CL-PS7111 Development Kit

Software User's Guide

Embedded Processors Division

This document describes sample code for the CL-PS7111 provided by Cirrus Logic Inc. No warranty is given for the suitability of the program code described herein for any purpose other than demonstrating functional operation of the CL-PS7111. The information contained in this document is subject to change without notice.

Version 1.3 April 1999

Table of Contents

1. Introduction

The CL-PS7111 development kit example software is targeted at software developers who plan to port operating systems and applications to the CL-PS7111. A library of routines is provided which configure and operate all of the peripherals on the CL-PS7111 evaluation board. Additionally, there is a set of sample programs that use this library to exercise the peripherals on the board.

To use the CL-PS7111 evaluation kit example software, the ARM® SDT Version 2.11a containing the project manager, C compiler, assembler, linker, debugger, and ARM instruction set emulator is required. The code in this kit is intended to be used on a PC running Windows® 95, Windows NT® 3.51, or Windows NT 4.0.

2. ARM Software Development Toolkit

Included on the CD-ROM is a 60-day evaluation version of the ARM SDT Version 2.11a. This is a fully functional evaluation copy of the SDT that will cease to execute after 60 days. To obtain the full version of the ARM SDT, contact ARM Ltd. (www.arm.com).

To install the ARM SDT, simply run setup.exe from the *sdt211a* directory on the CD-ROM and follow the on-screen directions. If you are unfamiliar with the ARM SDT, be sure to select the online manuals as one of the components to install.

The evaluation version of the SDT will create a seemingly useless directory called "c_dilla"; if you remove this directory or its contents, the evaluation version of the SDT will no longer work, will not uninstall correctly, and will not be able to be installed again on your system.

Once the SDT has been installed, there are two applications that will be used to build and debug applications: the ARM Project Manager and the ARM Debugger. These applications are described fully in the online manuals, but brief instructions for basic use of these applications is provided in the following sections.

2.1. ARM Project Manager

The ARM Project Manager is used to develop and build applications for the CL-PS7111 evaluation board. It is found in the "Start" menu under "Programs" then "ARM SDT v2.11a". Once run, it will show an empty workspace. The first thing to do is to open a project file (which has an extension of ".apj"). This is done by selecting "Open" from the "File" menu.

Once a project has been opened, there are several basic operations that you might perform. They are:

1) Edit the source code. To do this, click on the "+" beside the "ARM Executable Image" line in the project window, then the "+" next to the "Debug" line, then the "+" next to the "Sources" line, and then double click on the source file to be edited. The source file will be opened in another window. Standard editing operations can be performed on the source code in this window.

2) Build the executable. To do this, click on the "Build Debug" button at the bottom of the project window. You can build either a debug or release version of the executable; selecting the "Debug" or "Release" line from the project window will change the button at the bottom to "Build Debug" or "Build Release" respectively. Alternately, you can select "Build" from the "Project" menu or press Shift+F8 to build the executable. The debug version of an executable contains all the symbolic information needed to debug the executable, while the release version contains no symbolic information and is fully optimized.

3) Debug the executable. To do this, select "Debug" from the "Project" menu, or press F5. This will launch the ARM Debugger, load the executable image, and set a breakpoint at the beginning of the "main" function.

4) Run the executable. To do this, select "Execute" from the "Project" menu, or press Ctrl+F5. This will launch the ARM Debugger, load the executable image, and begin execution of the program.

2.2. ARM Debugger

The ARM Debugger is used to debug applications on the CL-PS7111 evaluation board. It is found in the "Start" menu under "Programs" then "ARM SDT v2.11a". Once run, it will show three windows, the "Execution Window", "Command Window", and "Console Window". The execution window shows the current execution context of the target system. The command window allows you to type commands to the debugger, though you will probably never use this since the functionality can be more easily accessed via the GUI. The console window shows the output from the target system, including anything printed out from your application.

If the ARM Project Manager launches the ARM Debugger, the image to be debugged will be automatically loaded into the target system's memory. Otherwise, the image to be debugged must be loaded manually by selecting "Load Image" from the "File" menu.

Once an image has been loaded into the target system's memory, there are several basic operations that you might perform. They are:

1) View the source files in the program image. Select "Source Files" from the "View" menu, or press Ctrl+F, and a window will be displayed showing all the source files in the program image. Double clicking on one of the file names in the "Source Files" window will open a new window that will display the selected source file.

2) Set a breakpoint. In either the "Execution Window" or a source file window, place the cursor on the line where you want to place a breakpoint. Then select "Toggle Breakpoint" from the "Execute" menu, or press F9. Performing the same action after selecting a line that already has a breakpoint on it will remove the breakpoint.

3) Run the program. Select "Go" from the "Execute" menu, or press F5. The program will begin executing. It will run until a breakpoint is reached, the program terminates, or a fatal error occurs.

4) Single-step the program. The program can be single-stepped in three different ways. You can step to the next point within the program execution, stepping into any procedures called, by selecting "Step In" from the "Execute" menu, or pressing F8. Think of this as stepping into the called procedure. You can step to the next point within the program execution, allowing any called procedure to execute to completion, by selecting "Step" from the "Execute" menu, or pressing F10. Think of this as stepping over any called procedures. Finally, you can step to the first point within the program execution after the current routine has completed executing by selecting "Step Out" from the "Execute" menu, or pressing Shift+F7. Think of this as stepping out of the current procedure.

5) Run the program to a specific line. This is roughly equivalent to setting a breakpoint, running the program, and then removing the breakpoint. Once a line has been selected in the "Execution Window" or a source file window, select "Run to cursor" from the "Execute" menu, or press F7.

6) View the ARM registers. Select "Registers\Current Mode" from the "View" menu. This will display a window with the currently accessible register set. This is the most useful register view for typical use; the other register views would be useful for OS development.

7) View the local variables of the current procedure. Select "Variables\Local" from the "View" menu, or press Ctrl+L. Right clicking on the variables will allow you to change the way the variable's value is displayed, or change the content of the variable. Double clicking on a pointer variable will open a new window showing the contents of the memory pointed to by the variable.

8) View the global variables of the current program. Select "Variables\Global" from the "View" menu, or press Ctrl+G. The global variable window can be manipulated in the exact same way as the local variable window.

9) View the contents of memory. Select "Memory…" from the "View" menu, or press Ctrl+M. A dialog will be displayed allowing you to specify the memory address you wish to view. Once you select "OK", a memory window will be displayed. The memory window will always show 4K worth of data, starting on a 4K boundary, and display the memory contents as hexadecimal 32-bit words.

3. Angel™

The example software provided in the evaluation kit is designed to run under control of the Angel debug monitor from ARM. In order to allow the sample code to handle the IRQ interrupt, the version of Angel provided with the evaluation kit (based on Angel 1.04) has been slightly modified to allow the interrupt to be hooked by application code.

There are two versions of Angel provided with the evaluation kit: one communicates with the host system debugger via UART1 and the other via UART2. The two versions are identical in every other respect. The two versions exist since the IR port uses UART1, and having Angel communicate with the host via UART1 would prevent the use of the IR port. The version that uses UART1 is called cl7111_1.rom and the version that uses UART2 is called cl7111_2.rom. They reside in the *ps7111\angel* directory of the CD-ROM. The evaluation board is shipped with cl7111_2.rom programmed into FLASH bank 0.

Also contained in the *ps7111\angel* directory is the *source* directory, containing the source code for this version of Angel. In order for this source code to be built, it must be placed in the same directory into which the ARM SDT was installed (i.e. in the same directory that contains the "bin", "cl", "demon", "include", "lib", etc. directories of the ARM SDT).

The ARM Project Manager file cl7111.apj under the *source\cl7111.b\apm* directory can be used to build Angel. There are two variants of Angel that can be built: Image1 and Image2. The Image1 variant uses UART1 to communicate with the host debugger and the Image2 variant uses UART2. Once built, the file *source\cl7111.b\apm\image1\cl7111.rom* corresponds to the cl7111_1.rom on the CD-ROM, and the file *source\cl7111.b\apm\image2\cl7111.rom* corresponds to cl7111_2.rom.

Angel enables the MMU on the CL-PS7111 and uses it to rearrange the memory map of the processor. The main reason this is done is to provide an additional mapping of DRAM at location 0x08000000. The stack checking code generated by the ARM compiler produces incorrect results when the stack is located at or above 0x80000000, and DRAM is located at 0xC0000000, so this re-mapping is done to place DRAM below 0x80000000. The memory map under Angel appears as follows:

0xC0000000 – 0xC0FFFFFF	DRAM Bank 0
0x80000000 – 0x80001800	CL-PS7111 registers
0x70000000 – 0x7000007F	On-chip boot ROM
0x60000000 – 0x600007FF	On-chip SRAM
0x50000000 – 0x5FFFFFFF	Expansion
0x40000000 – 0x4FFFFFFF	CL-PS6700 PCMCIA controller
0x30000000 – 0x3FFFFFFF	Keyboard scan latch
0x20000000 – 0x2FFFFFFF	CS8900A Ethernet controller
0x10000000 – 0x107FFFFF	FLASH Bank 0
0x08028000 – 0x08FFFFFF	User program space
0x08020000 – 0x08027FFF	Angel workspace
0x08000000 – 0x0801FFFF	LCD frame buffer
0x00000000 – 0x007FFFFF	FLASH Bank 0

When building an application to run under the control of Angel, the read-only base address should be set to 0x08028000.

4. Download.exe

Download.exe is a program used to write boot code into FLASH bank 0. It can be used to download a new version of Angel into the CL-PS7111 evaluation board. The program is located in the *ps7111* directory on the CD-ROM.

The command line syntax for download.exe is as follows:

download <filename> {<baud rate> {<com port>}}

Where <filename> is the name of the file to be downloaded into FLASH bank 0, <baud rate> is the optionally specified baud rate to use for the data download, and <com port> is the optionally specified COM port on the host PC that is used for communication with the evaluation board. The default baud rate used is 115,200, with 9,600, 19,200, 28,800, 38,400, and 57,600 being the other valid options. The default COM port used is COM1, with COM2, COM3, and COM4 being the other valid options. The COM port is specified as simply the port number (i.e. "2" for COM2).

In order to use download.exe to write boot code into FLASH bank 0, follow these steps:

1) Connect the supplied NULL modem cable between the "UART1" connector on the evaluation board and any available COM port on the host system.

2) Place a shunt on jumper JP2 (located directly above the upper left hand corner of the keyboard).

3) Apply power to the evaluation board.

4) Run download.exe.

5) Press the "reset" button on the evaluation board.

6) Press the "wakeup" button on the evaluation board.

7) Download.exe will display its progress as it programs FLASH bank 0. Wait until it says it is done.

8) Remove power from the evaluation board.

9) Remove the shunt from jumper JP2.

At this point the new boot code is programmed into FLASH bank 0. It can then be used in the normal operation of the board.

5. Board Setup

Follow these steps to setup communication with the evaluation board.

1) Connect the supplied NULL modem cable between the "UART2" connector on the evaluation board and any available COM port on the host system.

2) Apply power to the evaluation board.

3) Press the "reset" button on the evaluation board.

4) Press the "wakeup" button on the evaluation board.

5) Start the ARM Debugger on the host system.

6) Select "Configure Debugger…" from the "Options" menu.

7) Select "remote_a" as the target environment.

8) Click on the "Configure…" button to select the host COM port to be used. To speed the debugging process, select a baud rate of 115,200. Click on "OK" when done.

9) Click on "OK". The ARM Debugger should connect to Angel on the evaluation board and then print out a message similar to the following in the "Console Window":

```
Angel Debug Monitor for CL-PS7111 Eval Kit (Built with Serial(x1-2), no info) 1.04 (Advanced RISC Machines SDT 2.11a)
Angel Debug Monitor rebuilt on Nov 10 1998 at 11:07:40
Serial Rate: 115200
```

If there is a problem (such as a bad serial cable, attempting to use the wrong serial port, etc.), the following message will be displayed:

```
Cannot open target: the target is not responding.
```

The ARM Debugger can be changed between debugging programs on the CL-PS7111 evaluation board (via Angel) and the ARMulator software ARM emulator at any time. Simply select "Configure Debugger…" from the "Options" menu, select "remote_a" (for the CL-PS7111 evaluation board) or "ARMulate" for the ARMulator as the target environment, and then click on "OK".

If there are any communication problems between the host and the CL-PS7111 evaluation board, the ARM tools might revert to using the ARMulator as a result of an obscurely worded dialog box. If this happens, you will have to use the "Configure Debugger…" option to switch back to debugging on the CL-PS7111 evaluation board.

6. Lib7111

Lib7111 is a library of routines for accessing the various peripherals on the CL-PS7111 evaluation board. These routines are not necessarily the most efficient in terms of execution time or code space, they simply demonstrate the proper operation of the peripherals. The library is divided into separate source files for each peripheral. The lib7111.h file contains prototypes for all the functions in the library. The lib7111.apj file is the ARM Project Manager project file used to build the debug and release variants of the library.

The source code for this library is contained in the *ps7111\lib7111* directory of the CD-ROM.

6.1. adc.c

This file contains routines for using the synchronous serial interface of the CL-PS7111. The SSI is a master mode only SPI/Microwire compatible interface; on the evaluation board it is connected to a Maxim MAX148 10-bit ADC. The MAX148 is powered on and off via bit two of GPIO port E; the ADC is powered on when the bit is driven high and powered off when the bit is driven low.

6.1.1. ADCDisable

```
void ADCDisable(void)
```

This routine powers off the external MAX148 ADC.

6.1.2. ADCEnable

```
void ADCEnable(void)
```

This routine configures the SSI for 128kHz operation and powers on the external MAX148 ADC.

6.1.3. ADCGetData

```
long ADCGetData(long lChannel)
```

This routine tells the MAX148 ADC to sample data on the specified channel and returns the value of the sampled data. The following are valid channel values, along with the data that each channel will sample:

0	unused
1	The VDD voltage.
2	The TSPX value from the touch screen interface.
3	The TSMX value from the touch screen interface.

4 The potentiometer at R47. This potentiometer is not populated on the evaluation board; populate it with a 500KΩ potentiometer if desired.

5 The VREF voltage.

6 The TSMY value from the touch screen interface.

7 The TSPY value from the touch screen interface.

6.2. codec.c

This file contains routines for using the codec interface of the CL-PS7111. The codec interface is a bi-directional 8kHz, 8-bit mono serial codec interface; on the evaluation board it is connected to an Oki MSM7702-01MS telephone codec, which is configured to produce and consume μlaw PCM data. The MSM7702-01MS is powered on and off via bit 0 of GPIO port E; the codec is powered on when the bit is driven high and powered off when the bit is driven low.

The analog output of the Oki codec is connected to an Oki MSC1157MS-K amplifier. The MSC1157MS-K is powered on and off when the codec is powered on and off.

6.2.1. CodecDisable

```
void CodecDisable(void)
```

This routine powers off the internal codec interface and the external MSM7702-01MS and MSC1157MS-K. The codec interrupt is masked and the interrupt handler is removed.

6.2.2. CodecEnable

```
void CodecEnable(void)
```

This routine powers on the internal codec interface and the external MSM7702-01MS and MSC1157MS-K. An interrupt handler is installed to handle the codec interrupts and then the codec interrupt is unmasked. The codec interrupt handler will read all available data from the codec input FIFO, and will fill the codec output FIFO with either playback data (if any exists) or silence.

6.2.3. CodecPlay

```
void CodecPlay(char *pcBuffer, long lLength)
```

This routine plays the given buffer of data via the codec interface. This routine will not return until the entire buffer has been played out to the codec interface.

6.2.4. CodecPlayBg

```
void CodecPlayBg(char *pcBuffer, long lLength)
```

This routine plays the given buffer of data via the codec interface. This routine will return immediately, playing the buffer in the background while other processing continues.

6.2.5. CodecRecord

```
void CodecRecord(char *pcBuffer, long lLength)
```

This routine records data into the specified buffer via the codec interface. This routine will not return until the entire buffer has been filled with data read from the codec interface.

6.3. flash.c

This file contains routines for programming data into the Intel 28F320B3 FLASH memories on the CL-PS7111 evaluation board. The implementation of these routines is specific to the Intel FLASH memory. A sector is an individually eraseable block of memory within the FLASH memory. If there are multiple banks of FLASH memory in the system, it is assumed that they are all identical (from a software point of view).

6.3.1. FlashEraseChip

```
void FlashEraseChip(unsigned long ulFlashBase)
```

This routine performs a chip erase of the FLASH memory that is located at the given base address. When completed, the entire contents of the ROM will be 0xFF.

6.3.2. FlashEraseSector

```
void FlashEraseSector(unsigned long ulFlashBase, long lSector)
```

This routine performs an erase of a sector of the FLASH memory that is located at the given base address. The sector that contains the offset lSector is erased. When completed, the entire contents of that sector of ROM will be 0xFF.

6.3.3. FlashNumSectors

```
long FlashNumSectors(void)
```

Returns the number of sectors in the FLASH memory.

6.3.4. FlashProgramBlock

```
void FlashProgramBlock(unsigned long ulFlashBase, long lOffset,
                       unsigned char *pucData, long lNumBytes)
```

This routine programs data into the FLASH memory that is located at the given base address. The data is programmed at the specified offset into the FLASH memory. The FLASH memory (and therefore this routine) will hang if an attempt is made to program a 0 bit to 1…this can only be accomplished by an erase operation.

6.3.5. FlashSectorInfo

```
long FlashSectorInfo(long lSector, long *plSectorOffset,
                     long *plSectorLength)
```

Returns information about the specified sector of the FLASH memory. The offset to the beginning of the sector and the length of the sector are returned.

6.4. ir.c

This file contains routines for using the IrDA compatible port on the CL-PS7111. The IrDA port is a bi-directional infrared communication port that is capable of data transfers at up to 115200 baud. The IrDA port is implemented as a post-processing of the UART1 data stream; therefore the IrDA port can not be used in conjunction with UART1.

6.4.1. IRCharReady

```
long IRCharReady(void)
```

This routine determines if there is a character ready to be read from the IrDA port. The return value will be non-zero if there is a character waiting to be read.

6.4.2. IRDisable

```
void IRDisable(void)
```

This routine will turn off the power to the internal UART and IrDA port.

6.4.3. IREnable

```
long IREnable(long lDataRate, long lDataBits, long lStopBits,
              long lParity, long lEvenParity)
```

This routine will configure the UART/IrDA port to the specified data rate and format and power on the internal UART and IrDA port. The supported data rates are 115200, 57600, 38400, and 19200 baud. The supported data bits are 5, 6, 7, and 8. The supported stop bits are 1 and 2. If lParity is zero, there is no parity bit. If it is non-zero and lEvenParity is zero, then there is an odd parity bit, otherwise there is an even parity bit. The return value will be zero if the UART or IrDA port has already been configured, or if the data rate and/or format is invalid, and one otherwise.

6.4.4. IRReceiveChar

```
char IRReceiveChar(void)
```

This routine will read a character from the IrDA port and return it to the caller.

6.4.5. IRSendChar

```
void IRSendChar(char cChar)
```

This routine will send a character to the IrDA port.

6.5. isr.c

This file contains routines for handling the IRQ interrupt on the ARM processor in the CL-PS7111. The InterruptHandler routine is the actual interrupt handler; InterruptShell (contained in isrshell.s) calls it. It is responsible for determining the cause of the interrupt, calling the appropriate handler routine (if one has been registered), and performing the necessary steps to clear the interrupt.

6.5.1. InterruptInstallISR

```
void InterruptInstallISR(void)
```

This routine registers the IRQ interrupt handler with Angel so that it can handle any IRQ interrupts to the ARM processor. Calls to this routine are reference counted; if the ISR is already installed, the reference count is simply incremented.

6.5.2. InterruptRemoveISR

```
void InterruptRemoveISR(void)
```

This routine un-registers the IRQ interrupt handler with Angel so that it is no longer called to handle IRQ interrupts to the ARM processor. Calls to this routine are reference counted; the ISR is only un-registered when the reference count reaches zero.

6.5.3. InterruptSetADCHandler

```
void InterruptSetADCHandler(PFNISR pfnADC)
```

This routine is used to register the address of the routine that will be called when the SSI interrupt is the cause of the IRQ interrupt to the ARM processor.

6.5.4. InterruptSetCodecHandler

```
void InterruptSetCodecHandler(PFNISR pfnCodec)
```

This routine is used to register the address of the routine that will be called when the codec interrupt is the cause of the IRQ interrupt to the ARM processor.

6.6. isrshell.s

This file contains a shell routine used in handling the IRQ interrupt on the ARM processor of the CL-PS7111. Since the ARM Procedure Call Standard allows some of the ARM registers to be destroyed by the called routine, and the actual interrupt handler is in C and could therefore destroy some of those registers, the registers must be saved to prevent the interrupted program from being corrupted. The InterruptShell routine in this file takes care of saving and restoring those registers so that the C language interrupt handler does not trash the system. This routine is also tailored to the interrupt sharing mechanism used by the Angel debug monitor and as such would not work in a fully embedded system (i.e. no Angel).

6.7. kbd.c

This file contains routines for reading data from the keyboard row/column array. These routines do not perform any mapping of the row/column values to any kind of character value; they simply determine which switches in the keyboard array are currently closed. The row/column of each button is encoded as a binary number 0cccrrrr, where ccc is the column number and rrrr is the row number.

6.7.1. KbdGetButton

```
char KbdGetButton(void)
```

This routine waits until a button in the keyboard array is pressed and returns the row/column of the button that was pressed. If there are multiple buttons pressed in the keyboard array, the one returned is the lowest in column/row order. This routine will return on the press of the button, not the release.

6.7.2. KbdNoButton

```
void KbdNoButton(void)
```

This routine waits until none of the buttons in the keyboard array are pressed.

6.7.3. KbdRead

```
void KbdRead(char *pcKeysPressed)
```

This routine will scan the entire keyboard array to determine which buttons are currently pressed. A 128-entry array will be filled in with non-zero if the corresponding button is pressed and zero if it is not pressed.

6.8. lcd.c

This file contains generic drawing primitives for use on the LCD screen. None of these routines are aware of the dimensions of the LCD screen or its color depth.

6.8.1. LCDCircle

```
void LCDCircle(long lX, long lY, long lRadius, char cColor)
```

This routine draws a circle.

6.8.2. LCDFillCircle

```
void LCDFillCircle(long lX, long lY, long lRadius, char cColor)
```

This routine draws a filled circle.

6.8.3. LCDLine

```
void LCDLine(long lX1, long lY1, long lX2, long lY2, char cColor)
```

This routine draws a line.

6.8.4. LCDPrintChar

```
void LCDPrintChar(char cChar, long lX, long lY, char cColor)
```

This routine draws an ASCII character.

6.8.5. LCDPrintCharX2

```
void LCDPrintCharX2(char cChar, long lX, long lY, char cColor)
```

This routine draws an ASCII character at twice its normal width and height (i.e. 16x16).

6.8.6. LCDPrintString

```
void LCDPrintString(char *pcBuffer, long lX, long lY, char cColor)
```

This routine draws a string of ASCII characters.

6.8.7. LCDPrintStringX2

```
void LCDPrintString(char *pcBuffer, long lX, long lY, char cColor)
```

This routine draws a string of ASCII characters at twice their normal width and height (i.e. 16x16).

6.9. lcd_alps.c

This file contains routines for using the LCD controller on the CL-PS7111. These routines are specific to the ALPS LRHDD101XA LCD panel (a 640x240-pixel backlit panel) with the frame buffer located at address 0xC0000000. The LCD controller is configured for 16 shade gray-scale operation with a refresh rate of 60Hz. The ALPS LCD panel is powered on and off via bit 2 of GPIO port D; the LCD panel is powered on when the bit is driven high and powered off when the bit is driven low. The reference voltage for the LCD panel is powered on and off via bit 1 of GPIO port D; the LCD bias voltage is enabled when the bit is driven high and disabled when the bit is driven low. The LCD backlighting is powered on and off via bit 3 of GPIO port D; the LCD backlighting is powered on when the bit is driven high and powered off when the bit is driven low.

6.9.1. LCDBacklightOff

```
void LCDBacklightOff(void)
```

This routine turns off the backlighting on the LCD panel.

6.9.2. LCDBacklightOn

```
void LCDBacklightOn(void)
```

This routine turns on the backlighting on the LCD panel.

6.9.3. LCDCls

```
void LCDCls(void)
```

This routine erases the frame buffer.

6.9.4. LCDEnable

```
void LCDEnable(void)
```

This routine configures the LCD controller for a 640x240, 16 shade gray-scale, 60Hz refresh LCD panel.

6.9.5. LCDGetPixel

```
char LCDGetPixel(long lX, long lY)
```

This routine returns the current value of the specified pixel.

6.9.6. LCDOff

```
void LCDOff(void)
```

This routine turns off the LCD controller and the LCD panel.

6.9.7. LCDOn

```
void LCDOn(void)
```

This routine turns on the LCD controller and the LCD panel.

6.9.8. LCDSetPixel

```
void LCDSetPixel(long lX, long lY, char cColor)
```

This routine fills the specified pixel with the given color.

6.10. uart.c

This file contains routines for using the UART interfaces of the CL-PS7111. Each UART is a 16C550-like device capable of bi-directional asynchronous communication at up to 115200 baud.

Contained in this file are two global variables, lPort1Enabled and lPort2Enabled. These are used to determine which ports are currently in use. Initially, it is assumed that port 2 is in use (since that is the port being used by Angel for communication with the host debugger and reusing that port would interfere with Angel). If the hardware configuration varies, the initial values of these two variables can be changed to accurately reflect the hardware.

6.10.1. UARTCharReady

```
long UARTCharReady(long lPort)
```

This routine determines if there is a character ready to be read from the specified UART port. The return value will be non-zero if there is a character waiting to be read.

6.10.2. UARTDisable

```
void UARTDisable(long lPort)
```

This routine will turn off the power to the internal UART.

6.10.3. UARTEnable

```
long UARTEnable(long lPort, long lDataRate, long lDataBits,
                long lStopBits, long lParity, long lEvenParity)
```

This routine will configure the UART port to the specified data rate and format and power on the internal UART port. The supported data rates are 115200, 76800, 57600, 38400, 28800, 19200, 14400, 9600, 4800, 2400, 1200, and 110 baud. The supported data bits are 5, 6, 7, and 8. The supported stop bits are 1 and 2. If lParity is zero, there is no parity bit. If it is non-zero and lEvenParity is zero, then there is an odd parity bit, otherwise there is an even parity bit. The return value will be zero if the UART has already been configured (or the IrDA port if UART1 is requested) or if the data rate and/or format in invalid, and one otherwise.

6.10.4. UARTReceiveChar

```
char UARTReceiveChar(long lPort)
```

This routine will read a character from the specified UART and return it to the caller.

6.10.5. UARTSendChar

```
void UARTSendChar(long lPort, char cChar)
```

This routine will send a character to the specified UART.

7. Samples

There are several sample programs that use the routines in lib7111 to perform some simple demonstrations of the peripherals on the CL-PS7111 evaluation board. The source code for these routines is contained in the *ps7111\samples* directory of the CD-ROM.

7.1. adcsamp

adcsamp is a program that samples the value of potentiometer R47 with the SSI ADC and displays its value on the LCD panel. When run, it will take 16384 samples of the potentiometer, displaying each sample on the LCD panel as it is taken. After all the samples have been taken, the program will wait until a button is pressed on the evaluation board's keyboard, at which point the program will exit. The adcsamp.apj project file will build this program.

7.2. audio

audio is a program that records 65536 samples (~8.2 seconds) of PCM data from the microphone with the codec interface and then plays the data back on the speaker. When run, it will wait until a button on the evaluation board's keyboard is pressed. It will then record data from the microphone. It will then wait until a button on the keyboard is pressed again. It will then play back the recorded data. After a final keyboard button is pressed, it will exit. Messages on the LCD panel indicate what is currently happening. The audio.apj project file will build this program.

7.3. flashtst

flashtst is a program that tests the ability to program data into the FLASH memory. When run, it will go sector by sector through FLASH memory bank 1, erasing the sector, programming a bit pattern into the first word of the sector, reading back the bit pattern to verify that it was programmed correctly, repeating with a different bit pattern, and finally erasing the sector. Progress will be displayed on the LCD panel, along with status of the data verify on each sector. When all sectors have been programmed, it will wait for a keyboard button to be pressed, after which it will exit. The flashtst.apj project file will build this program.

7.4. irrecv

irrecv is a program that receives data from the IrDA port and displays it on the LCD panel. When run, the IrDA port is configured for 9600 baud, 8-N-1 data format. It will then print all characters received from the IrDA port onto the LCD panel. When the "-" character is received, the program will exit. The irrecv.apj project file will build this program.

7.5. irxmit

irxmit is a program that sends data out to the IrDA port. When run, the IrDA port is configured for 9600 baud, 8-N-1 data format. Data is then read from the host debugger using gets() (which uses the Angel multi-hosting library). The data is then sent out to the IrDA port. This will continue until a "-" character is sent, which will cause the program to exit. The irxmit.apj project file will build this program.

7.6. keyboard

keyboard is a program that displays the state of the keyboard matrix on the LCD panel. When run, a grid is drawn on the LCD panel that represents the keyboard array on the daughter card of the CL-PS7111 evaluation card. When a button on the keyboard is pressed, the corresponding square of the grid is filled with a solid circle. When the keyboard button is released, the corresponding square is cleared. The program exits when two keyboard buttons are pressed simultaneously. The keyboard.apj project file will build this program.

7.7. screen

screen is a program that demonstrates the drawing capabilities of the LCD panel. When run, it will draw a series of images on the LCD panel. Once each image is drawn, it will wait until a keyboard button is pressed, causing it to proceed to the next image. After the last image is drawn and a keyboard button pressed, the program will exit. The screen.apj project file will build this program.

7.8. touchscr

touchscr is a program that demonstrates the touch screen interface on the LCD panel. When run, it will go in sequence around the four corners of the LCD panel and ask that the screen be touched in the box drawn in that corner. It will start with the upper left corner and proceed clockwise. This is used to calibrate the touch screen interface. Once the four corners have been touched, drawing on the screen will leave a trail of "ink" on the LCD panel underneath the position drawn upon. Pressing a keyboard button will exit the program. The touchscr.apj project file will build this program.

7.9. uartecho

uartecho is a program that retransmits all data received on a UART port (i.e. echoes the data back to the sender). When run, it will configure the UART (which is configurable by the PORT define in the source code) for 9600 baud, 8-N-1 data format. It will then read characters from the UART, sending each character read back to the same UART. This will continue until a "-" character is received, which will cause the program to exit. The uartecho.apj project file will build this program.