To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

H8S/H8SX Families

Direct Drive LCD Demo Application Note

Introduction

The purpose of the LCD Direct Drive demo is to demonstrate how to create an interactive TFT-LCD panel application through Renesas DirectLCD API in a real-time environment on one Renesas LCD Direct Drive demonstration platforms which supports the external DMA (ExDMA). This document will explain in detail the program and source structure of the demo application.

The user of this document should also refer to the "Direct Drive LCD Design Guide.pdf" and "GAPI User Manual.pdf" (contained within the demo project) for more details on the operation of the demonstration code.

We selected FreeRTOS (open source) as the real-time environment for the Direct Drive LCD demo. The technical documents of FreeRTOS could be accessed at www.freertos.org.

Target Device

H8S2378, H8S2456, H8S1668R

Target LCD Panels

Hitachi, Kyocera, Sharp, Optrex

Contents

1. Source Code Structure

The entire source code directory, as shown in the **Figure 1**, contains four subdirectories. The content of each subdirectory will be discussed as follows.

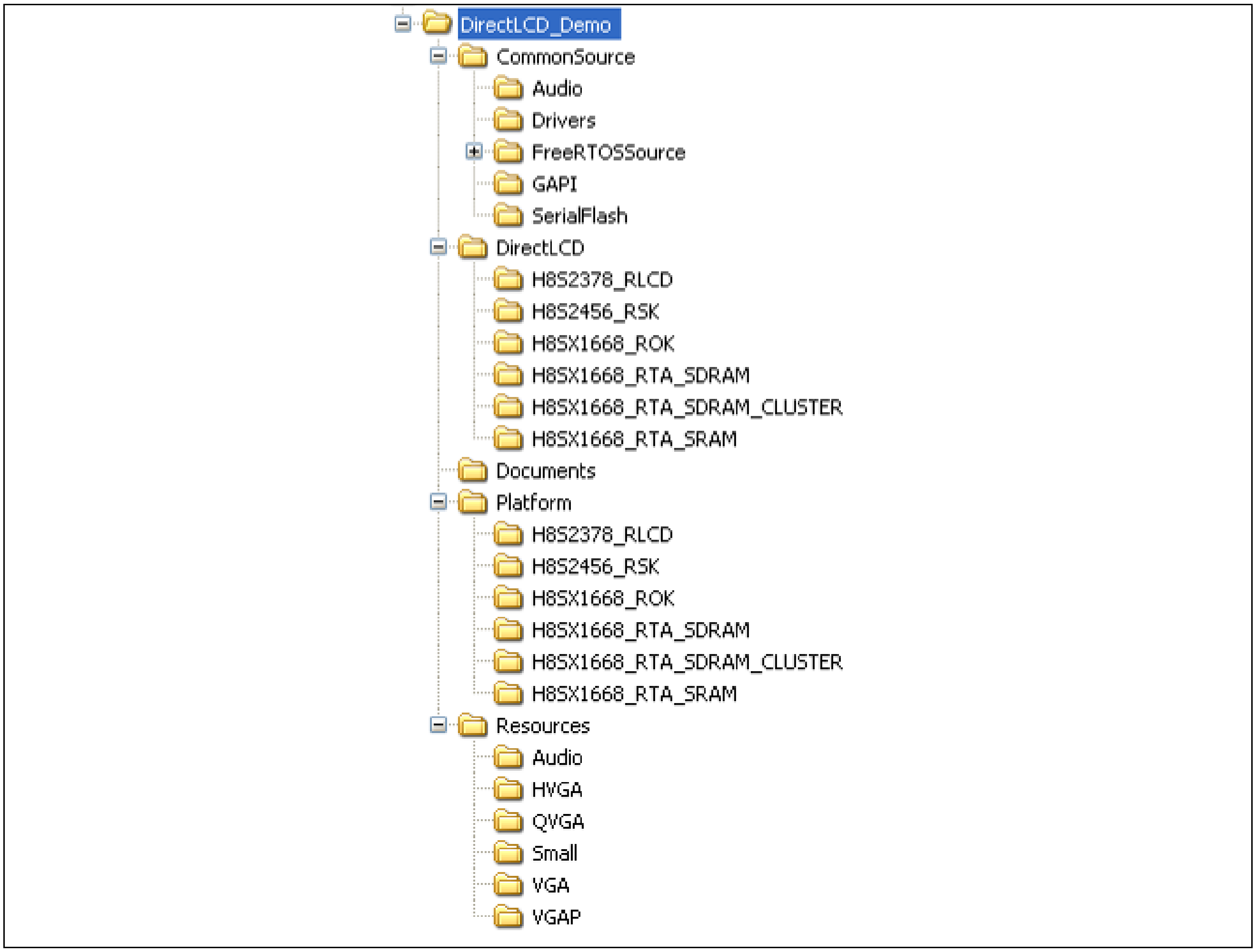


Figure 1

1.1 CommonSource

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

CommonSource (Project Demo Files)

File Name	***File Description***
main.c	Main program
Global.h	Global definitions for LCD Direct Drive
Frames.c	Initialize display frame buffers.
ScreenXXXX.c	Files that contain demo screen code. Each screen is contained in a corresponding file.
EventMgrTask.c	Implementation of event manager (touch and process)

ExMemory.c	External memory manager of RLCD demo
ExPool.c	External memory pool manager
HWSetup.c	Link to platform specific hardware setup
icon.c	Icon and Screen support functions of RLCD demo
Simple_printf.c	Simplified (small memory) Printf routines
FindFile.c	Accesses files located in the resource binary
Resources.c	Loads and stores data from serial flash
touchscreen.c	Touch screen driver of RLCD demo

Audio (PCM Audio Support)

File Name	***File Description***
Audio_WAV.c	Utilities to provide access to WAV file records
Streamaudio.c	PCM Audio playback demonstration code
ScreenAudio.c	Demonstration screen for Audio Code

Drivers (LCD Direct Driver)

File Name	***File Description***
DirectLCD_SBF.c	LCD Direct Driver for H8S devices.
DirectLCD_XBCFT.c	LCD Direct Driver for H8SX devices.
DirectLCD.h	The header file of LCD Direct Driver API
DirectLCD_CNF(Kyocera_QV).h	LCD panel definition of Kyocera QVGA
DirectLCD_CNF(Kyocera_VG).h	LCD panel definition of Kyocera VGA
DirectLCD_CNF(Sharp_T3).h	LCD panel definition of Sharp HVGA (WQVGA)
DirectLCD_CNF(Sharp_V7).h	LCD panel definition of Sharp VGA

FreeRTOSSource

File Name	***File Description***
croutine.c	Supports Co-Routine functions in FreeRTOS
list.c	Supports List functions
queue.c	Supports Queue functions
tasks.c	FreeRTOS Kernel

Note: MCU specific macros and parameters are in the *FreeRTOSPort.h* which resides in the platform specific directory. The details can be found in the CommonSource\FreeRTOSSource\portable\Renesas\H8S_H8SX directory.

GAPI (Renesas Graphics APIs)

File Name	***File Description***
gapi.h	Structure definitions and function prototypes for Renesas Graphics API
gapi_bmp.c	BMP support functions for Renesas Graphics API
gapi_button.c	Support for drawing buttons in Renesas Graphics API
gapi_copy.c	BMP copy functions for Renesas Graphics API
gapi_effects.c	Higher level effects for Renesas Graphics API
gapi_font.c	Font support for Renesas Graphics API
gapi_fill.c	Color/Block fill functions for Renesas Graphics API

Serial Flash

File Name	***File Description***

SPI_viaSSU.c	Driver layer for connection to the serial flash via SSU
SPI_viaSCI.c	Driver layer for connection to the serial flash via SCI
FlashSerialAtmel.c	Serial Flash commands interface for Atmel serial flash
FlashSerialSST.c	Serial Flash commands interface for SST serial flash

1.2 DirectLCD

This subdirectory contains HEW project and session files. Also, there is a project build directory for each platform configuration contained in this directory. No project source files are located here.

1.3 Platform

This subdirectory contains hardware platform configuration specific files. Each configuration contains an equivalent set of files. All of these files are "included" in the through the HEW include search path of "$(WORKSPDIR)\Platform\$(CONFIGNAME)".

File Name	***File Description***
hwsetup_platform.c	Platform configuration specific hardware configuration file.
hwsetup.h	Platform global hardware macro definitions.
DirectLCD_CNF(platform).h	Direct Driver hardware platform porting definitions.
DirectLCD_CNF.h	Direct Driver configuration definitions.
FreeRTOSPort.h	Platform specific RTOS porting definitions
iodefine.h	MCU Specific SFR definitions.

1.4 Resources

This directory contains the common non-source files that will be used in the demo application. The root directory contains the files being used in the current project, while the subdirectories contain resource which are sized to fit specific display panels. So, depending on which size panel you have, **you will need to replace the bitmaps in the root directory (Resources) with the appropriately sized ones (copy contents of resolution specific directory and paste)**.

2. Program Structure

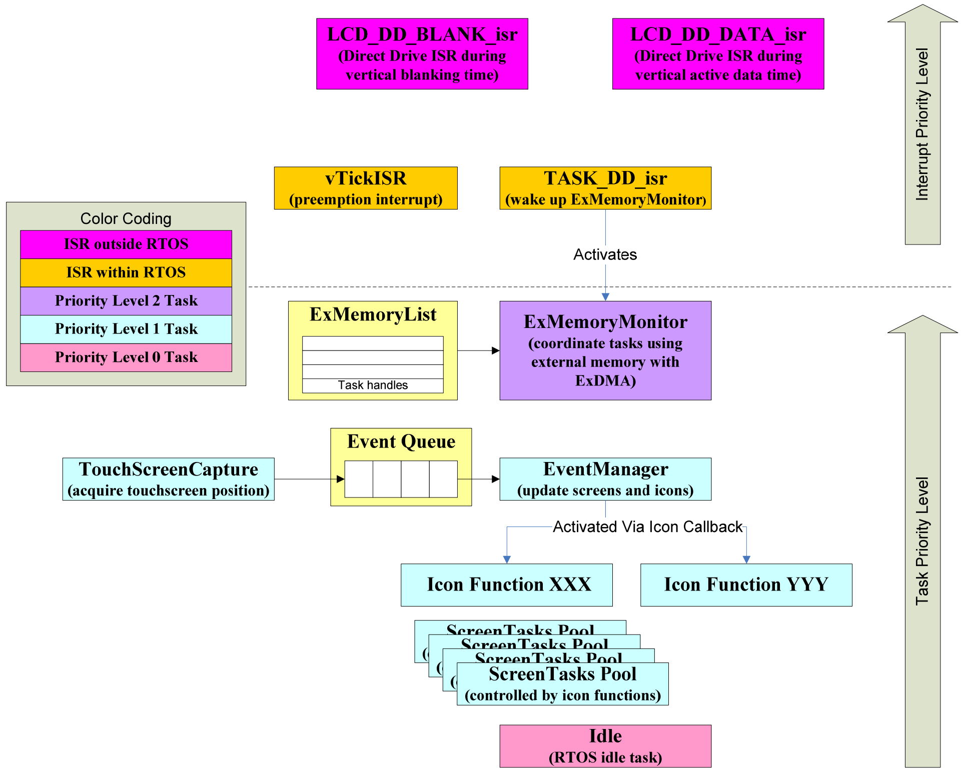
The following figure illustrates the interrupt service routines (ISRs) and tasks that are running the demo software.

The "Direct Drive" driver is contained within the two "LCD_DD_isrs". 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 a different ISR is active during the data transfer portion of the refresh cycle. These ISRs are triggered once per horizontal period of the refresh cycle.

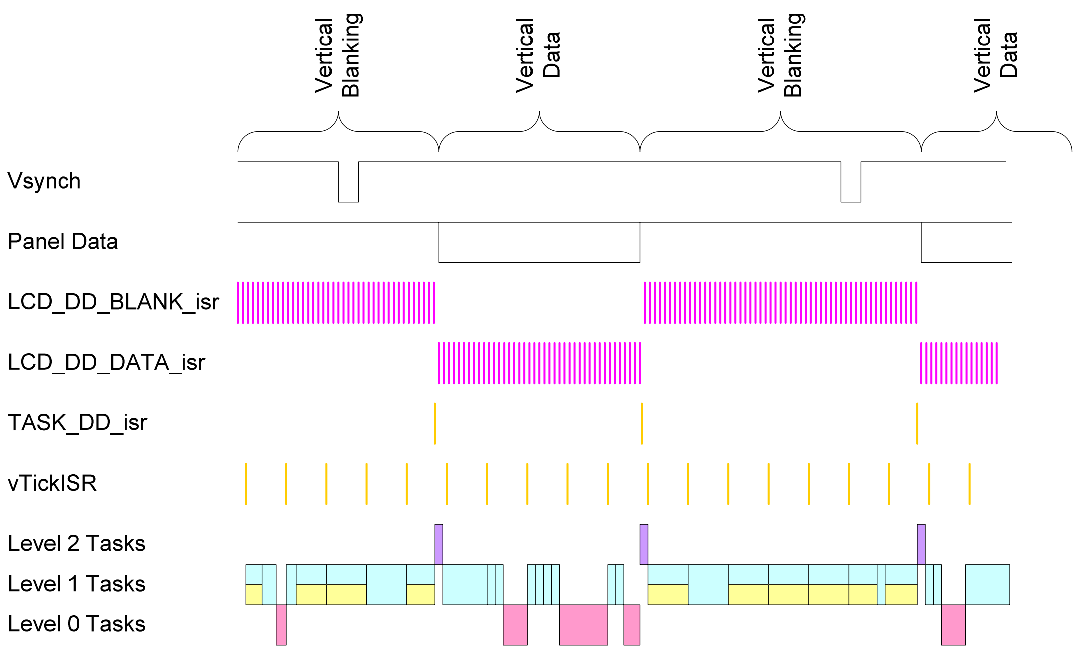
Twice per cycle (once before the data transfer starts and once after), the TASK_DD_isr is triggered to activate the "ExMemoryMonitor" task. This task is responsible for coordinating accesses to the external bus by software. Tasks that use the external bus (typically to update the frame RAM) are required to call the "ExMemoryAcquire" function to be registered in the "ExMemoryList" list. As many tasks as necessary can be added to this list within the configured size of the list (default is 16). The "ExMemoryMonitor" task suspends all tasks in this list at the beginning of the vertical data transfer and resumes these tasks at the beginning of the vertical blanking period. When a task will not be using external RAM for a period of time, the task can remove itself from the list by calling "ExMemoryRelease". The consequence of accessing the external bus without registering in the list is a 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.

The last ISR in the system is the "vTickISR" which causes the RTOS 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.

The majority of the tasks in the code are responsible for the behavior of the demo. The "TouchScreenCapture" task is responsible for cursor position and icon selection. This cursor information is sent through an Event Queue to the "EventManager" task. The queue mechanism was used to easily accommodate additional cursor movement from other sources (such as a mouse or USB). The "EventManager" task uses a mechanism to determine if buttons on the current screen have been activated. If a button is activated, its associated callback function will be called. In the cases where the button cannot complete a requested task (such as the countdown timer), it will launch the "longer term" function within a task (requested from the ScreenTasks pool).

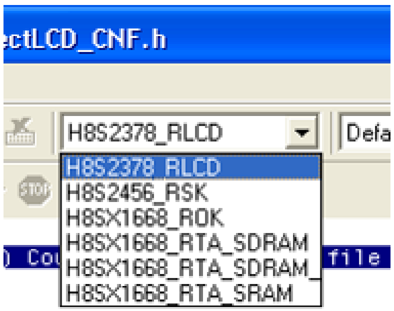


The next figure shows the interaction of the ISRs in relation to the LCD panel Vsynch signal. It also shows how tasks that have been registered as currently accessing the external bus (shown two-tone blue/yellow) are suspended during the "vertical data" portion of the period.



3. Configuration of RLCD Project in HEW

The first step in configuring the Direct Drive project is to select the appropriate configuration from the configuration pull down:



The next step is to ensure that the “DirectLCD_CNF.h” file for this configuration includes an appropriate header file for the LCD panel and operation mode desired. Please refer to the “Direct Drive LCD Design Guide.pdf” for more details.

```
#include "DirectLCD_CNF(Kyocera_QV).h"

// Specify Desired system behavior...will adjust vertical front porch to meet request.
#define DOT_CLOCK_FREQUENCY_DATA 16000000L
#define DOT_CLOCK_FREQUENCY_BLANK DOT_CLOCK_FREQUENCY_DATA
#define DESIRED_FRAME_RATE 60
#define MINIMUM_MCU_ACCESS_PCT 30
```

Finally ensure that the proper resources are copied into the “Resource” directory.

At this point you should be able to build the project run on your connected target.

4. Home Screen Generation

To facilitate the quick and simple inclusion of objects on the “Home” screen, the Icon Table for the home screen is created by the linker. This is accomplished by each top level screen file creating an Icon record in a special section (named _SCR_HM). When the linker runs, all of these records are collected into a single memory structure and this is referenced by the code as the “home” screen icons. Using this mechanism, screen files can easily be added and removed from the project (as their _SCR_HM records are added or removed by the linker). The following show an example of this code (and is used by each top level screen file).

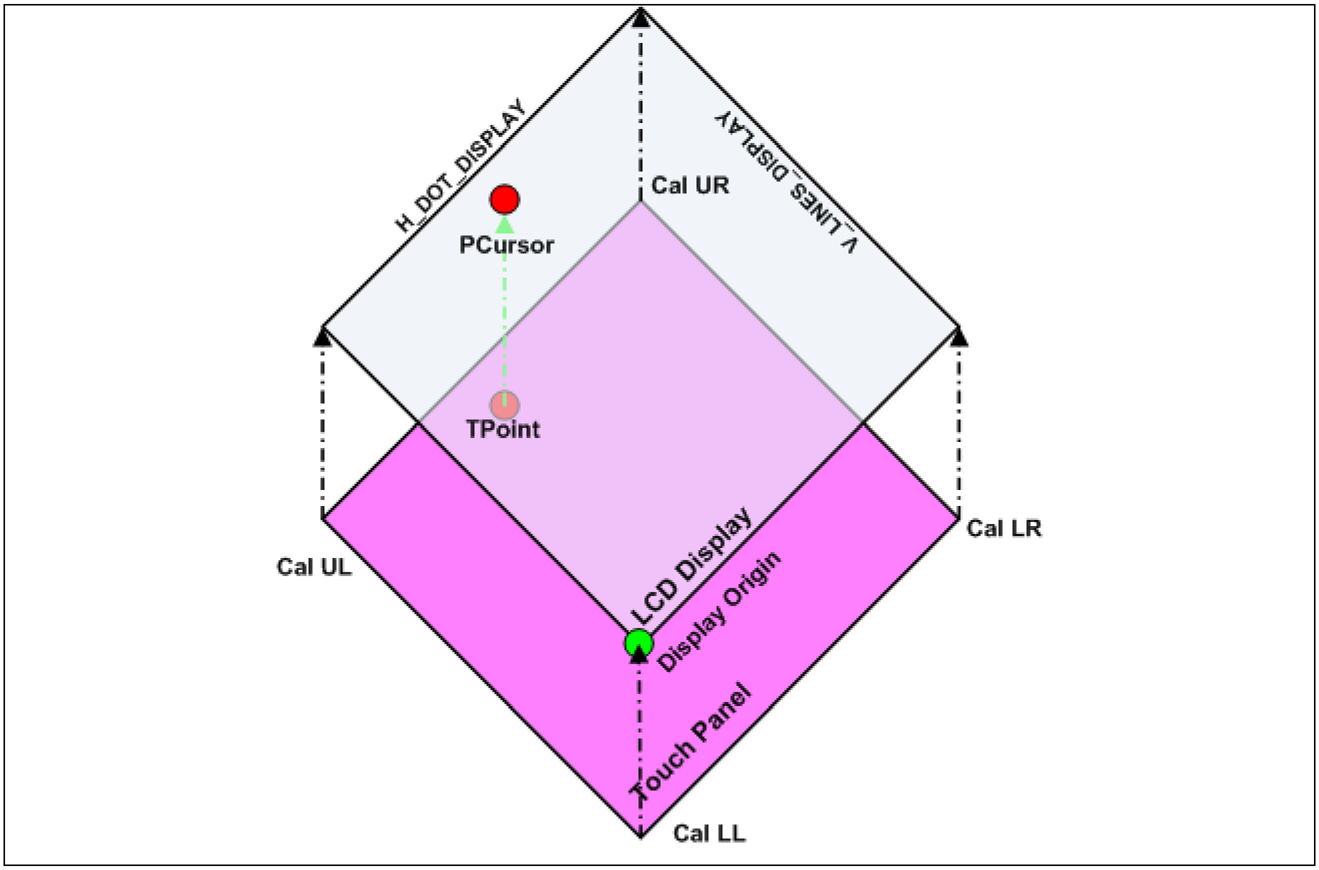
```
/* Include this in the Home Screen...the _SCR_HM section will be used as the Icon list for the home screen

   - By doing this, screens can be added (or removed) simply by linking them into the project */

#pragma section _SCR_HM

static const ICON_type HomeIcon= { &BMP_ButtonR, T_SchemeGreen,  ScreenAnimate,       SX(0.375), SY(0.783) };

#pragma section
```

5. The Calculation of Touch Screen Touch Position

The demonstration code currently uses a four point calibration mechanism. Initial calibration points are stored in the file “TouchScreenCal.h”. These initial values often are adequate for the purposes of our demonstration code, but different touch panels or platform hardware may require different calibration settings. In the demonstration code, a calibration screen can be entered by selecting from the touch screen (or pressing one of the physical platform buttons).

This calibration screen will request the user to touch each of the touch screen corners in turn and then display the newly acquired calibration constants (which can then be entered into "TouchScreenCal.h" for future use).

The 4-wire touch screen inputs are obtained by a task that sequentially drives the X-plane of the touch screen and reads the Y-plane on the A/D converter (and vice-versa). The A/D conversions during this acquisition are oversampled by using the MCU's DTC (Data Transfer Controller) peripheral capabilities. These acquired samples are checked for touch condition, filtered, debounced and then finally linearized by the calibration constants to provide user events in screen coordinates.



6. Demonstration Platform Resource Storage

In the demo program, resources (BMP's, Fonts, Audio files, etc) are accessed from a resource image file located in the linear memory of the MCU. This resource image file can be located in internal flash or external RAM. In the case of external RAM, the resource file is transferred to RAM on power on from serial flash by the demo application code.

This application code accesses information in the resource image file by resource name by use of the "FileFind()" function.

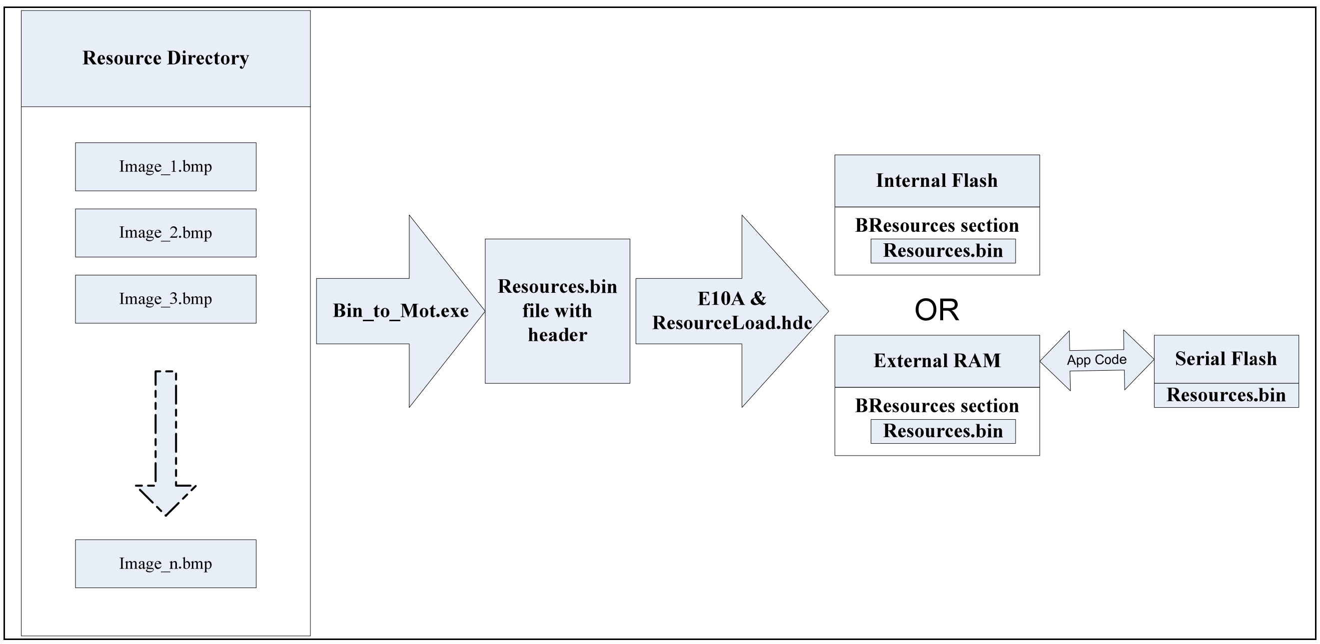
The following table illustrates the format of the file header resource image file. Each entry is 32 bytes long. The filename does not include the extension (in this example, "Image_1.bmp" would become "Image_1". The first record provides size information for the entire resource file.

	Record Name (max 24 characters)	Location (4 Bytes)	Size (4 Bytes)
0x000000	"BFS_Header"	0x00000000	Resource File Size
0x000020	"Image_1"	Imagae_1 offset location	Image_1 size
0x000040	…		

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).
The 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** (on platforms without serial flash), the resource file is loaded whenever the project code (DirectLCD.abs) is downloaded to the target by use of a debugger settings load script.

If the resource file is located in **serial flash**, the resource file is loaded by manually running the "ResourceLoad.hdc" script for the platform from the "Command Line" window of HEW. Please note that during the programming of serial flash, the code may require up to 1 minute to complete programming prior to the application starting (dependent on Resources.bin size)…please be patient! Also realize that 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).



7. External Parallel Storage

Images also could be stored in the external flash and accessed the same as illustrated previously via the FileFind() utility.

The Bin_to_Mot.exe is a command line program that converts any binary file (e.g. bitmaps) to Motorola S-record, Intel, or binary formats. As well, it combines multiple files and adds a header to the output image with location and size information. The Bin_to_Mot.exe program is located in the $(WORKSPACEDIR) directory of your project. Then open a "Command Prompt" window (under Start > Programs > Accessories) and running the program.

Usage: Bin_to_mot -[b/c/i/m][aaaaaaaa] [-B][-f] file(s)_in file_out

Example for the purpose of combining bitmaps into a single S-record file (shown as in the **Figure 9**):
C:\...\Bin_to_mot –m400000 "image1.bmp" "image2.bmp" "image2.bmp" **srecord.mot**

Where '–m' specifies Motorola (S record) output, "image1.bmp", "image2.bmp", "image3.bmp" are the image files to convert and '**srecord.mot**' is the output file. The '400000' is an offset in hex. The external flash

ROM address starts at 0x400000. A batch file is included in the "Utilities" folder that was used to create the image file for the demos included with the kit.

The following table illustrates the format of the file header that is output by the Bin_to_mot program. Each entry is 32 bytes long. The filename does not include the extension. The first record provides size information for the entire resource file.

	Record Name (max 24 characters)	Location (4 Bytes)	Size (4 Bytes)
0x000000	"BFS_Header"	0x00000000	Resource File Size
0x000020	Image_1	Image_1 offset location	Image_1 size
0x000040	…		

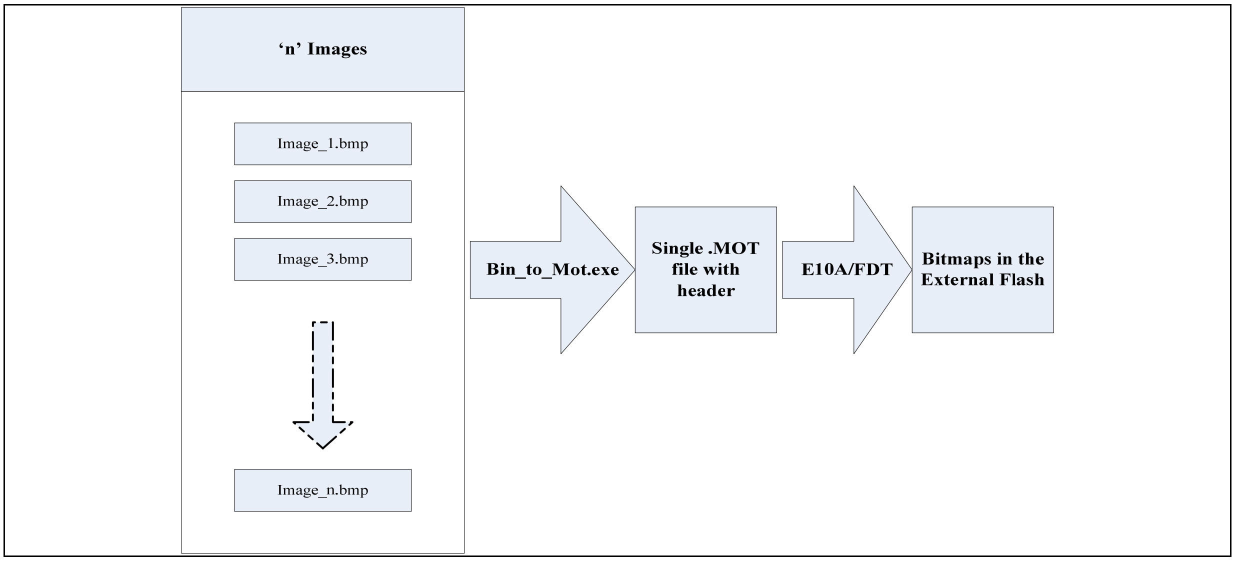
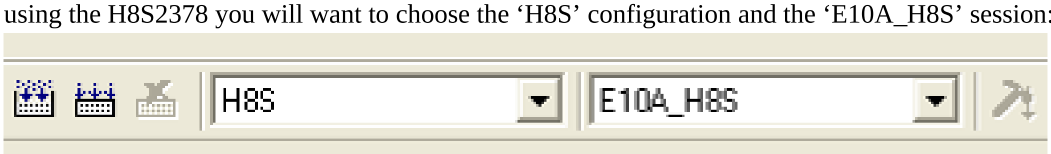


Figure 9

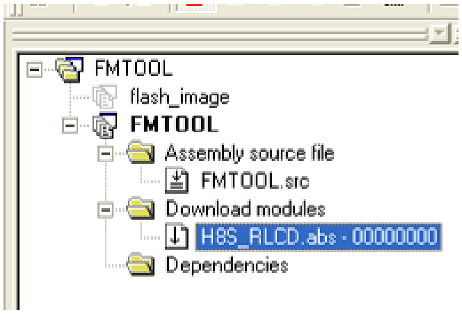
Instructions for programming the External Flash using the E10a debugger are show below:

1) Start off by loading an assembly application into RAM that has routines for writing and erasing the memory. Once you have loaded the project you will need to select the correct configuration and session. If you are using the H8S2378 you will want to choose the 'H8S' configuration and the 'E10A_H8S' session:

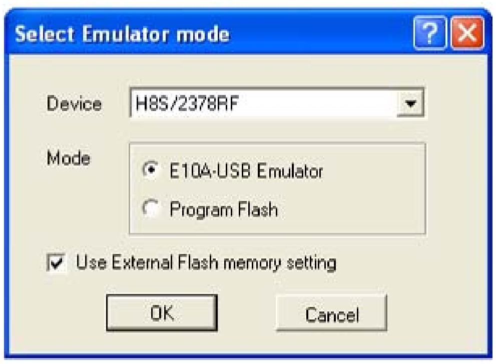


If you are using the H8SX1668 then you will select the 'H8SX' configuration and the 'E10A_H8SX' session.

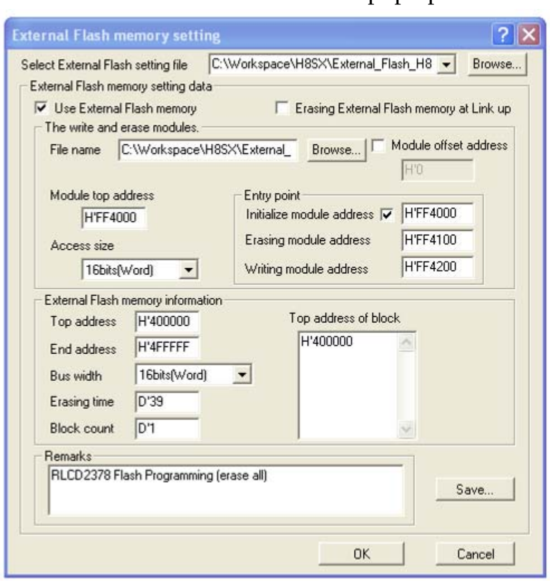
2) Build the project and then connect to the board. Once connected double click on the file under "Download Modules" to download the application to the board.



3) Now that the memory loader application is in RAM you can now move onto the project where you have the file you want to download into flash. When you connect to your board, make sure to check the "Use External Flash memory setting" box:



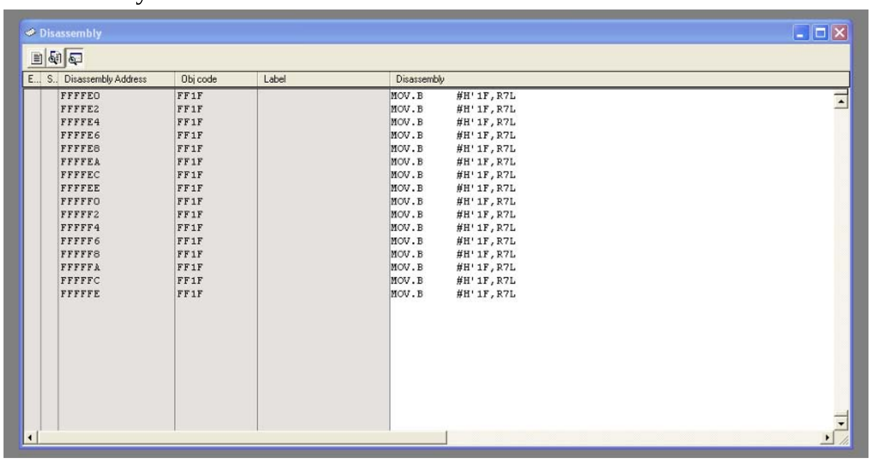
4) Click OK and this window should pop up:



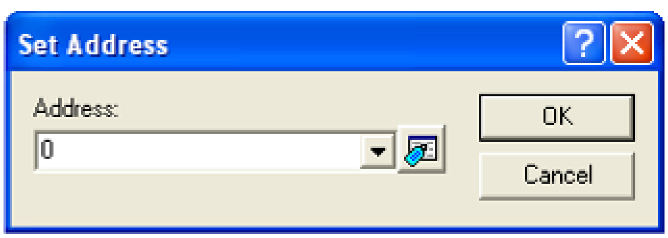
5) For the 'Select External Flash setting file' option at the top click the 'Browse' button. Navigate to the FMTOOL directory that was used earlier and look for the file 'RLCD2378.EFF' under the 'Debug' directory.

You will use this file whether you are using the H8S part, or the H8SX part. Once you have chosen this file you will then need to click the browse button next to the 'File name' option. Navigate to the FMTOOL directory again and choose the .mot file that you created earlier. If you are using the H8S2378 board then the mot file will be under the 'H8S' directory. If you are using the H8SX1668 board then the mot file will be under the 'H8SX' directory. Once you have the mot file selected, everything else should be fine so go ahead and click OK.

6) Once the connection is finished you are free to download the file you want to flash. If you have a mot file then go ahead and double click on the file underneath 'Download Modules'. After double clicking on the file a window should pop up and blue bar should flash across. This means that the file has been sent to the cached version of the memory on your system. To make the board actually write the information into the external memory you will have to execute an instruction. To do this click on Debug -> Reset CPU. This should open a 'Disassembly' window:

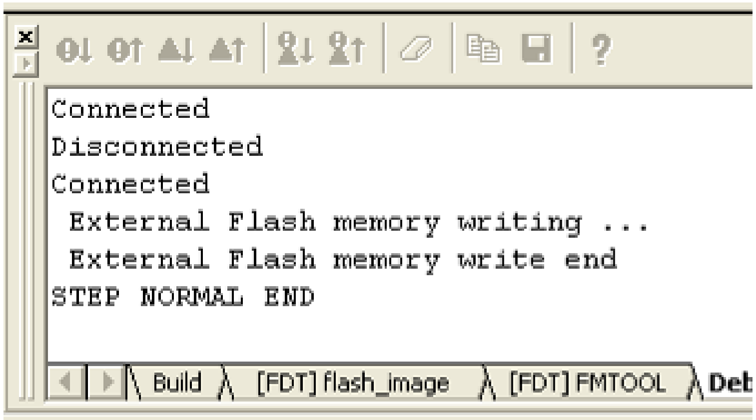


In the 'Disassembly' window double click on an entry in the "Disassembly Address' column. In the set address window that pops up set the address to 0 and click OK.

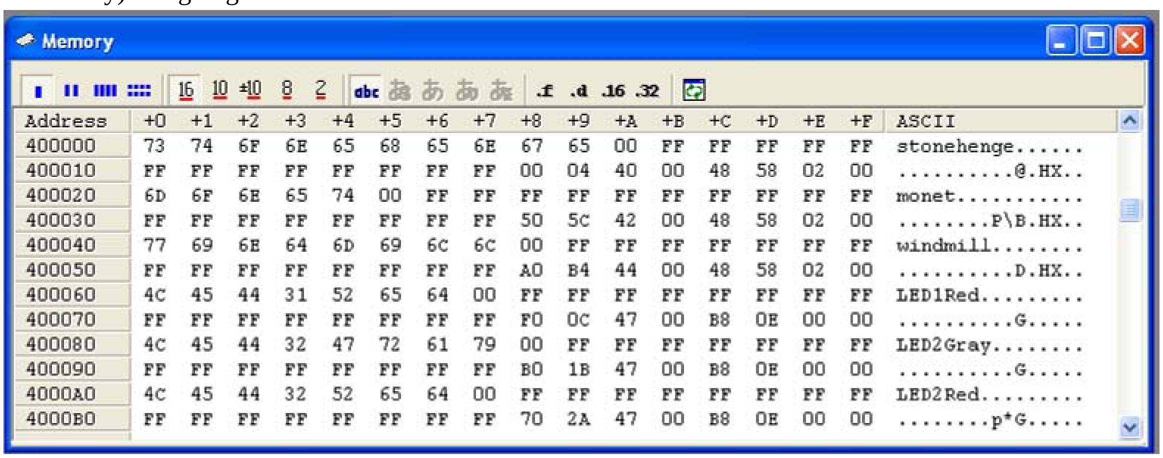


Now on any entry under the 'Disassembly' column right click and choose 'Set PC Here'. Next, under the Debug drop down click 'Step In'. This should start the transfer to memory and you should see this in the

'Debug' window when it is done:



7) You can check that your file has been downloaded to memory by opening a memory window (View -> CPU -> Memory) and going to the location where the data should be.



8. OS API Abstraction Layer

In order to provide the portability amongst various RTOS, RLCD kit supplies the OS API Abstraction layer to achieve such a goal. The APIs of this layer cover the key components of RTOS, Task, Queue, Semaphore, and Kernel. For the details, please refer to ***OSLayer.h***

Task

RLCD_TaskCreate - create a new task and add it to the list of tasks that are ready to run.

RLCD_TaskDelayUntil - delay a task until a specific time.

RLCD_TaskDelay - delay a task for a given number of ticks.

RLCD_TaskSuspend - suspend a given task.

RLCD_TaskYield - force a context switch

RLCD_TaskResume - resume a given suspended task

RLCD_GetTaskHandle - get the current running task handle

RLCD_GetTaskTickCount - get the count of ticks since the task scheduler is running

Queue and Semaphore

RLCD_QueueCreate - create a new queue instance.

RLCD_QueueSend - post an item on a queue.

RLCD_QueueReceive - receive an item from a queue.

RLCD_BinarySemaphoreCreate - create a binary semaphore

RLCD_BinarySemaphoreTake - obtain a binary semaphore

RLCD_BinarySemaphoreGive - release a binary semaphore

Kernel Functions and Parameters

RLCD_OSStart - start the real-time task scheduler

demotaskSTACK_SIZE - stack size for the context switch

mainTOUCHCAL_PRIORITY - task priority of Touch Screen Calibration

mainBUSMONITOR_PRIORITY - task priority of exDMAC Bus Monitor

mainEVENTMGR_PRIORITY - task priority of Event Manager

mainDEMOTASKS_PRIORITY - task priority of Demo tasks

TICK_RATE_MS - tick rate (milliseconds)

Website and Support

Renesas Technology Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/inquiry
csc@renesas.com

Revision Record

Rev.	Date	Description	
		Page	Summary
1.00	Aug.20.08	-	First edition issued
2.00	Dec.04.08	-	Second edition issued
2.01	June.12.09	-	Modified project structure.
2.02	Sept.30.09	-	Updated resource storage section. Typographic Corrections.