RX600 Series

Direct Drive LCD Demonstration

R01AN0331EU0103
Rev.1.03
Sep 24, 2012

Introduction

The purpose of the LCD Direct Drive demonstration is to show how to create an interactive TFT-LCD panel application using the Renesas DirectLCD API (application program interface) in a real-time environment. The accompanying workspace and software are provided as sample code that will execute on a Renesas LCD Direct Drive demonstration platform. This document will explain in detail the program structure of this sample code.

The user of this document should also refer to the “Direct Drive LCD Design Guide.pdf” and “GAPI User Manual.pdf” (contained within the sample code workspace) for more details on the operation of the API’s used in the sample code.

This sample code utilizes the FreeRTOS as the real-time operating system. The technical documents of FreeRTOS can be accessed at www.freertos.org.

Target Device

RX62N, RX63N

Target LCD Panels

LCD panels with a standard TTL RGB, HSynch, VSynch, PixClk, and Data Enable interface.

The Direct Drive LCD solution is highly configurable, and capable of producing many different timing configurations which drive the input signals of TFT-LCD panels from various panel manufacturers. The signal timing generated from the Direct Drive LCD solution depends on your choice panel resolution, frame buffer memory, and desired panel refresh and animation rates.

Although Renesas provides guidelines and examples for configuring the signal timing, Renesas is not responsible for meeting the AC timing specifications of your specific choice of TFT-LCD panel. Please contact your TFT-LCD panel manufacturer to ensure the Direct Drive LCD solution complies with the panel timing limitations.

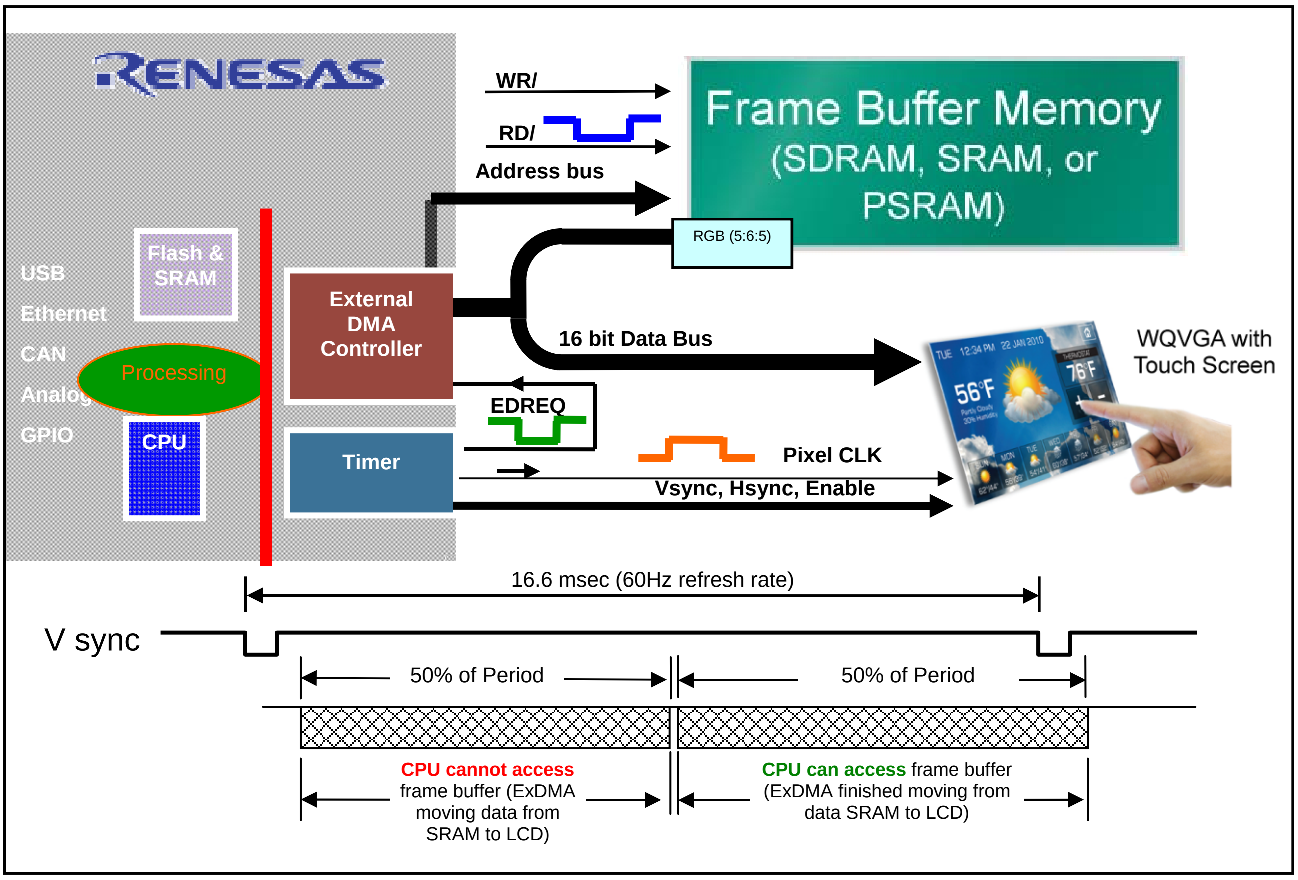
Contents

1. Background

1.1 What is LCD Direct Drive?

LCD Direct Drive is the ability to control a TFT LCD panel via the on chip peripherals of Renesas microcontrollers. These peripherals include the external bus controller (BSC), External DMA controller (ExDMAC) and timer units (TPU or MTU). The LCD Direct Drive API will transparently transfer the contents of a RAM frame buffer to an LCD panel using less than 2% of the available processor bandwidth (WQVGA panel at 60Hz on RX62N running at 100Mhz).

At any time, the MCU can be executing code and accessing data from internal flash and RAM. The contents of the external RAM frame buffer can be updated by the MCU during the vertical blanking portion of the LCD update cycle. The coordination of external bus access by the MCU is handled by the API to ensure there is no contention.



2. Demonstration Application

The following instructions assume a reasonable familiarity with the Renesas development tools.

2.1 Installation

To install the demonstration application, run the installation program that accompanies this application note appropriate for your target platform (DirectLCD_RX62N_RSK.exe for example). Ensure that the required HEW, compiler and debugger components are previously installed (the DirectLCD installer will note which versions are expected). The install will ask for a location to unpack the workspace. Use the default location if acceptable.

2.2 Compile

Open the demonstration workspace by double clicking on “DirectLCD.hws”.

2.2.1 r_Packages.lib

To facilitate simple reuse of the provided components in customer projects, the LCD direct drive API, Graphics API and RTOS are separated into the library project “r_Packages.lib”. This library is then easily included in the user application project (and the DirectLCD demo project). All source code is provided for this library, and can be debugged as part of the customer project.

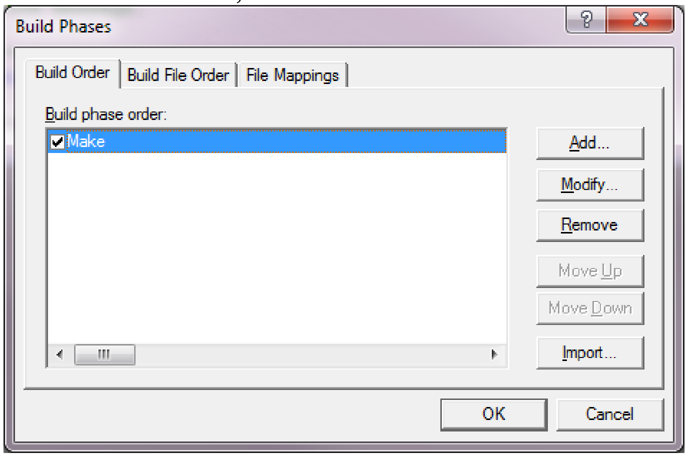
To build the “r_Packages.lib”, set the current project to “r_Packages” by right clicking on project and selecting “Set as Current Project”.

Special one-time procedure for 64-bit windows. To address issues associated with paths in 64-bit Windows OS, we must provide an alternate path to the HEW program directory.

From the command shell, create an alternate path to the HEW program directory. Bold text is required command entries.

```
C:\Users\jbrabend01>cd \
C:\>md Renesas  (This directory may already exist on your computer so ignore any “already exists” message.)
C:\>cd Renesas
C:\Renesas>mklink /D Hew "c:\Program Files (x86)\Renesas\Hew"
symbolic link created for Hew <<===>> c:\Program Files (x86)\Renesas\Hew
C:\>exit
```

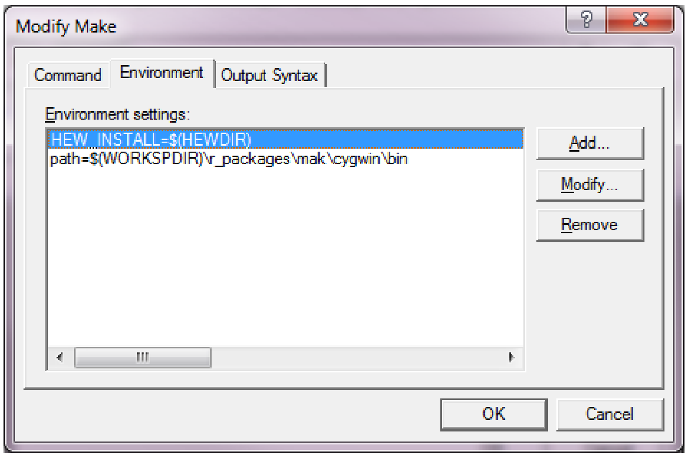
Within the HEW IDE, select: Build->Build Phases



Build phase order:

☑ Make

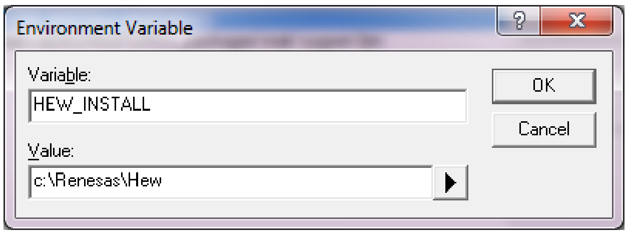
Click on “make” and select “Modify”, select “Environment” tab.



Environment settings:

HEW_INSTALL=$(HEWDIR)
path=$(WORKSPDIR)\r_packages\mak\cygwin\bin

Click on “HEW_INSTALL” and select “Modify”

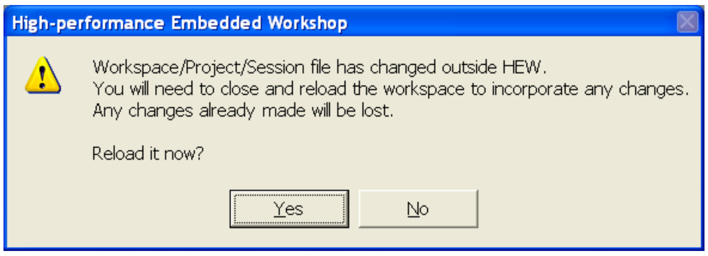


Variable: HEW_INSTALL

Value: c:\Renesas\Hew

Change the “Value” to the newly created alternate path to the HEW program directory.
Select “OK” to exit all dialogs.

Build the "HWP" configuration of r_Packages.lib by pressing <F7>. This will build a browsable tree under the HEW project window in addition to the library. It is normal to receive the following message when building the "HWP" configuration (as the HEW project file is built). Simply select "Yes" to continue.


High-performance Embedded Workshop

Workspace/Project/Session file has changed outside HEW.
You will need to close and reload the workspace to incorporate any changes.
Any changes already made will be lost.

Reload it now?

After the library has been built once, future builds can be run using the "New" configuration (HWP configuration is only necessary to build the browsable project the first time). The "r_Packages.lib" library only needs to be re-built when its configuration files change.

2.2.2 DirectLCD.abs

Select the demonstration application project by right clicking on the "DirectLCD" project and selecting "Set as Current Project". Build the project by pressing <F7>.

2.3 Running the application

Connect your debugger (E1) to your target hardware platform (RX62N RSK LCD Direct Drive) and establish a debugging session.

Download the previously built "DirectLCD.abs" to the target.

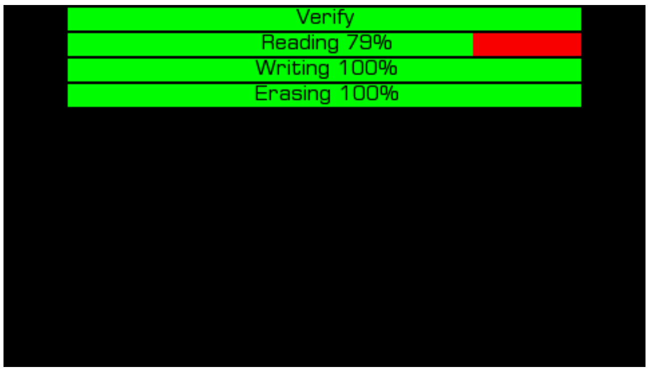
In addition to building the "DirectLCD.abs" executable, the build process also creates a "Resources.bin" file containing all of the graphical resources contained within the workspace's "Resources" directory. This file is stored in external serial flash in the demonstration application. Whenever the contents of this file change, it must be saved to the targets PCB's serial flash. This is accomplished by executing a HEW script file "ResourceLoad.hdc" located in the DirectLCD project directory. In this demonstration workspace, this script file is preloaded in the "Command line" "Console", and it can be executed by simply pressing the "play" button on the console. Ensure a debugger connection has been established and the DirectLCD.abs file has been downloaded prior to running this script.

This script only needs to run when the contents of the "Resources.bin" have changed.

Once the "DirectLCD.abs" and "Resources.bin" files have downloaded to the target, running the project will display the demonstration screens.

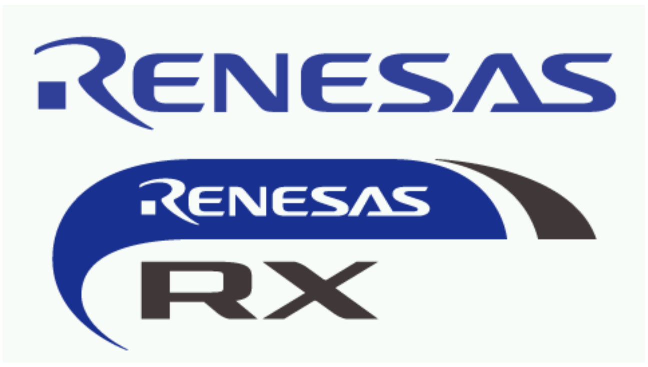
2.3.1 Resource Loading Screen

On power on, the resource contents of the serial flash are transferred to external RAM for runtime access. This screen indicates the status of resource reading and writing (when programming script is executed). The "Resources.bin" file has embedded CRC values to allow verification of the file transfer. This screen is generated prior to starting normal screen handling in "eventmgr.c". The font used for this screen is stored internally to the MCU and loaded with the application. All other screen resources are accessed from the loaded "Resources.bin".



2.3.2 Splash Screen

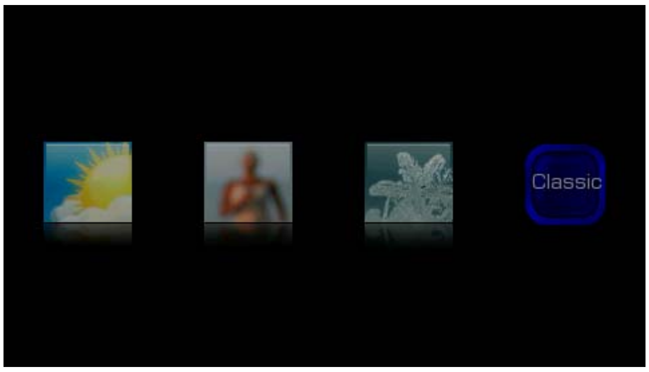
The demonstration loads the file “_SplashScreen.bmp” and displays this for four seconds prior to transitioning to the home screen. This screen can be bypassed by touching the display. This screen is generated prior to starting normal screen handling in “eventmgr.c”.



2.3.3 Home Screen

Once the framework has loaded the resources, it will start processing the screen defined by the “ScreenHomeData” structure. In this demonstration, that structure is defined in “ScreenHome.c”. This structure refers to a list of screen objects that are located in the _SCR_HM memory section. The objects in this memory section are defined in various files and collected by the linker at build time to create the complete object list. This behavior allows for simple inclusion/exclusion of objects by linking/not linking files into the project.

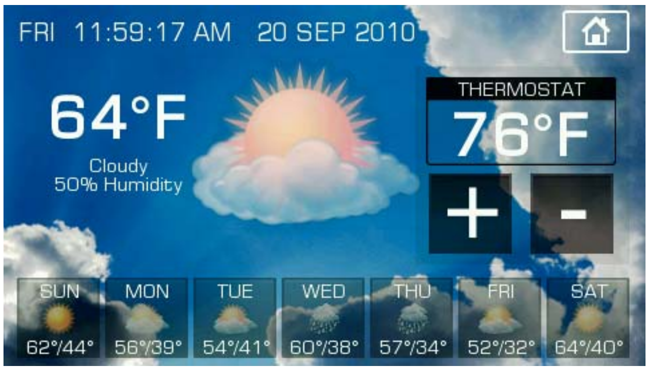
Here you can see objects that are defined in four different screen files. Pressing any of these objects causes their associated callback function to be executed. In the case of the objects on this screen, the callback sends a new screen event that causes the framework to transition to processing the data structure associated with that screen file.



2.3.4 Thermostat Screen

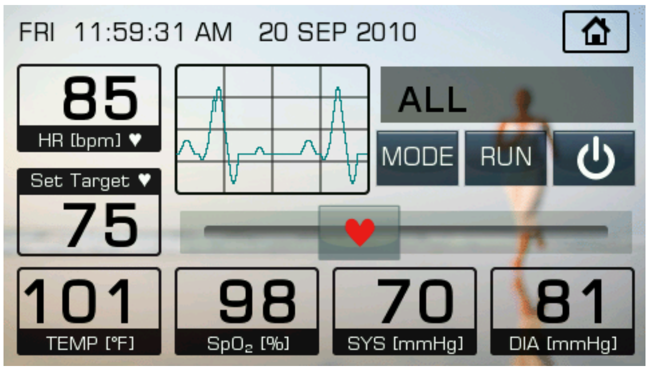
This screen is defined in the file “ScreenThermo.c”. It shows many aspects of the graphics API including anti-aliased fonts, UTF-8 decoding, alpha blending and animation. Notable on this screen is how the buttons on the bottom of screen are run-time composed by combining the screen shape, the forecast icon, the forecast text and the day- of-the-week text (see source code function “DayButtonDraw”). These components can be dynamically selected to create the necessary graphics as the conditions change.

Pressing the “Home” button in the upper right corner of the screen will trigger an event to switch to the previous screen, this is true for all sub-screens.



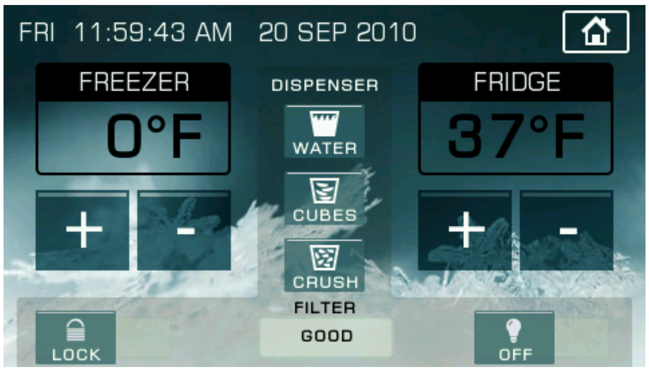
2.3.5 Medical Screen

This screen is defined in the file “ScreenMed.c”. Notable on this screen is the graph being displayed. If the slider is not touched, it displays a repeating waveform. If the “heart slider” is pressed, the graph will reflect the slider value (see source code function “drawGraph”). Additionally, this code demonstrates the usage of the slider. Sliders, buttons and several other behaviors are collected in the “SliderHandler” and “ButtonHandler” functions in the file “ScreenObjs.c”. These functions process event behavior and appropriately draw objects on the screen and maintain state about the object. These functions are not part of the Graphics API, but use the API for their screen drawing needs.



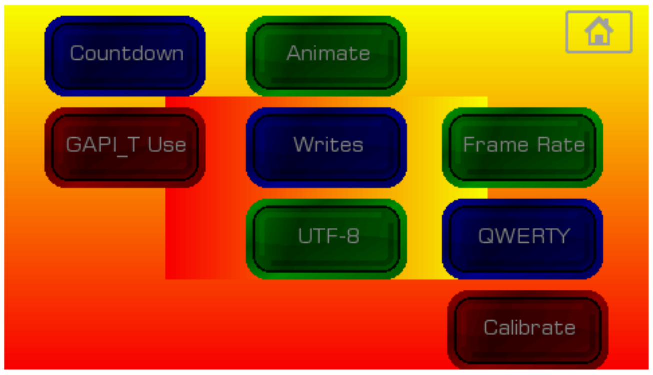
2.3.6 Refrigerator Screen

This screen is defined in the file “ScreenRefrig.c”. On this screen, two data boxes are being maintained with the temperature values via the “DataBoxHandler” and the “lock” and “light” buttons are drawn using the “SliderSwitchHandler”. Both of these handlers are supplied in the “ScreenObjs.c” file. As user objects are created, often they readily lend themselves to creating common handlers that can easily be reused. Examining these supplied handlers provides insight into how you can create your own.



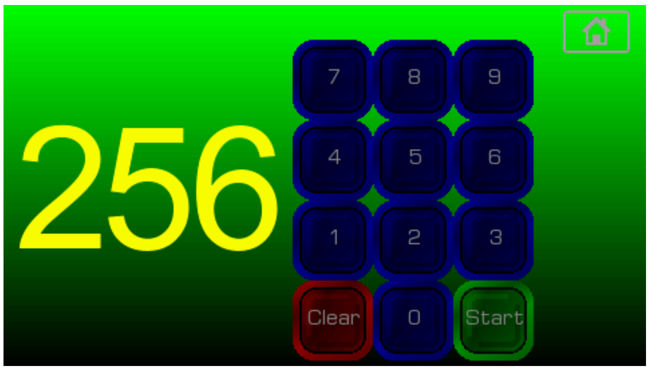
2.3.7 Classic Home Screen

This screen is defined in the file “ScreenHomeClassic.c”. The objects on this screen are collected by the linker from files allocating objects in the _SCR_HM_CLASSIC section. This allows for their simple inclusion/exclusion of objects similar to the home screen. The objects on this screen demonstrate basic behavior of buttons and capabilities as described below.



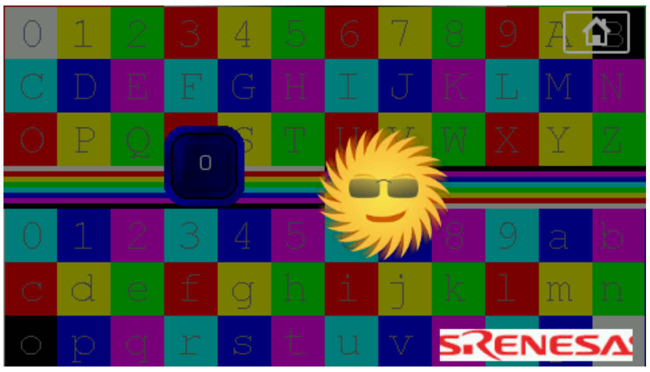
2.3.8 Countdown Screen

This screen is defined in the file “ScreenCountdown.c”. This screen demonstrates how to cleanly display transparent text by compositing on work frame and transferring to the display frame. Also, with the large text the anti-aliasing of the fonts can be clearly seen. Another technique shown is the usage of a single callback to handle a multi-button object (the numeric keypad). This screen also uses a “screen task” thread to update the timer value while running (showing GAPI running under multiple threads).



2.3.9 Animate Screen

This screen is defined in the file “ScreenAnimate.c”. This screen shows how the GAPI can be used to animate an object. Here two screen tasks are used to update the two images (one for the sun and one for the scrolling Renesas banner). Also note that the sun image has a transparency (alpha) channel that is used during rendering to blend the image into the background as it moves.

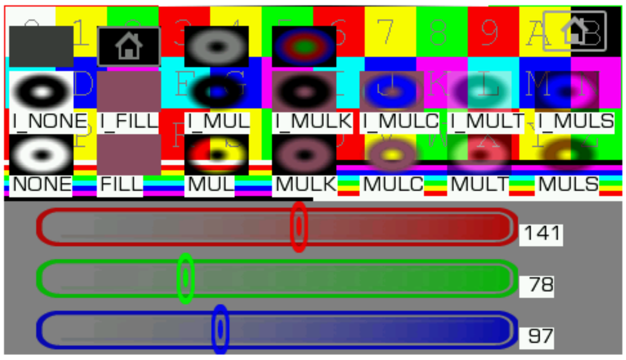


2.3.10 GAPI_T Use Screen

This screen is defined in the file "ScreenSlider.c". This screen shows usage of multiple instances of the slider object., as well as illustrates the behavior of the GAPI copy modes. The slider values control the color of the R_GAPI_CopySub "transparency" attribute.

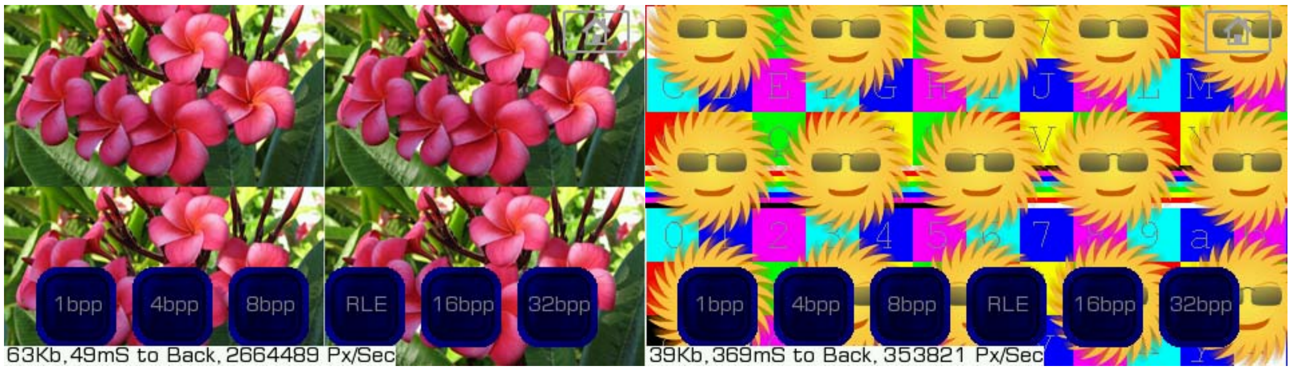
The first icon in the top row, selects the background area under the output area for the R_GAPI_CopySub source to demonstrate dimming/fading/shading.

The remaining top row icons select corresponding source images for the R_GAPI_CopySub source. The rendered output area shows the processing of the various modes as labeled.



2.3.11 Writing Screen

This screen is defined in the file "ScreenWrites.c". This screen measures the amount of time that it takes to display a screen of data based on the source image format. The amount of time required depends on the image size, the image type, and the available vertical blanking time. Here, you can see the difference between writing a 16bpp image that is basically just copying data (49mS for 4 images to fill screen) and a 32bpp image with an alpha channel that requires much more processing (369mS for 15 images to fill screen). In both cases the data was written with a 60Hz refresh and 44% vertical blanking.

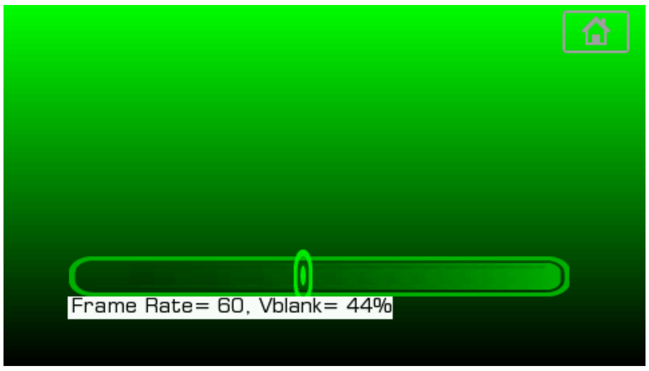


63Kb,49mS to Back,2664489 Px/Sec

39Kb,369mS to Back, 353821 Px/Sec

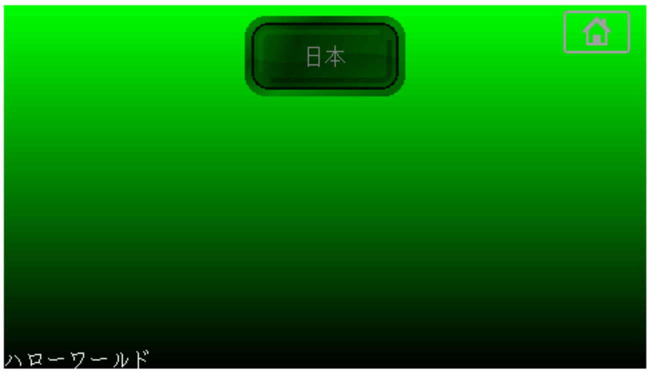
2.3.12 Frame Rate Screen

This screen is defined in the file "ScreenIO.c". This screen allows the screen frame refresh rate and vertical blanking percentage to be adjusted. The Direct Drive API allows this attribute to be dynamically adjusted. Using this screen you can see the affect on the LCD panel characteristics and how the vertical blanking percentage affects performance using the write test screen.



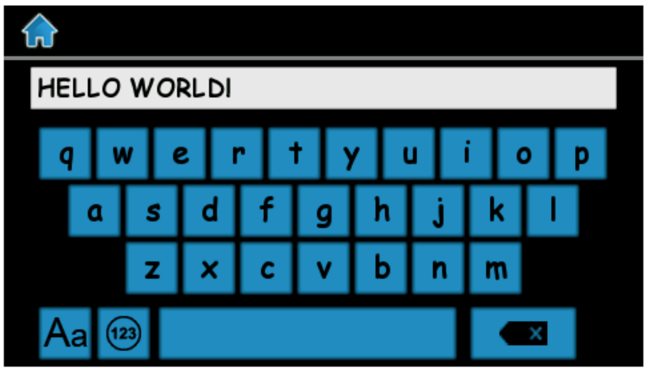
2.3.13 UTF-8 Screen

This screen is defined in the file “ScreenUTF.c”. This screen shows an example of usage of the UTF8 string processing capability of GAPI..



2.3.14 QWERTY Screen

This screen is defined in the file “ScreenQWERTY.c”. This screen shows an example of a virtual keyboard.

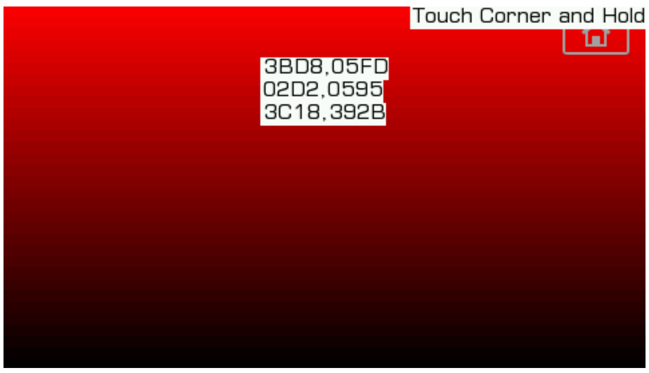


2.3.15 Calibration Screen

This screen is defined in the file “ScreenCalibrate.c”. This screen allows the adjustment of the calibration points used by the touchscreen driver. The values determined in this screen are entered into the “TouchCalibration” data structure; additionally these values can be saved to internal or external flash as part of a calibration procedure. In the demonstration code, predetermined values are loaded into this data structure. In the case of the LCD panel supplied with the kit, the consistency of the supplied touch-screen overlay has not required individual calibration.

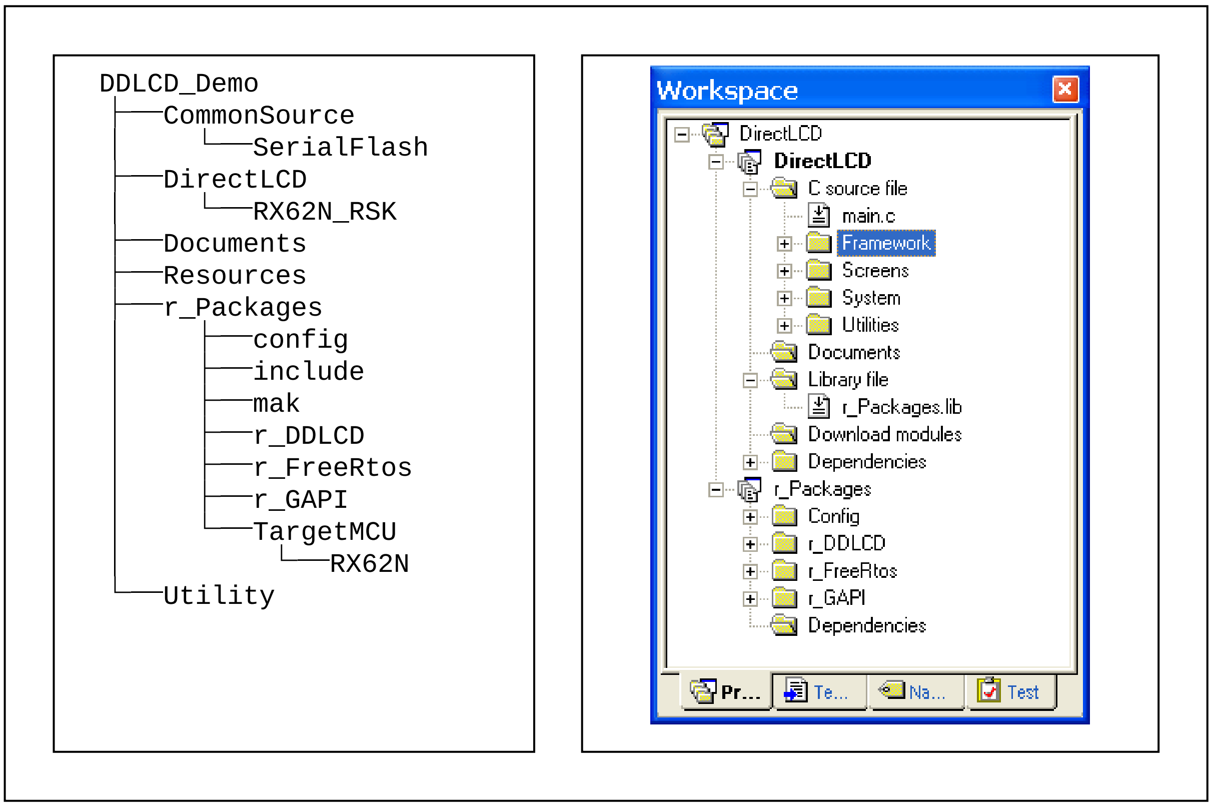
Once this screen has been entered, it cannot be exited without completing the calibration procedure. When the calibration is completed, all touches to the screen are shown with their corresponding x,y coordinates. To evaluate the effectiveness of the procedure, the user can select non-requested touch points and observe the resulting behavior. For example, touching the left side instead of right side during calibration will invert touch locations on this axis. As the initial calibration points are pre-determined, simply resetting the MCU will return to appropriate behavior.

To facilitate use with an unknown panel with incorrect calibration constants, pressing SW1 on the PCB will start the calibration process.



3. Code Structure

The source code directory is shown on the left and the HEW workspace is shown on the right in **Figure 1**. For clarity in reading this application note, it is recommended that the project is opened and referred to.



```
DDLCD_Demo
 ├──CommonSource
 │     └──SerialFlash
 ├──DirectLCD
 │     └──RX62N_RSK
 ├──Documents
 ├──Resources
 ├──r_Packages
 │     ├──config
 │     ├──include
 │     ├──mak
 │     ├──r_DDLCD
 │     ├──r_FreeRtos
 │     ├──r_GAPI
 │     └──TargetMCU
 │           └──RX62N
 └──Utility
```

3.1 File Descriptions

This subdirectory contains all the common source files shared by the Direct Drive platforms. The descriptions of those files are listed in following tables.

3.1.1 *Initialization Code, project, object and other files*

File Name	Category	File Description	Location
resetprg.c	Standard C	Reset vector initialization	/DirectLCD
hwsetup_platform.c	Standard C	Initial MCU hardware configuration	/DirectLCD
*.hwp, *.hsf, *.h	HEW	HEW project files	/DirectLCD
.	HEW	Generated object files and libraries	/DirectLCD/RX62N_RSK

.	Documents	PDF files with project documents, APIs and schematics.	/Documents
.	Resources	Resource files (graphic, font, audio, etc), that are used within the demonstration project. The contents of this directory are packaged into a single “resources.bin” file by the project build process. This file can then be written to the serial flash for runtime usage.	/Resources
.	Utility	Windows applications that are used to create files used by the demonstration.	/Utility

3.1.2 *Application Demo Code*

File Name	**Category**	**File Description**	**Location**
main.c	Standard C	Main program	/CommonSource
Global.h	Demo Header	Global definitions and macros	/CommonSource
LCD_Demo.h	Demo Header	Graphics application definitions and macros for LCD Direct Drive demo	/CommonSource
EventMgr.c	Framework	Framework tasks for controlling the touch screen and screen manager functions.	/CommonSource
Frames.c/ Frames.h	Framework	Routines and API to manage the memory rasters allocated by the demo.	/CommonSource
ScreenMgr.c /ScreenMgr.h	Framework	Routines and API to control the running and switching of screens.	/CommonSource
ScreenObjs.c/ ScreenObjs.h	Framework	Routines and API to provide common screen object behavior such as buttons and sliders.	/CommonSource
Screen*.c	Screens	Files that contain graphics demo screen code. Each screen is contained in a corresponding file	/CommonSource
crc.c	Utility	Routines to generate CRC values.	/CommonSource
FindFile.c/ FindFile.h	Utility	Routines and API to access the contents of the resource files	/CommonSource
FlashSerial.h	Utility	API for the serial flash access routines	/CommonSource
FlashSerial.c	Utility	Serial flash memory access routines.	/CommonSource/SerialFlash
iResources.h	Utility	Resource image file that is always located in internal flash memory, thus allowing use during boot process.	/CommonSource
Resources.c	Utility	Routines to load, store and verify resource files in serial flash memory.	/CommonSource
Simple_printf.c	Utility	Simplified (small memory, reentrant) Printf routines	/CommonSource
SPI_via<*port*>.c	Utility	Serial flash memory access routines specific to a serial peripheral.	/CommonSource/SerialFlash
Timer_RTC.c/	Utility	Routines and API to provide common screen	/CommonSource

Timer_RTC.h		object behavior such as buttons and sliders.	
TouchScreen.c/ TouchScreen.h	Utility	Routines and API for touch screen driver	/CommonSource

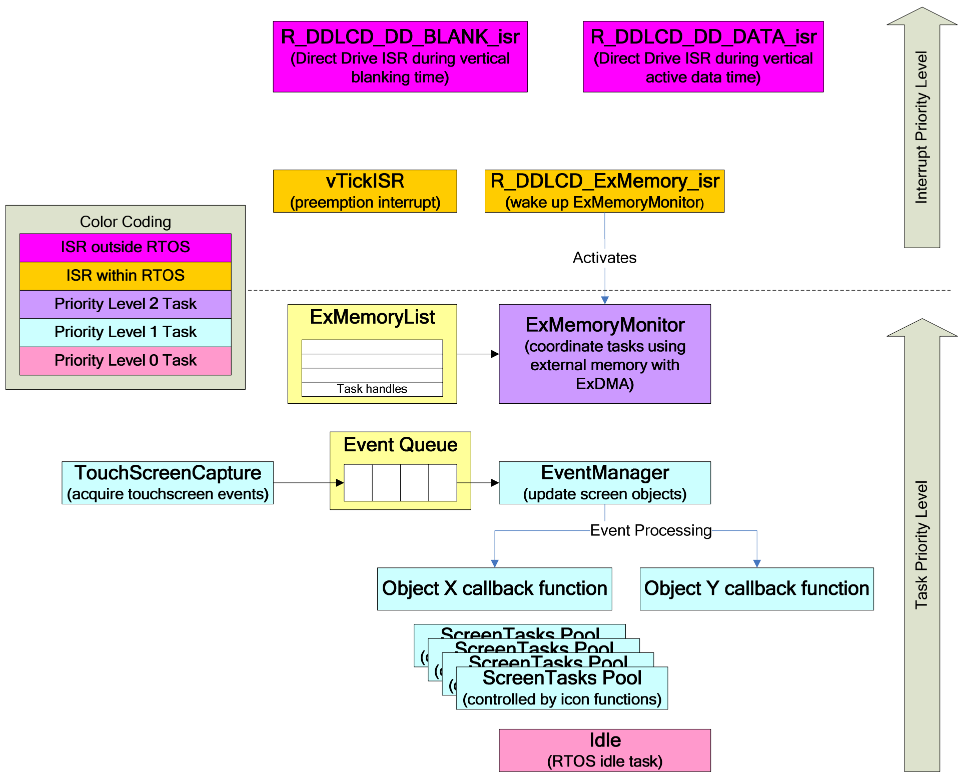
3.1.3 *r_Packages*

The "r_Packages.lib" file is included in the DirectLCD project. The contents of this library are described below.

File Name	**Category**	**File Description**	**Location**
*.h	Configuration	The files in this directory are used to configure the packages during the build process. Please refer to the relevant package document for details. The r_Pacakges.lib needs to be rebuilt after any changes to these files. **The files in this directory are the only r_Packages files to be modified by the user.**	/r_Packages/config
*.h	API	The files in this directory provide access to the package's APIs. Note that the files in this directory are created during the r_Pacakges.lib build process. **This is the only r_Packages include path the user needs to include in their project.** **All supported API calls are accessed through the include files in this directory. Access to the library is only supported through these public APIs.**	/r_Packages/include
.	Build	The r_Packages library build environment. This "make" based build environment is invoked by "building" the r_Packages project.	/r_Packages/mak
.	DDLCD	The Renesas Direct Drive LCD library package Please refer to the provided Direct Drive user manual for details and API.	/r_Packages/r_DDLCD
.	GAPI	The Renesas GAPI graphics library package Please refer to the provided GAPI user manual for details and API.	/r_Packages/r_GAPI
.	RTOS	The FreeRTOS library package Please refer to the FreeRTOS website for details and API.	/r_Packages/r_FreeRTOS

4. Program Operation

The following figure illustrates the interrupt service routines (ISRs) and tasks that are running in the LCD Direct Drive demonstration project.



4.1 RTOS

The LCD Direct Drive demonstration requires an RTOS to manage the tasks and access to the external bus. FreeRTOS has been chosen for this purpose. However, most RTOS's are equally suited to the needs of the system. To aid in porting to another RTOS, all usages of the RTOS are documented in the code with the comment “RTOS_USAGE”.

4.2 Interrupts

The LCD Direct Drive API executes two “R_DDLCD_DD isrs” once the driver is initialized. These ISRs control the ExDMA and timer channels that transfer data from the external frame RAM to the LCD panel. For optimized performance, one ISR is active during the vertical blanking time, and the other ISR is active during the data transfer portion of the refresh cycle. These ISRs are triggered once per horizontal period (line) of the refresh cycle.

Twice per vertical period (frame) (once before the data transfer starts and once after it ends), the R_DDLCD_ExMemory_isr is triggered to activate the “ExMemoryMonitor” task. This task is responsible for coordinating software's access to the external bus. Tasks that use the external bus (typically to update the frame RAM) are required to call the “R_DDLCD_ExMemoryAcquire” function to notify the framework. Multiple tasks can be simultaneously registered to access the external bus (up to the configurable DD_EXMEMORY_MANAGER_MAX_TASKS limit). The “ExMemoryMonitor” task suspends all registered tasks at the beginning of the vertical data transfer and resumes these tasks at the beginning of the vertical blanking period. Tasks no longer using external RAM for a period of time can unregister itself by calling “R_DDLCD_ExMemoryRelease”.

The consequence of accessing the external bus without registering is contention for the external bus. If this occurs, the MCU core will be held in a wait state until the ExDMA peripheral has completed its current block transfer; also visible screen artifacts may be seen if the MCU core access delays the start of the ExDMA transfer.

The last ISR shown in the diagram is the “vTickISR” which causes the FreeRTOS scheduler to pre-empt the currently running task and evaluate which task should run next. If multiple tasks at the same task priority level are ready to run, they will be run in turn after each “vTickISR” in a “round robin” fashion.

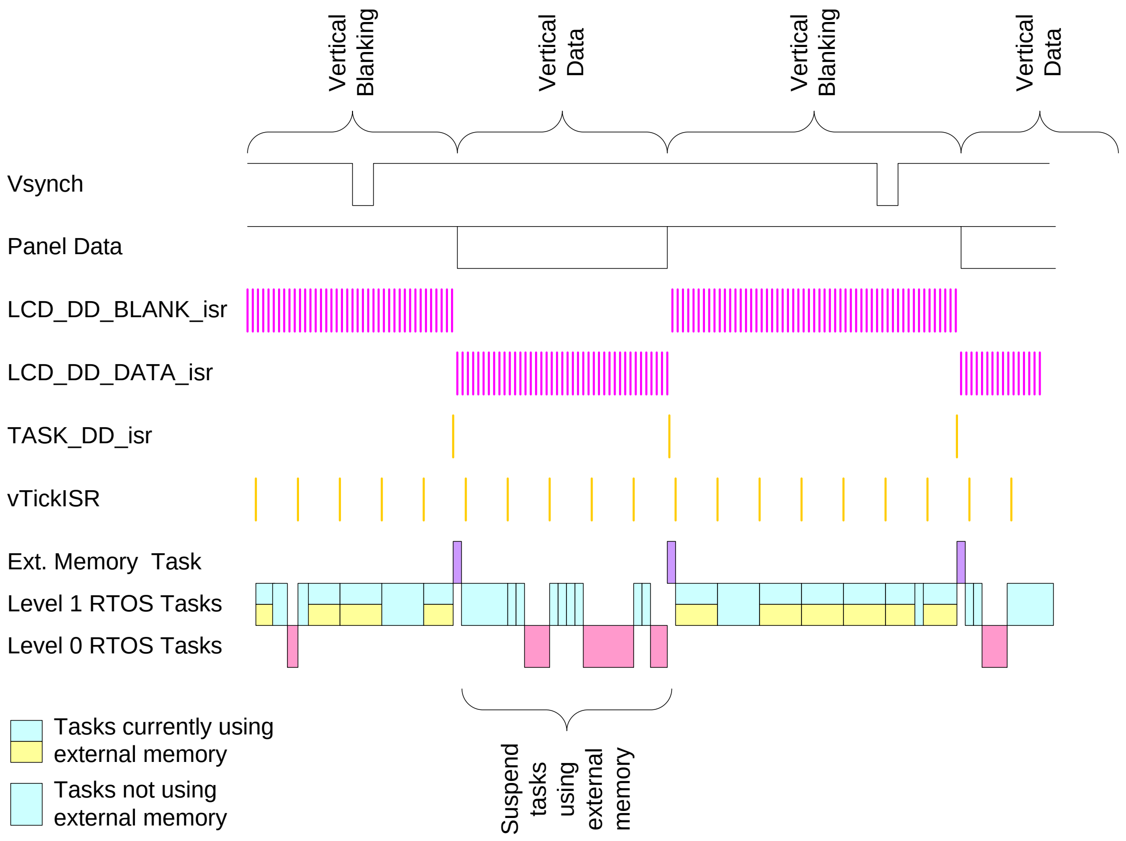
4.3 Tasks

The “TouchScreenCapture” task is responsible for acquiring events from the touchscreen overlay. When a touch event is judged to have occurred by this task, the information is added to the EventQueue.

The EventQueue is also used to accept events from other sources such as push buttons, timers or application source code.

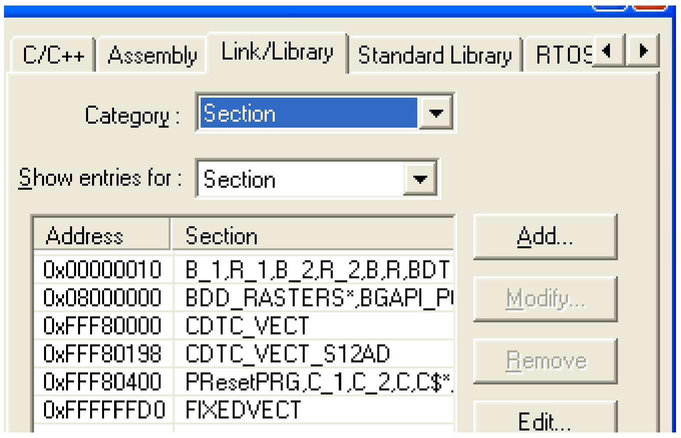
The "EventManager" task determines how events on the current screen should be processed. Every event received is passed to the callback function associated with each object on the current screen. It is the responsibility of the object callback functions to determine if and how to process the event. The supplied demonstration code shows numerous examples of event processing.

The next figure shows the interaction of the ISRs in relation to the LCD panel Vsynch signal. It also shows how tasks that have been registered as currently accessing the external bus are suspended during the "vertical data" portion of the period.



4.4 Memory Usage

The LCD Direct Drive demonstration code uses both external RAM memory (for frame buffers, resource storage and pools) and internal RAM and Flash memory (for program storage, stack, variables and constants). The location of these resources in memory is determined by the toolchain linker options.



Category : Section

Show entries for : Section

Address	Section
0x00000010	B_1,R_1,B_2,R_2,B,R,BDT
0x08000000	BDD_RASTERS*,BGAPI_PI
0xFFF80000	CDTC_VECT
0xFFF80198	CDTC_VECT_S12AD
0xFFF80400	PResetPRG,C_1,C_2,C,C$*,
0xFFFFFFD0	FIXEDVECT

In the RX600 family, internal RAM is mapped to the bottom of the memory space, internal flash in mapped to the top of the memory space, and external SDRAM is located at 0x0800 0000.

4.4.1 Sections

Section Name	Type	Description
B_*	RAM	BSS memory that is zero filled by startup code. The heap is allocated within this section, and because the RTOS makes extensive use of the heap, significant memory is allocated by the HEAPSIZE macro in “sbrk.h”.
R_*	RAM	Initialized RAM memory. This area is initialized by startup code with data contained in the D_* sections.
BDTC_TABLE	RAM	Data storage structures used by the DTC unit.
SU	RAM	User stack space. As the RTOS allocates individual stacks from the heap for each task, minimal space is required in this section.
SI	RAM	Interrupt stack space. This memory is used during the processing of interrupts for stack usage. As interrupts can be nested, care must be taken to ensure adequate space is allocated.
BDD_RASTERS	External RAM	Space allocated by the Direct Drive API for the display raster. The demonstration uses 3 frames, background, work and display.
BGAPI_POOL*	External RAM	Space allocated by the GAPI for dynamic memory pools.
BResources	External RAM	Memory used for runtime access of the Resources.bin file. Note that because this file is of variable size, the demonstration code is allocating 2Mb for this use in “ScreenMgr.c” (this is the size of the serial flash on the demonstration PCB).
CDTC_VECT*	Flash	These sections allocate the vector table used by the DTC peripheral. Because this table must be aligned to a 4Kbyte boundary, it is located in the start of flash.
PResetPrg	Flash	Program section for reset code.
C*	Flash	Sections allocated for constant variables in the demonstration code.
C$VECT	Flash	The MCU’s relocatable vector table.
D*	Flash	Initialization data for the R* sections.
P	Flash	Program memory associated with the application. Noteworthy is that the entire demonstration application only requires ~40Kb of code space.
W	Flash	Constant memory allocated for use in switch tables.
C_SCR_HM*	Flash	Constant structures that define the contents of the home screen.
C_SCR_HM_CLASSIC*	Flash	Constant structures that define the contents of the classic home screen.
FIXEDVECT	Flash	The MCUs fixed vector table including the reset vector.

5. Resource Storage Access

Resources (BMP’s, Fonts, Audio files, etc) are accessed from a resource image file located in the linear memory of the MCU. These resource image files can be located in internal flash or external RAM. When external RAM is used, the resource file is transferred at power on from non-linear memory (such as serial flash or SD card) by the application code. Prior to the resource image being written from external RAM to serial flash, and also after it is read from serial flash to external RAM, the CRC value in the resource image is verified.

The resource image file is created by a custom utility “ResourceGen.exe”. This command line application takes option arguments as follows:

Option	Description
-D <subdirectory>	Process all files in <subdirectory>.
-e	Include file extension in record name.
-q	Run in quiet mode (minimal output).

-b	Generate binary data file.
-c	Generate 'c' source array (comma delimited by array).
-f	Suppress records containing only 0xFF for -i/-m options).
-i[aaaaaaaa]	Generate Intel hex record format. [aaaaaaaa] optional hex address relocation
-m[aaaaaaaa]	Generate Motorola hex record format. [aaaaaaaa] optional hex address relocation

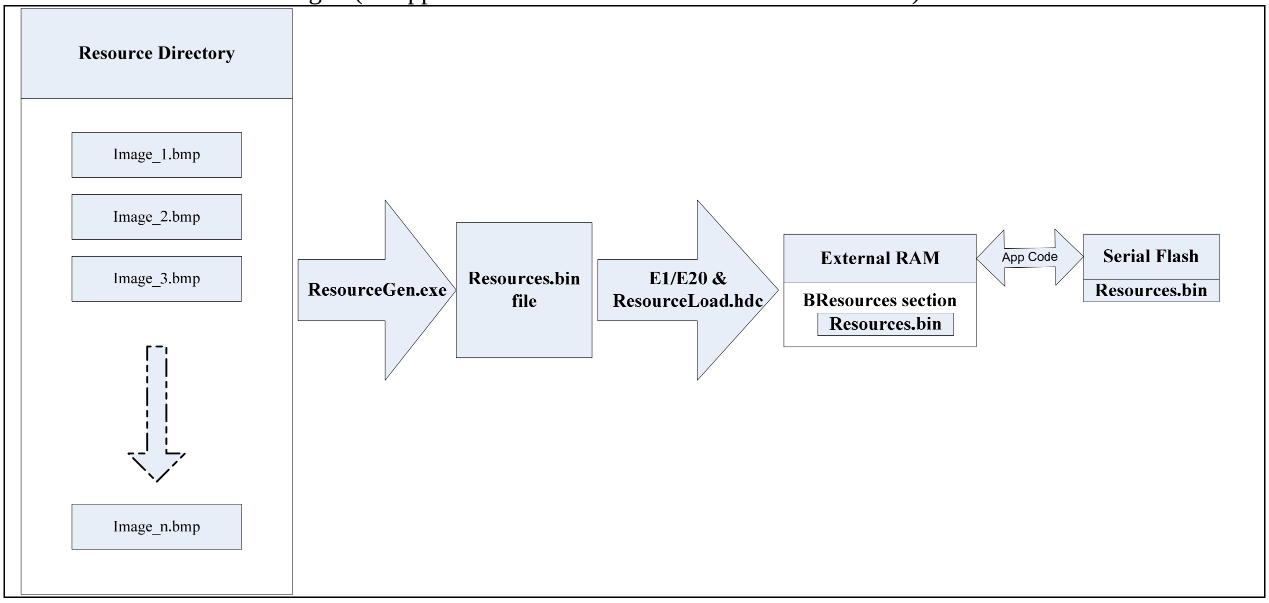
The demonstration application code runs this utility during every build via the HEW custom build phase capability. The options in this build phase are “-b -q -D Resources Resources.bin”; this will take the contents of the Resources directory and include them in the Resources.bin file.

The following table illustrates the format of the file header resource image file. Each entry is 32 bytes long. The first record provides CRC and size information for the resource file. Unless the –e option is used, the record name does not include the extension (File_1.bmp would become “File_1”)

	Record Name (max 24 characters)	**Location (4 Bytes)**	**Size (4 Bytes)**
0x000000	“BFS_Header”	CRC	Resource File Size
0x000020	“File_1”	File_1 offset location	File_1 size
0x000040	“File_2”	File_2 offset location	File_2 size
0x000060	…		
0x0xxxx0	0xFFFFFFFFFFFF (termination record)	0xFFFFFFFF	0xFFFFFFFF

The application code accesses information in an image file by resource name using the “FileFind()” function.

The Direct Drive application demonstration workspace provides HEW build phases (ResourceBuild), utility (Bin_to_mot.exe) and scripts (ResourceLoad.hdc) that manage the creation and loading of this resource file (Resources.bin).
This custom build phase is executed whenever the project is built. All contents of the “Resources” directory are included in the Resources.bin file.
If the resource file is located in internal flash, typically you would include it into the project by generating a comma delimited array using the “-c” option. Refer to the “iResources.h” file in the demonstration code. The resource file than become part of your project download as a constant array.
If the resource file is located in serial flash, it can be loaded manually by running the “ResourceLoad.hdc” script for the platform from the “Command Line” window of HEW. The programming of the serial flash only needs to occur when the resources have been changed (no application code is contained in the resource file).



Website and Support

Renesas Electronics Website

http://www.renesas.com/

Inquiries

http://www.renesas.com/inquiry

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Oct.8.10	—	First edition issued
1.01	Nov.18.10	—	Clarifications and typographic corrections
1.02	Jul.18.12	8,9,10	Update screen descriptions
1.03	Sep.24.12	4	Add 64-bit Windows procedure

General Precautions in the Handling of MPU/MCU Products

The following usage notes are applicable to all MPU/MCU products from Renesas. For detailed usage notes on the products covered by this document, refer to the relevant sections of the document as well as any technical updates that have been issued for the products.

1. Handling of Unused Pins

   Handle unused pins in accord with the directions given under Handling of Unused Pins in the manual.

   — The input pins of CMOS products are generally in the high-impedance state. In operation with an unused pin in the open-circuit state, extra electromagnetic noise is induced in the vicinity of LSI, an associated shoot-through current flows internally, and malfunctions occur due to the false recognition of the pin state as an input signal become possible. Unused pins should be handled as described under Handling of Unused Pins in the manual.

2. Processing at Power-on

   The state of the product is undefined at the moment when power is supplied.

   — The states of internal circuits in the LSI are indeterminate and the states of register settings and pins are undefined at the moment when power is supplied.
   In a finished product where the reset signal is applied to the external reset pin, the states of pins are not guaranteed from the moment when power is supplied until the reset process is completed.
   In a similar way, the states of pins in a product that is reset by an on-chip power-on reset function are not guaranteed from the moment when power is supplied until the power reaches the level at which resetting has been specified.

3. Prohibition of Access to Reserved Addresses

   Access to reserved addresses is prohibited.

   — The reserved addresses are provided for the possible future expansion of functions. Do not access these addresses; the correct operation of LSI is not guaranteed if they are accessed.

4. Clock Signals

   After applying a reset, only release the reset line after the operating clock signal has become stable. When switching the clock signal during program execution, wait until the target clock signal has stabilized.

   — When the clock signal is generated with an external resonator (or from an external oscillator) during a reset, ensure that the reset line is only released after full stabilization of the clock signal. Moreover, when switching to a clock signal produced with an external resonator (or by an external oscillator) while program execution is in progress, wait until the target clock signal is stable.

5. Differences between Products

   Before changing from one product to another, i.e. to a product with a different part number, confirm that the change will not lead to problems.

   — The characteristics of an MPU or MCU in the same group but having a different part number may differ in terms of the internal memory capacity, layout pattern, and other factors, which can affect the ranges of electrical characteristics, such as characteristic values, operating margins, immunity to noise, and amount of radiated noise. When changing to a product with a different part number, implement a system-evaluation test for the given product.

SALES OFFICES Renesas Electronics Corporation http://www.renesas.com

Refer to "http://www.renesas.com/" for the latest and detailed information.

Renesas Electronics America Inc.
2880 Scott Boulevard Santa Clara, CA 95050-2554, U.S.A.
Tel: +1-408-588-6000, Fax: +1-408-588-6130

Renesas Electronics Canada Limited
1101 Nicholson Road, Newmarket, Ontario L3Y 9C3, Canada
Tel: +1-905-898-5441, Fax: +1-905-898-3220

Renesas Electronics Europe Limited
Dukes Meadow, Millboard Road, Bourne End, Buckinghamshire, SL8 5FH, U.K
Tel: +44-1628-651-700, Fax: +44-1628-651-804

Renesas Electronics Europe GmbH
Arcadiastrasse 10, 40472 Düsseldorf, Germany
Tel: +49-211-65030, Fax: +49-211-6503-1327

Renesas Electronics (China) Co., Ltd.
7th Floor, Quantum Plaza, No.27 ZhiChunLu Haidian District, Beijing 100083, P.R.China
Tel: +86-10-8235-1155, Fax: +86-10-8235-7679

Renesas Electronics (Shanghai) Co., Ltd.
Unit 204, 205, AZIA Center, No.1233 Lujiazui Ring Rd., Pudong District, Shanghai 200120, China
Tel: +86-21-5877-1818, Fax: +86-21-6887-7858 / -7898

Renesas Electronics Hong Kong Limited
Unit 1601-1613, 16/F., Tower 2, Grand Century Place, 193 Prince Edward Road West, Mongkok, Kowloon, Hong Kong
Tel: +852-2886-9318, Fax: +852 2886-9022/9044

Renesas Electronics Taiwan Co., Ltd.
13F, No. 363, Fu Shing North Road, Taipei, Taiwan
Tel: +886-2-8175-9600, Fax: +886 2-8175-9670

Renesas Electronics Singapore Pte. Ltd.
80 Bendemeer Road, Unit #06-02 Hyflux Innovation Centre Singapore 339949
Tel: +65-6213-0200, Fax: +65-6213-0300

Renesas Electronics Malaysia Sdn.Bhd.
Unit 906, Block B, Menara Amcorp, Amcorp Trade Centre, No. 18, Jln Persiaran Barat, 46050 Petaling Jaya, Selangor Darul Ehsan, Malaysia
Tel: +60-3-7955-9390, Fax: +60-3-7955-9510

Renesas Electronics Korea Co., Ltd.
11F., Samik Lavied' or Bldg., 720-2 Yeoksam-Dong, Kangnam-Ku, Seoul 135-080, Korea
Tel: +82-2-558-3737, Fax: +82-2-558-5141