display shift S/C = 0:Cursor move R/L = 1:Shift to the right R/L = 0:Shift to the left DL = 1:8 bits, DL = 0:4 bits N = 1:2 Lines, N = 0:1 line F = 1:5x 10 dots, F=0:5x 7 dots FB = 1:Internally operating FB = 0:Can accept instruction										DD RAM: Display data RAM CG RAM: Character generator RAM ACG: CG RAM Address ADD: DD RAM Address : Corresponds to cursor address AC: Address counter used for both DD and CG RAM address.	Execution time changes when frequency changes Example: When fcp or fosc is 270 KHz: 40 μs$\times$ $^{250}/_{270}$=37 μs

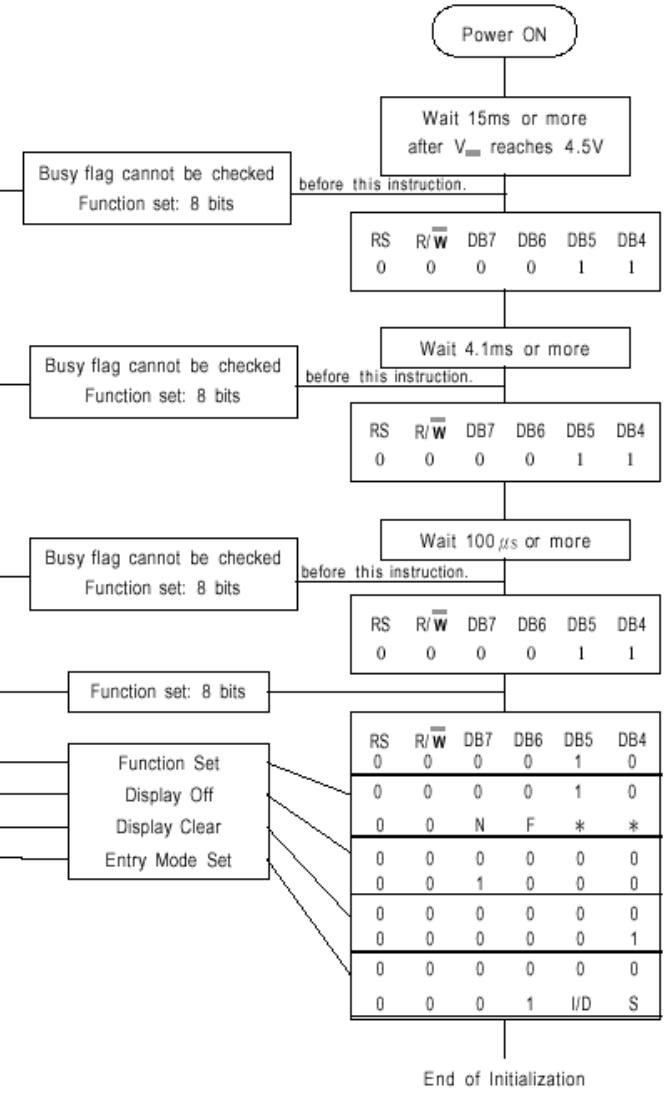
If the power supply conditions for correctly operating the internal reset circuit are not met, initialization by instruction is required, or use the following procedure for initialization.

Instructions

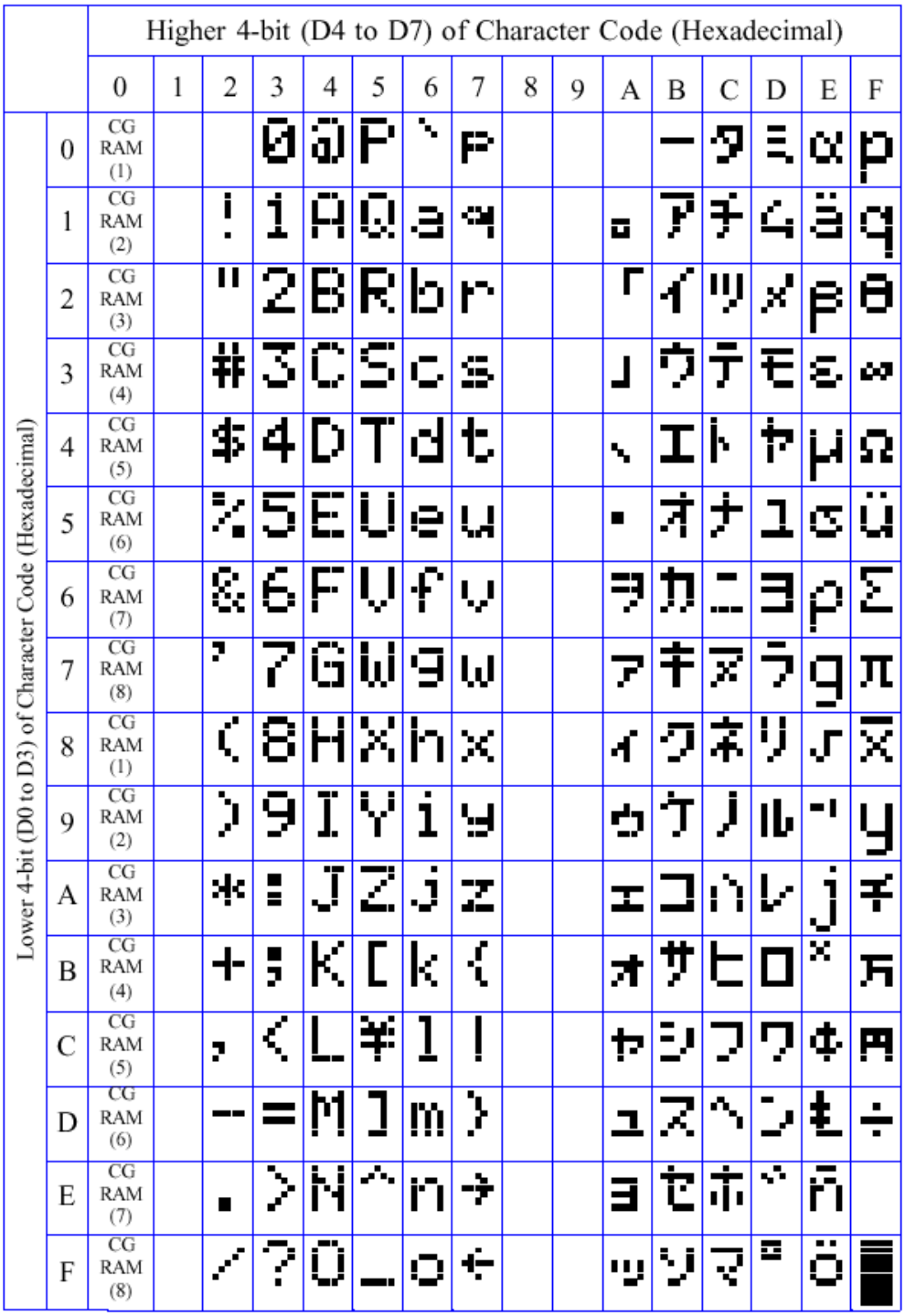
1) 8 Bit Interface



2) 4 bit Interface



· Busy flag be checked after following instructions are completed. If busy flag is not checked, the waiting time between instructions should be longer than the execution time of these instructions.


Higher 4-bit (D4 to D7) of Character Code (Hexadecimal)
0 1 2 3 4 5 6 7 8 9 A B C D E F
Lower 4-bit (D0 to D3) of Character Code (Hexadecimal)
0 CG RAM (1)
1 CG RAM (2)
2 CG RAM (3)
3 CG RAM (4)
4 CG RAM (5)
5 CG RAM (6)
6 CG RAM (7)
7 CG RAM (8)
8 CG RAM (1)
9 CG RAM (2)
A CG RAM (3)
B CG RAM (4)
C CG RAM (5)
D CG RAM (6)
E CG RAM (7)
F CG RAM (8)

Electrical Characteristics

Item	Symbol	Condition	Min	Typ	Max
Input Voltage (high)	Vih	H Level	2.2 V	—	Vdd
Input Voltage (low)	Vil	L Level	0 V	—	0.6 V
Recommended LCD Driving Voltage (Standard Temp.)	Vdd - Vo	0°C	—	4.8 V	5.4 V
		25°C	4.2 V	4.6 V	—
		50°C	3.9 V	4.3 V	—
Recommended LCD Driving Voltage (Wide Temp.)	Vdd - Vo	-20°C	—	6.4 V	7.2 V
		0°C	—	4.8 V	—
		50°C	—	4.3 V	—
		70°C	3.7 V	4.2 V	—
Power Supply Current	Idd	Vdd = 5.0 V ν = 270 kHz	—	0.5 mA	1.0 mA
LED Backlight Voltage	Vf	R = 6.8 Ω	—	4.6 V	5.0 V
LED Backlight Current	If	R = 6.8 Ω	—	240 mA	480 mA

SCHEMATICS

This page consists of links to the schematics of interest to an engineer using the RabbitFLEX BL300F.

090-0219 RabbitFLEX BL300F General Schematic

www.rabbit.com/documentation/schemat/090-0219.pdf

090-0193 PowerCore 3800 Module Schematic

www.rabbit.com/documentation/schemat/090-0193.pdf

090-0193 PowerCore 3810 Module Schematic

www.rabbit.com/documentation/schemat/090-0193.pdf

090-0042 Demonstration Board Schematic

www.rabbit.com/documentation/schemat/090-0042.pdf

090-0128 Programming Cable Schematic

www.rabbit.com/documentation/schemat/090-0128.pdf

Index

L

M

N

O

P

R

S

T

U

W