Digi - 20-101-1195 Datasheet

Welcome to E-XFL.COM

Understanding Embedded - Microcontroller, Microprocessor, FPGA Modules

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

Applications of Embedded - Microcontroller,

Details	
Product Status	Obsolete
Module/Board Type	MPU Core
Core Processor	Rabbit 3000
Co-Processor	-
Speed	44.2MHz
Flash Size	512KB (Internal), 4MB (External)
RAM Size	1MB
Connector Type	2 IDC Headers 2x17, 1 IDC Header 2x5
Size / Dimension	1.85" x 2.73" (47mm x 69mm)
Operating Temperature	-40°C ~ 85°C
Purchase URL	https://www.e-xfl.com/product-detail/digi-international/20-101-1195

1.3 Advantages of the RCM3309 and RCM3319

- Fast time to market using a fully engineered, “ready-to-run/ready-to-program” microprocessor core.
- Competitive pricing when compared with the alternative of purchasing and assembling individual components.
- Easy C-language program development and debugging
- Program download utility (Rabbit Field Utility) and cloning board options for rapid production loading of programs.
- Generous memory size allows large programs with tens of thousands of lines of code, and substantial data storage.
- Integrated Ethernet port for network connectivity, with royalty-free TCP/IP software.
- Ideal for network-enabling security and access systems, home automation, HVAC systems, and industrial controls

If Dynamic C appears to compile the BIOS successfully, but you then receive a communication error message when you compile and load a sample program, it is possible that your PC cannot handle the higher program-loading baud rate. Try changing the maximum download rate to a slower baud rate as follows.

- Locate the **Serial Options** dialog on the “Communications” tab in the Dynamic C **Options > Project Options** menu. Select a slower Max download baud rate. Click **OK** to save.

If a program compiles and loads, but then loses target communication before you can begin debugging, it is possible that your PC cannot handle the default debugging baud rate. Try lowering the debugging baud rate as follows.

- Locate the **Serial Options** dialog in the Dynamic C **Options > Project Options > Communications** menu. Choose a lower debug baud rate. Click **OK** to save.

Press **<Ctrl-Y>** to force Dynamic C to recompile the BIOS. The LEDs on the USB programming cable will blink and you should receive a `Bios compiled successfully` message.

2.5 Where Do I Go From Here?

If the sample program ran fine, you are now ready to go on to other sample programs and to develop your own applications. The source code for the sample programs is provided to allow you to modify them for your own use. The ***RCM3309/RCM3319 User’s Manual*** also provides complete hardware reference information and describes the software function calls for the RCM3309 and the RCM3319, the Prototyping Board, and the optional LCD/keypad module.

For advanced development topics, refer to the ***Dynamic C User’s Manual*** and the ***Dynamic C TCP/IP User’s Manual***, also in the online documentation set.

2.5.1 Technical Support

NOTE: If you purchased your RCM3309/RCM3319 through a distributor or through a Rabbit partner, contact the distributor or partner first for technical support.

If there are any problems at this point:

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

Figure 6 shows the use of the Rabbit 3000 microprocessor ports in the RCM3309/ RCM3319 modules.

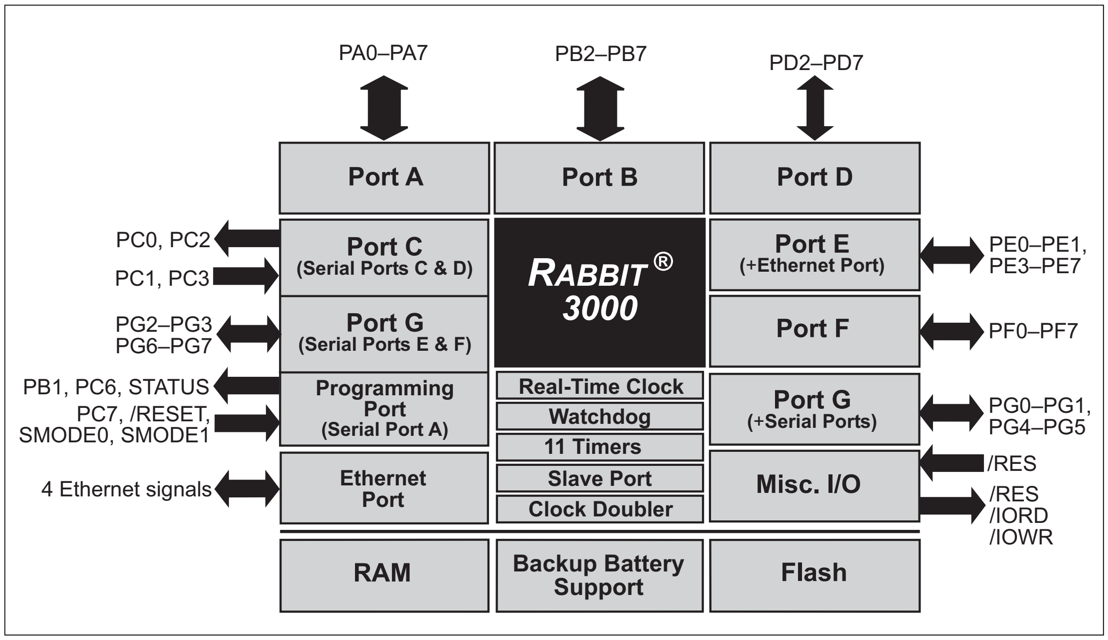

Figure 6. Use of Rabbit 3000 Ports

The ports on the Rabbit 3000 microprocessor used in the RCM3309/RCM3319 are configurable, and so the factory defaults can be reconfigured. Table 2 lists the Rabbit 3000 factory defaults and the alternate configurations.

5.1.1 Developing Programs Remotely with Dynamic C

Dynamic C is an integrated development environment that allows you to edit, compile, and debug your programs. Dynamic C has the ability to allow programming over the Internet or local Ethernet. This is accomplished in one of two ways.

1. Via the RabbitLink, which allows a Rabbit-based target to have programs downloaded to it and debugged with the same ease as exists when the target is connected directly to a PC.
2. The RCM3309/RCM3319 has a featured remote application update written specifically to allow the RCM3309/RCM3319 to be programmed over the Internet or local Ethernet. These programs, **`DLP_STATIC.C`** and **`DLP_WEB.C`**, are available in the Dynamic C **`SAMPLES\RCM3300\RemoteApplicationUpdate`** folder. Complete information on the use of these programs is provided in the ***Remote Application Update*** instructions, which are available with the online documentation.

Dynamic C provides sample programs to illustrate the use of a download manager.

digOut

```
void digOut(int channel, int value);
```

DESCRIPTION

Writes a value to an output channel on Prototyping Board header J10. Do not use this function if you have installed the stepper motor chips at U2 and U3.

PARAMETERS

`channel`	output channel 0–7 (OUT00–OUT07).
`value`	value (0 or 1) to output.

RETURN VALUE

None.

SEE ALSO

`brdInit`

5.2.6.5 RabbitNet Port

The function calls described in this section are used to configure the RabbitNet port on the Prototyping Board for use with RabbitNet peripheral cards. The user's manual for the specific peripheral card you are using contains additional function calls related to the RabbitNet protocol and the individual peripheral card. Appendix 0 provides additional information about the RabbitNet.

These RabbitNet peripheral cards are available at the present time.

- Digital I/O Card (RN1100)
- A/D Converter Card (RN1200)
- D/A Converter Card (RN1300)
- Relay Card (RN1400)
- Keypad/Display Interface (RN1600)

Before using the RabbitNet port, add the following lines at the start of your program.

```
#define RN_MAX_DEV 10  // max number of devices
#define RN_MAX_DATA 16 // max number of data bytes in any transaction
#define RN_MAX_PORT 2  // max number of serial ports
```

Set the following bits in **`RNSTATUSABORT`** to abort transmitting data after the status byte is returned. This does not affect the status byte and still can be interpreted. Set any bit combination to abort:

bit 7—device busy is hard-coded into driver
bit 5—identifies router or slave
bits 4,3,2—peripheral-board-specific bits
bit 1—command rejected
bit 0—watchdog timeout

```
#define RNSTATUSABORT 0x80
     // hard-coded driver default to abort if the peripheral board is busy
```

rn_sp_info

`void rn_sp_info();`

DESCRIPTION

Provides **`rn_init()`** with the serial port control information needed for RCM3309/RCM3319 modules.

RETURN VALUE

None.

6.4.1 How to Set IP Addresses in the Sample Programs

With the introduction of Dynamic C 7.30 we have taken steps to make it easier to run many of our sample programs. You will see a **TCPCONFIG** macro. This macro tells Dynamic C to select your configuration from a list of default configurations. You will have three choices when you encounter a sample program with the **TCPCONFIG** macro.

1. You can replace the **TCPCONFIG** macro with individual **MY_IP_ADDRESS**, **MY_NET-MASK**, **MY_GATEWAY**, and **MY_NAMESERVER** macros in each program.
2. You can leave **TCPCONFIG** at the usual default of 1, which will set the IP configurations to **10.10.6.100**, the netmask to **255.255.255.0**, and the nameserver and gateway to **10.10.6.1**. If you would like to change the default values, for example, to use an IP address of **10.1.1.2** for the RCM3309/RCM3319 board, and **10.1.1.1** for your PC, you can edit the values in the section that directly follows the "General Configuration" comment in the **TCP_CONFIG.LIB** library. You will find this library in the **LIB\TCPIP** directory.
3. You can create a **CUSTOM_CONFIG.LIB** library and use a **TCPCONFIG** value greater than 100. Instructions for doing this are at the beginning of the **TCP_CONFIG.LIB** library in the **LIB\TCPIP** directory.

There are some other "standard" configurations for **TCPCONFIG** that let you select different features such as DHCP. Their values are documented at the top of the **TCP_CONFIG.LIB** library in the **LIB\TCPIP** directory. More information is available in the ***Dynamic C TCP/IP User's Manual***.

A.5 Jumper Configurations

Figure A-5 shows the jumper locations used to configure the various RCM3309/RCM3319 options. The black square indicates pin 1.

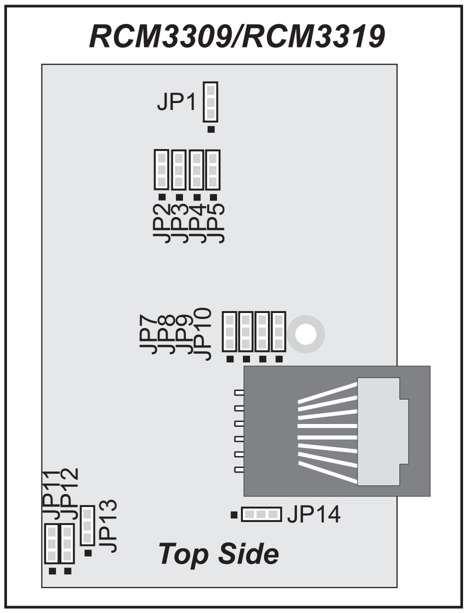

Figure A-5. Location of RCM3309/RCM3319 Configurable Positions

Table A-8 lists the configuration options.

Table A-8. RCM3309/RCM3319 Jumper Configurations

Header	Description	Pins Connected		Factory Default
JP1	Serial Flash Chip Enable Indicator	1–2		×
JP2	ACT or PD1 Output on J61 pin 34	1–2	ACT	×
		2–3	PD1	
JP3	LINK or PD0 Output on J61 pin 33	1–2	LINK	×
		2–3	PD0	
JP4	ENET or PE0 Output on J62 pin 19	1–2	ENET	
		2–3	PE0	×
JP5	NAND Flash Chip Enable	1–2	Reserved for future use	n.c.
		2–3	PD1 controls NAND Flash	

B.3 Power Supply

The RCM3309/RCM3319 requires a regulated 3.15 V to 3.45 V DC power source to operate. Depending on the amount of current required by the application, different regulators can be used to supply this voltage.

The Prototyping Board has an onboard +5 V switching power regulator from which a +3.3 V linear regulator draws its supply. Thus both +5 V and +3.3 V are available on the Prototyping Board.

The Prototyping Board itself is protected against reverse polarity by a diode at D1 as shown in Figure B-3.

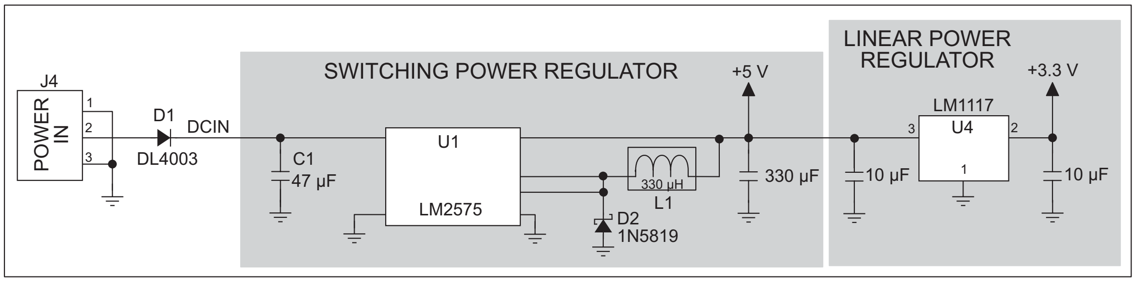

Figure B-3. Prototyping Board Power Supply

Figure B-11 shows the stepper-motor driver circuit.

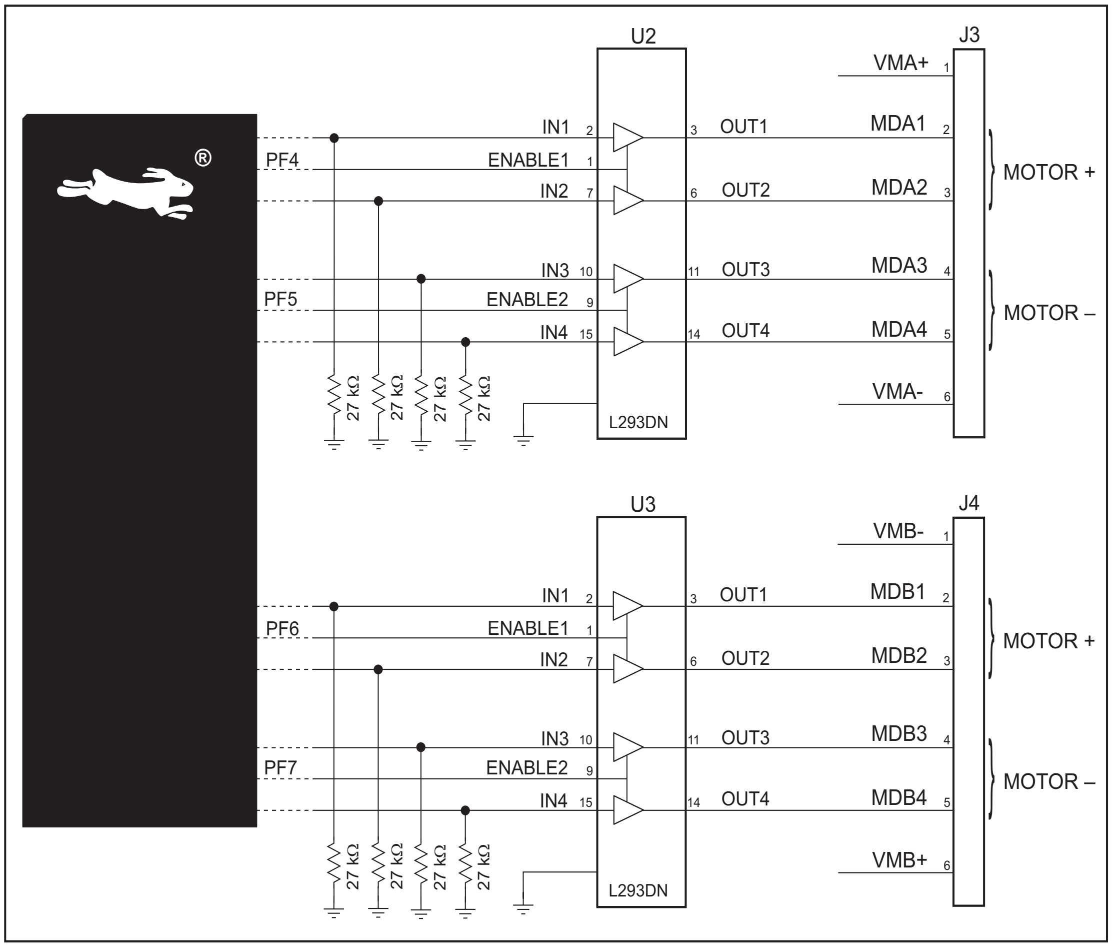

Figure B-11. Stepper-Motor Driver Circuit

The stepper motor(s) can be powered either from the onboard power supply or from an external power based on the jumper settings on headers JP1 and JP2.

Table B-3. Stepper Motor Power-Supply Options

Header	Pins Connected		Factory Default
JP1	1–2 9–10	Onboard power supply to U2	×
	3–4 7–8	External power supply to U2	
JP2	1–2 9–10	Onboard power supply to U3	×
	3–4 7–8	External power supply to U3	

C.3 Keypad Labeling

The keypad may be labeled according to your needs. A template is provided in Figure C-4 to allow you to design your own keypad label insert.

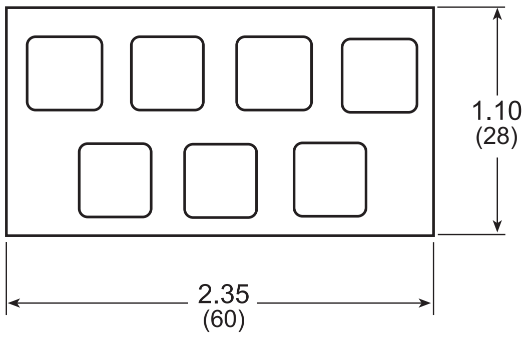

Figure C-4. Keypad Template

To replace the keypad legend, remove the old legend and insert your new legend prepared according to the template in Figure C-4. The keypad legend is located under the blue keypad matte, and is accessible from the left only as shown in Figure C-5.

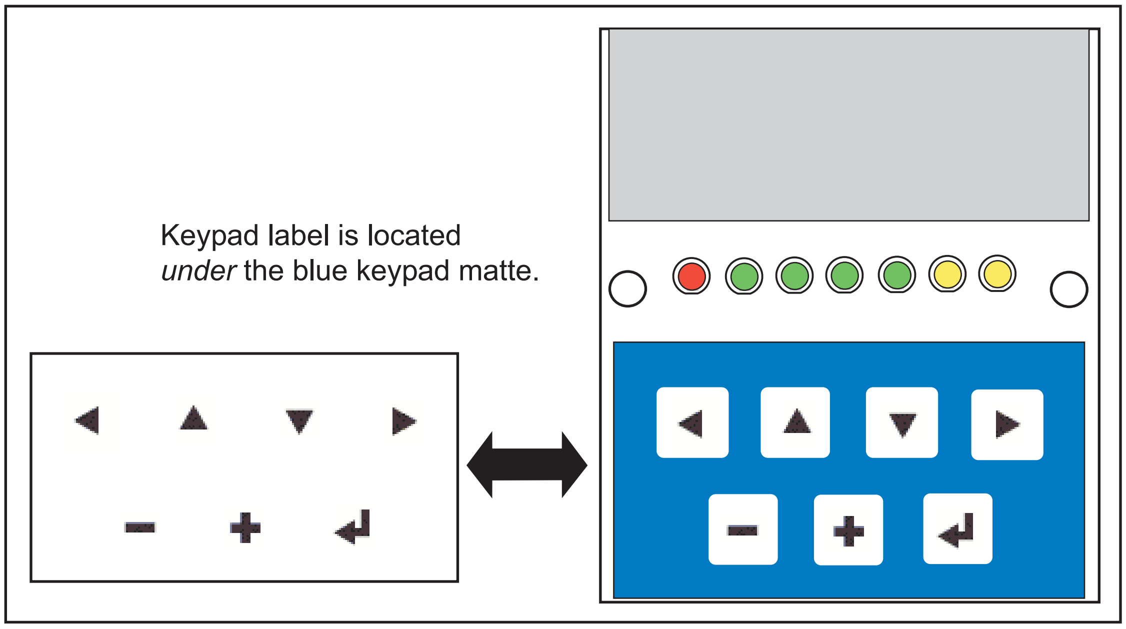

Figure C-5. Removing and Inserting Keypad Label

The sample program **`KEYBASIC.C`** in the **`122x32_1x7`** folder in **`SAMPLES\LCD_KEYPAD`** shows how to reconfigure the keypad for different applications.

C.5 Mounting LCD/Keypad Module on the Prototyping Board

Install the LCD/keypad module on header sockets LCD1JA, LCD1JB, and LCD1JC of the Prototyping Board as shown in Figure C-7. Be careful to align the pins over the headers, and do not bend them as you press down to mate the LCD/keypad module with the Prototyping Board.

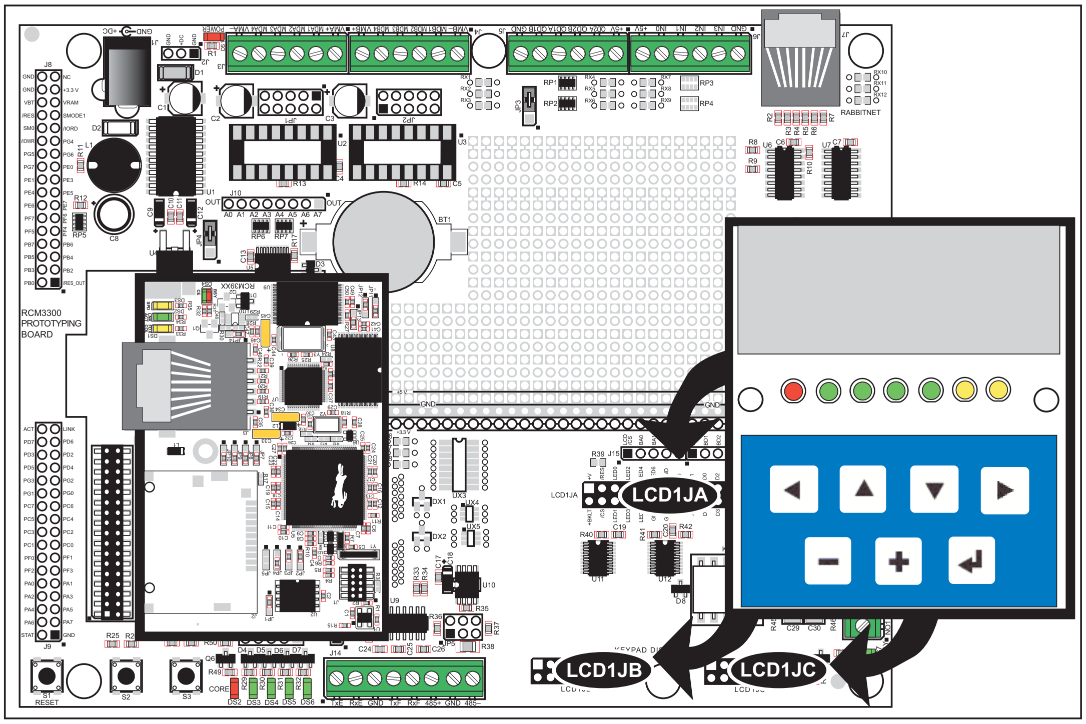

Figure C-7. Install LCD/Keypad Module on Prototyping Board

C.7 Sample Programs

Sample programs illustrating the use of the LCD/keypad module with the Prototyping Board are provided in the **`SAMPLES\RCM3300\LCD_KEYPAD`** folder.

These sample programs use the external I/O bus on the Rabbit 3000 chip, and so the **`#define PORTA_AUX_IO`** line is already included in the sample programs.

Each sample program has comments that describe the purpose and function of the program. Follow the instructions at the beginning of the sample program. To run a sample program, open it with the **File** menu (if it is not still open), then compile and run it by pressing **F9**. The RCM3309/RCM3319 must be connected to a PC using the programming cable as described in Chapter 2, “Getting Started.”

Complete information on Dynamic C is provided in the ***Dynamic C User’s Manual***.

- **`KEYPADTOLED.C`**—This program demonstrates the use of the external I/O bus. The program will light up an LED on the LCD/keypad module and will display a message on the LCD when a key press is detected. The DS3, DS4, DS5, and DS6 LEDs on the Prototyping Board and the red **BSY** LED (DS4) on the RCM3309/RCM3319 module will also light up.
- **`LCDKEYFUN.C`**—This program demonstrates how to draw primitive features from the graphic library (lines, circles, polygons), and also demonstrates the keypad with the key release option.
- **`SWITCHTOLCD.C`**—This program demonstrates the use of the external I/O bus. The program will light up an LED on the LCD/keypad module and will display a message on the LCD when a switch press is detected. The DS1 and DS2 LEDs on the Prototyping Board will also light up.

Additional sample programs are available in the **`SAMPLES\LCD_KEYPAD\122x32_1x7`** folder.

`glFontCharAddr`

```
unsigned long glFontCharAddr(fontInfo *pInfo, char letter);
```

DESCRIPTION

Returns the **`xmem`** address of the character from the specified font set.

PARAMETERS

`pInfo`	pointer to the **`xmem`** address of the bitmap font set.
`letter`	an ASCII character.

RETURN VALUE

`xmem` address of bitmap character font, column major and byte-aligned.

SEE ALSO

`glPutFont, glPrintf`

`glPutFont`

```
void glPutFont(int x, int y, fontInfo *pInfo, char code);
```

DESCRIPTION

Puts an entry from the font table to the page buffer and on the LCD if the buffer is unlocked. Each font character's bitmap is column major and byte-aligned. Any portion of the bitmap character that is outside the LCD display area will be clipped.

PARAMETERS

`x`	the *x* coordinate (column) of the top left corner of the text.
`y`	the *y* coordinate (row) of the top left corner of the text.
`pInfo`	a pointer to the font descriptor.
`code`	the ASCII character to display.

RETURN VALUE

None.

SEE ALSO

`glFontCharAddr, glPrintf`

glPrintf

```
void glPrintf(int x, int y, fontInfo *pInfo, char *fmt, ...);
```

DESCRIPTION

Prints a formatted string (much like **`printf`**) on the LCD screen. Only the character codes that exist in the font set are printed, all others are skipped. For example, '\b', '\t', '\n' and '\r' (ASCII backspace, tab, new line, and carriage return, respectively) will be printed if they exist in the font set, but will not have any effect as control characters. Any portion of the bitmap character that is outside the LCD display area will be clipped.

PARAMETERS

`x`	the *x* coordinate (column) of the upper left corner of the text.
`y`	the *y* coordinate (row) of the upper left corner of the text.
`pInfo`	a pointer to the font descriptor.
`fmt`	pointer to a formatted string.
`...`	formatted string conversion parameter(s).

EXAMPLE

```
glprintf(0,0, &fi12x16, "Test %d\n", count);
```

RETURN VALUE

None.

SEE ALSO

`glXFontInit`

glBuffLock

```
void glBuffLock(void);
```

DESCRIPTION

Increments LCD screen locking counter. Graphic calls are recorded in the LCD memory buffer and are not transferred to the LCD if the counter is non-zero.

NOTE: `glBuffLock()` and `glBuffUnlock()` can be nested up to a level of 255, but be sure to balance the calls. It is not a requirement to use these procedures, but a set of `glBuffLock()` and `glBuffUnlock()` bracketing a set of related graphic calls speeds up the rendering significantly.

RETURN VALUE

None.

SEE ALSO

`glBuffUnlock, glSwap`

glBuffUnlock

```
void glBuffUnlock(void);
```

DESCRIPTION

Decrements the LCD screen locking counter. The contents of the LCD buffer are transferred to the LCD if the counter goes to zero.

RETURN VALUE

None.

SEE ALSO

`glBuffLock, glSwap`

glPlotDot

`void glPlotDot(int x, int y);`

DESCRIPTION

Draws a single pixel in the LCD buffer, and on the LCD if the buffer is unlocked. If the coordinates are outside the LCD display area, the dot will not be plotted.

PARAMETERS

`x`	the *x* coordinate of the dot.
`y`	the *y* coordinate of the dot.

RETURN VALUE

None.

SEE ALSO

`glPlotline, glPlotPolygon, glPlotCircle`

glPlotLine

`void glPlotLine(int x0, int y0, int x1, int y1);`

DESCRIPTION

Draws a line in the LCD buffer, and on the LCD if the buffer is unlocked. Any portion of the line that is beyond the LCD display area will be clipped.

PARAMETERS

`x0`	the *x* coordinate of one endpoint of the line.
`y0`	the *y* coordinate of one endpoint of the line.
`x1`	the *x* coordinate of the other endpoint of the line.
`y1`	the *y* coordinate of the other endpoint of the line.

RETURN VALUE

None.

SEE ALSO

`glPlotDot, glPlotPolygon, glPlotCircle`

glXPutBitmap

```
void glXPutBitmap(int left, int top, int width, int height,
   unsigned long bitmap);
```

DESCRIPTION

Draws bitmap in the specified space. The data for the bitmap are stored in **`xmem`**. This function calls **`glXPutFastmap()`** automatically if the bitmap is byte-aligned (the left edge and the width are each evenly divisible by 8).

Any portion of a bitmap image or character that is outside the LCD display area will be clipped.

PARAMETERS

`left`	the top left corner of the bitmap.
`top`	the top left corner of the bitmap.
`width`	the width of the bitmap.
`height`	the height of the bitmap.
`bitmap`	the address of the bitmap in **`xmem`**.

RETURN VALUE

None.

SEE ALSO

`glXPutFastmap, glPrintf`

glXPutFastmap

```
void glXPutFastmap(int left, int top, int width, int height,
   unsigned long bitmap);
```

DESCRIPTION

Draws bitmap in the specified space. The data for the bitmap are stored in **`xmem`**. This function is like **`glXPutBitmap()`**, except that it is faster. The restriction is that the bitmap must be byte-aligned.

Any portion of a bitmap image or character that is outside the LCD display area will be clipped.

PARAMETERS

`left`	the top left corner of the bitmap, must be evenly divisible by 8, otherwise truncates.
`top`	the top left corner of the bitmap.
`width`	the width of the bitmap, must be evenly divisible by 8, otherwise truncates.
`height`	the height of the bitmap.
`bitmap`	the address of the bitmap in **`xmem`**.

RETURN VALUE

None.

SEE ALSO

`glXPutBitmap, glPrintf`

`rn_sw_wdt`

`int rn_sw_wdt(int handle, float timeout);`

DESCRIPTION

Sets software watchdog timeout period. Call this function prior to enabling the software watchdog timer. This function will check device information to determine that the peripheral card is connected to a master.

PARAMETERS

`handle`	is an address index to device information. Use **`rn_device()`** or **`rn_find()`** to establish the handle.
`timeout`	is a timeout period from 0.025 to 6.375 seconds in increments of 0.025 seconds. Entering a zero value will disable the software watchdog timer.

RETURN VALUE

The status byte from the previous command.

-1 means that device information indicates the peripheral card is not connected to the master.