

MC9S12HY-Family Demonstration Lab Training

by: Gordon Borland,
Steven McLaughlin
Microcontroller Solutions Group, Auto R & D

1 Introduction

This publication serves to document the demonstration lab software examples. The examples describe how to configure and use the modules for users to get started with the MC9S12HY Family of MCUs.

The examples included here illustrate a basic configuration of the modules to allow users to quickly start developing their own applications.

Complete code is available for all examples. This can be downloaded onto an MC9S12HY64 target such as the DEMO9S12HY64 demo board upon which this demonstration lab is based.

Each module of the MC9S12HY Family has its own standalone software and is discussed within the respective section of this document.

Contents

1	Introduction	1
2	Setup	2
2.1	Tools Setup	2
2.2	Board Setup	2
3	Demonstration Lab Examples	2
3.1	CPMU Clocks	2
3.2	Flash Programming Example	4
3.3	Emulated EEPROM Driver	8
3.4	MSCAN Module	13
3.5	PWM Module	14
3.6	S12HY Low Power Modes	15
3.7	MMC Program Flash Paging Window	17
3.8	ADC Module	20
3.9	Timer Module	21
3.10	SCI Communications	23
3.11	SPI Communications	24
3.12	Motor Control Module	25
3.13	LCD Module	26
4	Conclusion	27
5	References	28

2 Setup

2.1 Tools Setup

NOTE

Before starting any of the module examples in this document, it is important that you install CodeWarrior Development Studio and CodeWarrior for Microcontrollers as described in the **DEMO9S12HY64 Quick Start Guide** that accompanies the demonstration board.

2.2 Board Setup

The following steps provide a basic configuration for each of the module examples in this document. Any deviation from this basic configuration or any specific requirements for a module will be outlined in the relevant module chapter.

1. The DEMO9S12HY64 board should be configured with the default jumper settings as shown in the **DEMO9S12HY64 Quick Start Guide** that accompanies the demonstration board.
2. Connect an A/B USB cable to an open USB port on the host PC and the USB connector on the DEMO9S12HY64 demonstration board. Follow the on-screen instructions to install the necessary USB drivers if required.
3. Move the ON/OFF switch (SW5) to the ON position.
4. The green "+5V" LED above the ON/OFF switch should illuminate.

3 Demonstration Lab Examples

3.1 CPMU Clocks

This lab example shows how to produce PLL based bus clocks using different clock modes of the CPMU module. The example software initializes the PLL to run in default 8 MHz PEI mode, 12.5 MHz PEI mode, 32 MHz PEE mode, and 4 MHz PBE mode. The changes in bus clock can be observed via the pulse rate of LED1 and the frequency can be measured by monitoring the ECLK signal (Bus clock) on an oscilloscope.

3.1.1 Setup

The following steps should be completed before running the lab example:

1. Start CodeWarrior™ by selecting it in the Microsoft Windows Start menu.
2. From the CodeWarrior main menu, select **File > Open** and select **S12HY CPMU Demo.mcp** file.
3. Click on the **Open** button. The project window will open.
4. The C code of this demonstration is contained within the **main.c** file. Double-click on the file to open it.

5. From the main menu, select **Project > Debug**. This will compile the source code, generate an executable file, and download it to the demo board.
6. A new debugger environment will open. After the download to the demo board is completed, close the debugger environment.
7. The PLL configuration is sent to the serial communications port on the DEMO9S12HY64 board (baud rate = 9600, data bits = 8, parity = N, stop bits = 1, Flow control = Hardware). Open a terminal window on the PC with this configuration.
8. The bus clock speed is represented on pin 59 PH2/ECLK. The ECLK signal is equivalent to the MCU bus speed and can be monitored by attaching an oscilloscope probe to pin 23 of the **DS1** header.

3.1.2 Instructions

Follow these instructions to run the lab example:

1. Hit RESET. The MCU is now running in it's default PEI mode with a bus clock frequency of 8 MHz.
2. Monitor the ECLK signal on the oscilloscope. ECLK will match the bus clock frequency. Observe the LED pulse rate. Examine the PLL configuration on the terminal window.
3. Hit RESET whilst pressing down SW1. The MCU is now running in PEI mode with a bus clock frequency of 12.5 MHz.
4. Monitor the ECLK signal on the oscilloscope. ECLK will match the bus clock frequency. Observe the LED pulse rate. Examine the PLL configuration on the terminal window.
5. Hit RESET whilst pressing down SW2. The MCU is now running in PEE mode with a bus clock frequency of 32 MHz.
6. Monitor the ECLK signal on the oscilloscope. ECLK will match the bus clock frequency. Observe the LED pulse rate. Examine the PLL configuration on the terminal window.
7. Hit RESET whilst pressing down SW3. The MCU is now running in PBE mode with a bus clock frequency of 4 MHz.
8. Monitor the ECLK signal on the oscilloscope. ECLK will match the bus clock frequency. Observe the LED pulse rate. Examine the PLL configuration on the terminal window.

3.1.3 Summary

The CPMU PLL has three modes, PLL Engaged Internal Mode (PEI), PLL Engaged External Mode (PEE), and PLL Bypassed External Mode (PBE).

From reset, the bus clock is derived from the CPMU PLL using the 1 MHz internally generated reference clock as it's source (PEI mode). This is the default clock mode from reset and will produce a bus clock of 8 MHz.

For further information on the CPMU module please refer to the following documentation which is available at www.freescale.com.

- Application note titled *Comparison of the S12XS CRG Module with S12P CPMU Module* (document AN3622)
- *MC9S12HY64 Reference Manual* (document MC9S12HY64RM)

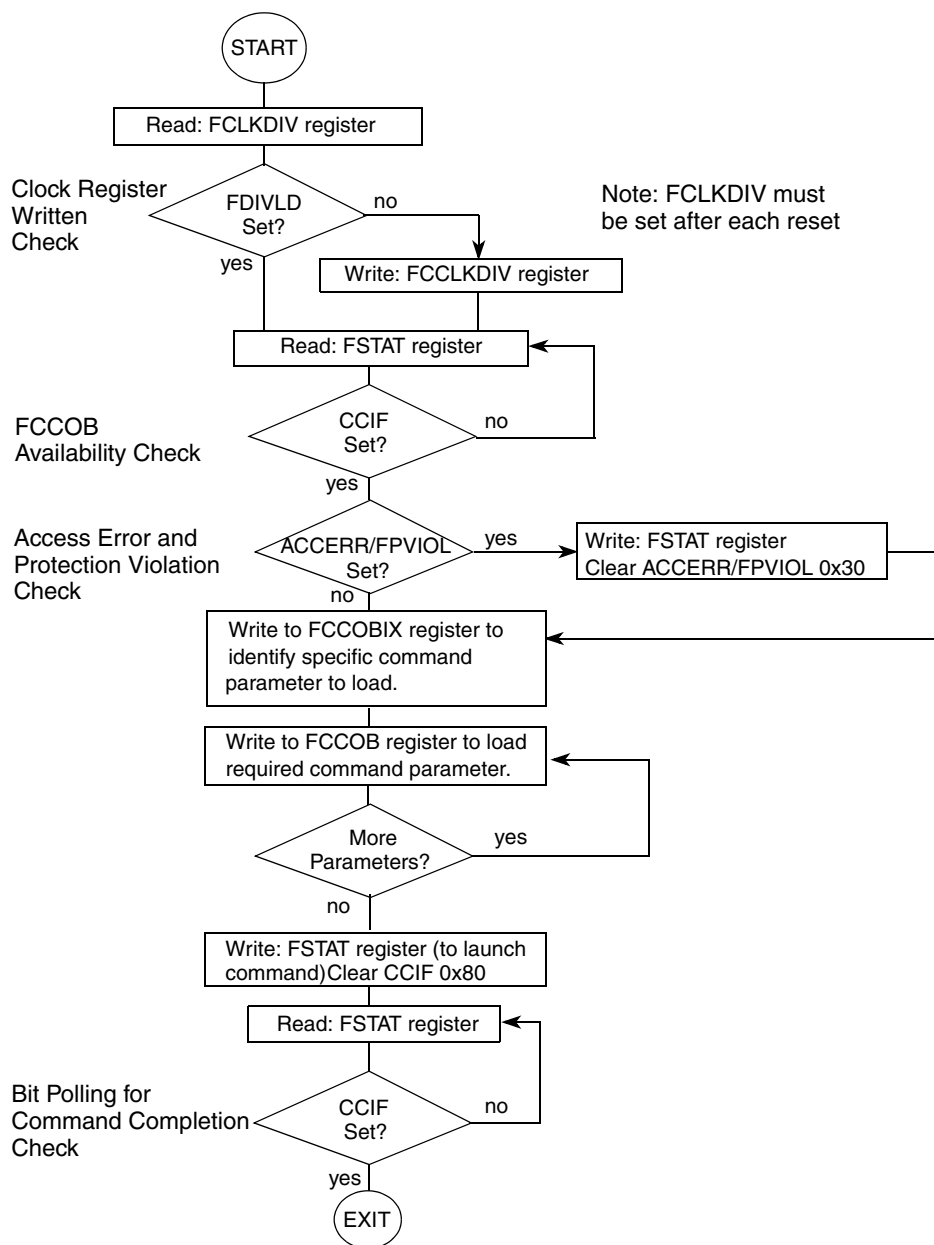
3.2 Flash Programming Example

3.2.1 Flash Overview

The flash technology module (FTM- a 1.5T split gate transistor flash technology) contains program flash (P-flash), and data flash (D-flash). P-flash is intended primarily for non-volatile code storage. D-flash is used as basic flash memory for non-volatile data storage or non-volatile storage to support emulated EEPROM or a combination of both. The user interfaces with this module via the following steps.

1. Set the flash clock divider (FCLKDIV).
2. Check the status of the Flash status register (FSTAT).
3. Make sure the command complete interrupt flag is set (CCIF=1).
4. Launch the appropriate flash commands (program, erase, verify and so on) via FCCOBIX and FCCOB registers.
5. Check flash status register and that CCIF=1.

A flow chart of these steps is shown below:



3.2.2 Code Example Explanation and Walkthrough

The user will note that this program is compiled and run from RAM, because sections of flash will be erased in this example. The security information (0xFF0F) in the flash configuration field is not erased.

NOTE

If the security information is accidentally erased, the part will be secured and cannot be re-programmed until it is unsecured.

The file of interest is **main.c**. The purpose of this demonstration is to show:

- Launching a flash command.

- Demonstrate programming and erasing of flash.

This example has been written with a series of user software breakpoints so that the only thing that has to be done is to hit the **Run**  button.

On start-up, the debugger should begin the program in 'main'.

3.2.3 Breakpoint 1 — Launch Flash Command — Filling P-Flash

The importance at this breakpoint is the LaunchFlashCommand function. This function is responsible for exercising the flash block depending upon the flash command given. The flash commands are briefly described in the flash.h header file and within the S12HY reference manual, section 15.4, "Flash Command Description." Stepping through will take the user to the function below.

```

/*****
Function Name : LaunchFlashCommand
Engineer      :
Date          :
Arguments     :
Return        :
Notes         : This function does not check if the Flash is erased.
                  This function does not explicitly verify that the data has been
                  successfully programmed.
                  This function must be located in RAM or a flash block not
                  being programmed.
*****/

tU08 LaunchFlashCommand(char params, tU08 command, tU08 ccob0, tU16 ccob1, tU16 ccob2, tU16 ccob3, tU16 ccob4, tU16 ccob5)
{
    if(FSTAT_CCIF == 1)
    {
        /* Clear any error flags*/
        FSTAT = (FPVIOL_MASK | ACCERR_MASK);

        /* Write the command id / ccob0 */
        FCCOBIX = 0;
        FCCOBHI = command;
        FCCOBLO = ccob0;

        if(++FCCOBIX != params) {
            FCCOB = ccob1; /* Write next data word to CCOB buffer. */
            if(++FCCOBIX != params) {
                FCCOB = ccob2; /* Write next data word to CCOB buffer. */
                if(++FCCOBIX != params) {
                    FCCOB = ccob3; /* Write next data word to CCOB buffer. */
                    if(++FCCOBIX != params) {
                        FCCOB = ccob4; /* Write next data word to CCOB buffer. */
                        if(++FCCOBIX != params) {
                            FCCOB = ccob5; /* Write next data word to CCOB buffer. */
                        }
                    }
                }
            }
        }
        FCCOBIX = params-1;

        /* Clear command buffer empty flag by writing a 1 to it */
        FSTAT = CCIF_MASK;
        while (!FSTAT_CCIF) { /* wait for the command to complete */
            /* Return status. */
        }
        return(FSTAT); /* command completed */
    }
    return(FLASH_BUSY); /* state machine busy */
}

```

Figure 1. Launch Flash command function

3.2.4 Breakpoint 2 - Launched Program Commands - Known Data

On entry of the second breakpoint, the memory maps have been set up to show the P-flash being erased (0xFFFF state), then programmed with known parameters (0xAAAA). All P-flash pages have been filled with 0xAAAA; 0x1400, 0x4000, 0xC000, and the PPAGE 0C-0F – 0x8000- 0xBFFF (local entry).

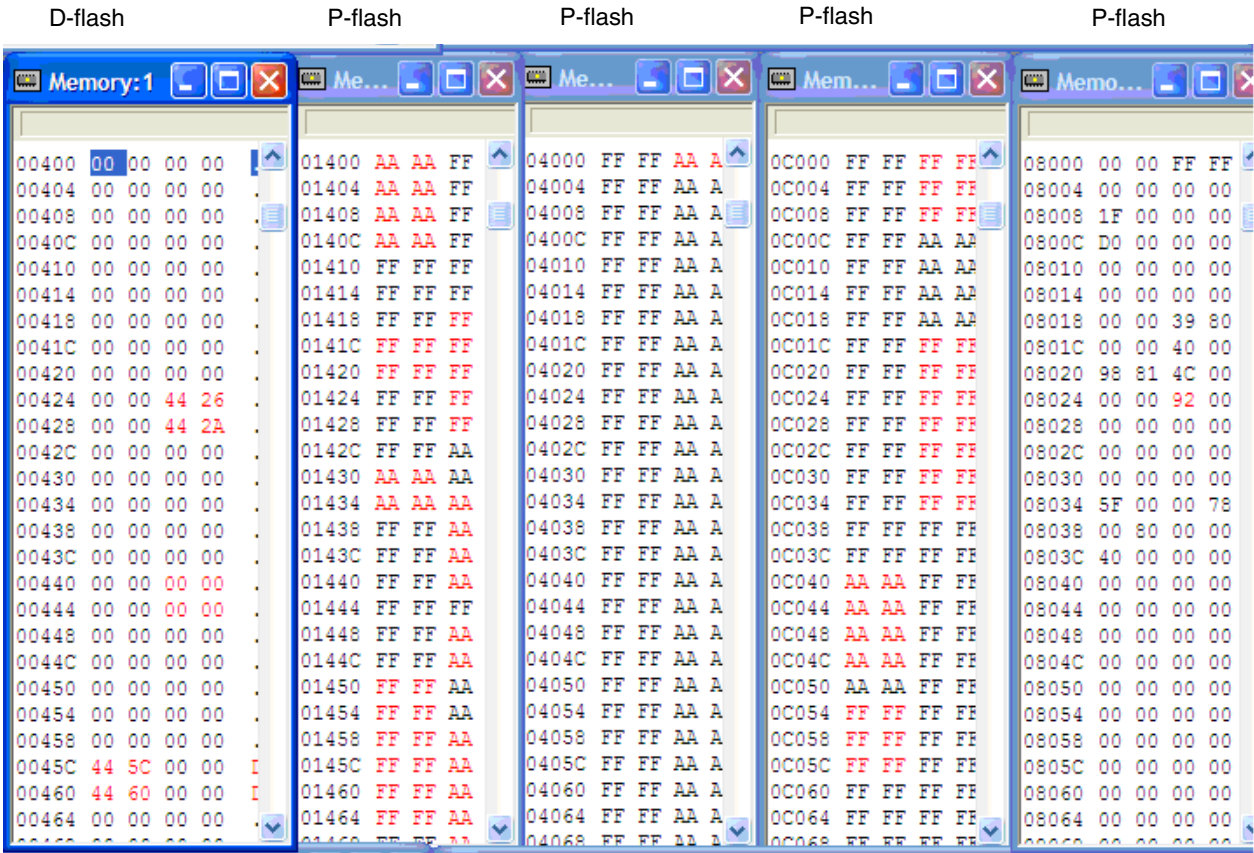
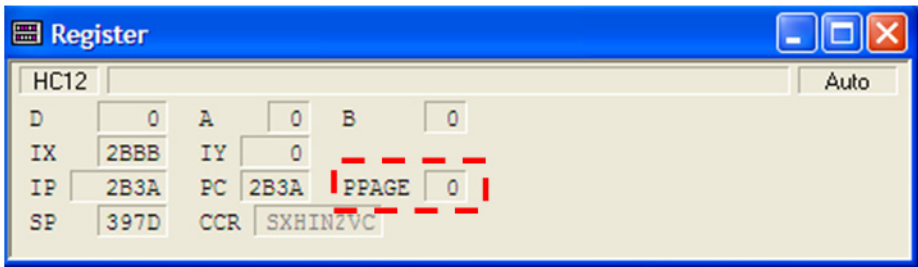


Figure 2. The memory maps showing real time erasing and programming

The user will note that 0x8000, the P-flash window will show different content depending on the PPAGE. When the programming has occurred, check PPAGE C-F by entering the character in the PPAGE register.



By altering the PPAGE, the user can see the 64k windows of programmed information at 0x8000.

3.2.5 Breakpoint 3 - Launched Program Commands - Address Data

Same as before, except the data being written to the P-flash is different; the data written is the actual addresses of the P-flash.

3.2.6 Breakpoint 4 - D-flash - Launched Program Commands

The same functions are used but will now perform activity on the D-flash. The only differences to the flash command function is the memory address issued and flash commands — in other words, D-flash instead of P-flash.

The end of the demonstration is indicated by the LEDs on the EVB being toggled. This example does not re-program the device to default, this will happen on the next re-load of a program by allowing NVM erasing.

3.2.7 Summary

The demonstration software has shown how to initialize the flash command to perform programming, erasing, and erase verify on both the P-flash and D-flash. It is vital that the flow diagram is followed for correct operation. Deviation from this could cause errors when working with the P/D-flash. Although this demonstration did not include it, it is good practice to verify that the correct data has been programmed to the flash.

3.3 Emulated EEPROM Driver

3.3.1 Emulated EEPROM Overview

Electrically Erasable Programmable Read-Only Memory (EEPROM), which can be byte or word programmed and erased, is often used in automotive electronic control units. This flexibility for program and erase operations makes it suitable for data storage of application variables that must be maintained when power is removed and needs to be updated individually during run-time. For the devices without EEPROM memory, the page-erasable flash memory can be used to emulate for EEPROM through EEPROM emulation software.

The EEPROM emulation driver for S12HY implements the fixed-length data record scheme emulation on split gate flash. The EEPROM functionality to be emulated include organizing data records, initializing and de-initializing EEPROM, reporting EEPROM status, reading and writing data records.

Four or more sectors shall be involved in emulation with a round robin scheduling scheme.

3.3.2 Code Example Explanation and Walkthrough

The file of interest is **NormalDemo.c** where the 'main' function resides. The purpose of this demonstration is to show how:

- The D-flash is initialized for EEE.
- The active and alternative sectors are assigned.
- The active sector is filled and swapped (and erased) with an alternative sector.

In an application only, the last point above is of relevance — the D-flash would be continually read and written to and sectors would be copied, swapped, and erased. This example has been written with a series of user software breakpoints so that the only thing that has to be done is to hit the **Run**  button.

On start-up, the debugger should begin the program in 'main'

```
void main(void)
{
    UINT16 returnCode;
    UINT16 readAddr;
    UINT16 erasingCycles;
    UINT16 temp[EED_DATA_VALUE_SIZE/EED_MIN_PROG_SIZE] = {0,0,0,0};
    UINT8 i,j;
```

3.3.3 Breakpoint 1 — Erase the D-flash

```
/* This erases the D-Flash sectors used for EED. The entire emulated EEPROM
   from EED_FLASH_START_ADDRESS to EED_FLASH_END_ADDRESS is erased. */
asm BGND;
/* User breakpoint1 - the function below will erase the D-flash */
returnCode = FSL_DeinitEeprom();

if (EED_OK != returnCode)
{
    ErrorTrap();
}

/* Wait until DeinitEeprom is done */
while(BUSY == EE_Status)
```

On selecting the run button, the first software breakpoint will be hit. Breakpoint 1 stops at the function which is responsible for initializing the D-flash. This function erases and assigns the physical D-flash which shall be used for EEE. The user will notice that the selected D-flash (0x400, 0x500, 0x600, 0x700) will be in the erased state.

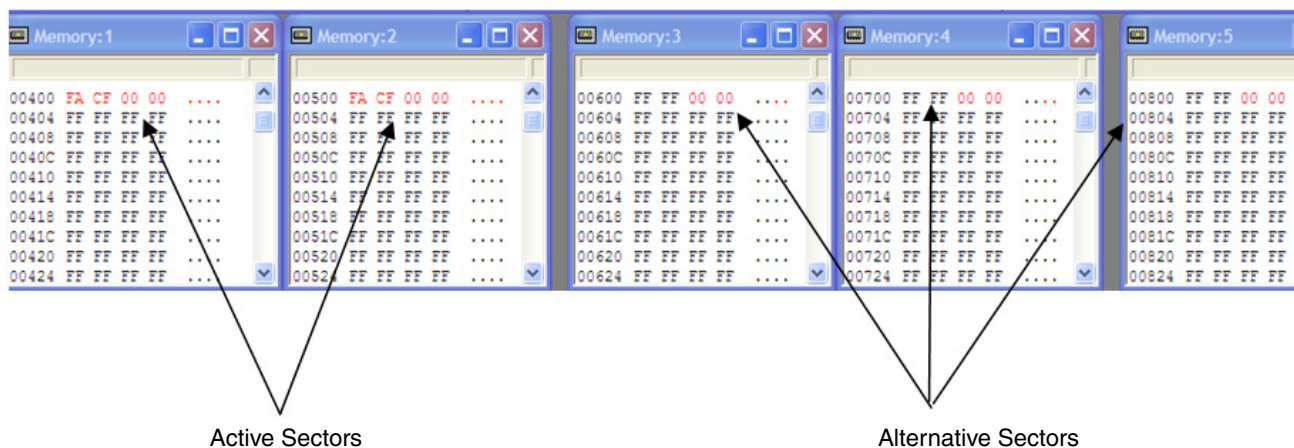
3.3.4 Breakpoint 2 — Initialize the Active and Alternative Sectors

```
asm BGND;
/* User breakpoint 2 - the function below will write FACF0000 to Active sector and FFFF0000 to al
returnCode = FSL_InitEeprom(); /* initialise EEPROM */
/* Hit run to the next software breakpoint, where first record will be written */

if (EED_OK != returnCode)
{
    ErrorTrap();
}

/* Wait until InitEeprom is done */
while(BUSY == EE_Status)
{
    FSL_Main();
}
```

The D-flash has now been erased and will have to be arranged as active and alternative sectors. This software driver requires that at least two alternative sectors are available. This is to deal with any brownout or dead sector situations. The 'FSL_InitEeprom' function initializes the two sectors at location 0x4400 and 0x4500 to 'active' and the remaining sectors 0x4600, 0x4700, and 0x4800 to 'alternative'. On completion, the active sectors are defined by 0xFACF0000 and alternative are defined by 0xFFFF0000.



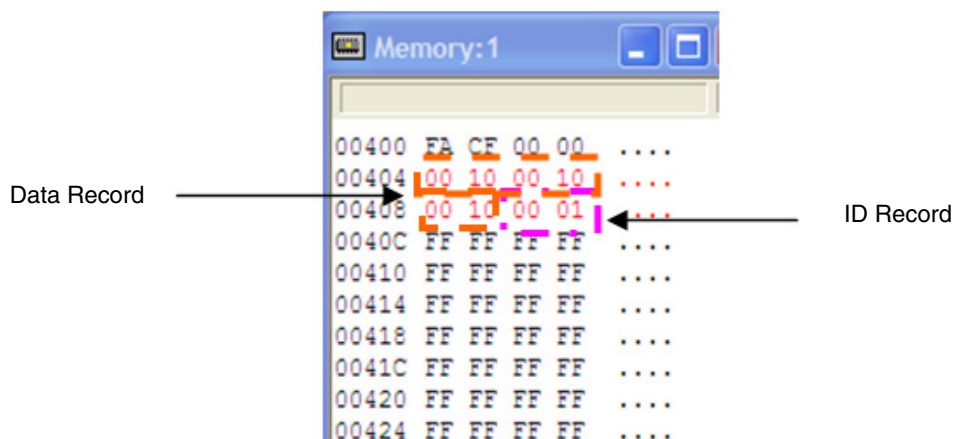
3.3.5 Breakpoint 3 — Write First Data and ID Record

```

asm BGND;
/*software breakpoint 3 - the function below will write the first data record and ID*/
returnCode = FSL_WriteEeprom(DATA_ID_ONE, temp);
if (EED_OK != returnCode)
{
    ErrorTrap();
}
/* Wait until WriteEeprom is done */
while(BUSY == EE_Status)
{
    FSL_Main();
}

```

The data to be written is defined within a header file — 0x10. When executed, this function will write the data 0x10 and will assign this with a record ID of 0x01. The ID size is 2 bytes and the data size has been configured to 6 bytes. The specific EEE user guide explains how to set the data size.



3.3.6 Breakpoint 4 — Completely Write the Active Sectors

```
asm BGND;
/* software breakpoint 4 - the first record has been written and the function below will write t/
/* It will loop here until all the active sectors are filled*/

for(i = DATA_ID_ONE; i <= EED_MAX_RECORD_NUMBER; i++)
{
    for(j = 0; j < (EED_DATA_VALUE_SIZE/EED_MIN_PROG_SIZE); j++)
    {
        temp[j] = i + DATA_VALUE;
    }
    returnCode = FSL_WriteEeprom(i, temp);
    if (EED_OK != returnCode)
    {

```

This writes data records such that the active block is completely filled. The next write after the loop will cause a swap. The active sectors at 0x400 and 0x500 have been filled since the code executes a for loop until it reaches the end of the second active sector.

00400 FA CF 00 00	00500 FA CF 00 00	005DC 00 4A 00 4A
00404 00 10 00 10	00504 00 2F 00 2F ./. /	005E0 00 4A 00 3A
00408 00 10 00 01	00508 00 2F 00 1F ./. .	005E4 00 4B 00 4B
0040C 00 11 00 11	0050C 00 30 00 30 .0.0	005E8 00 4B 00 3B
00410 00 11 00 01	00510 00 30 00 20 .0.	005EC 00 4C 00 4C
00414 00 12 00 12	00514 00 31 00 31 .1.1	005F0 00 4C 00 3C
00418 00 12 00 02	00518 00 31 00 21 .1.!	005F4 00 4D 00 4D
0041C 00 13 00 13	0051C 00 32 00 32 .2.2	005F8 00 4D 00 3D
00420 00 13 00 03	00520 00 32 00 22 .2."	005FC FF FF FF FF
00424 00 14 00 14	00524 00 33 00 33 .3.3	00600 FF FF 00 00 ← Beginning of first alternative sector
00428 00 14 00 04	00528 00 33 00 23 .3.#	00604 FF FF FF FF
0042C 00 15 00 15	0052C 00 34 00 34 .4.4	00608 FF FF FF FF
00430 00 15 00 05	00530 00 34 00 24 .4.#	0060C FF FF FF FF
00434 00 16 00 16	00534 00 35 00 35 .5.5	00610 FF FF FF FF
00438 00 16 00 06	00538 00 35 00 25 .5.%	00614 FF FF FF FF
0043C 00 17 00 17	0053C 00 36 00 36 .6.6	00618 FF FF FF FF
00440 00 17 00 07	00540 00 36 00 26 .6.#	0061C FF FF FF FF
00444 00 18 00 18	00544 00 37 00 37 .7.7	00620 FF FF FF FF
00448 00 18 00 08	00548 00 37 00 27 .7.'	00624 FF FF FF FF
0044C 00 19 00 19	0054C 00 38 00 38 .8.8	00628 FF FF FF FF
00450 00 19 00 09	00550 00 38 00 28 .8.(	0062C FF FF FF FF
00454 00 1A 00 1A	00554 00 39 00 39 .9.9	00630 FF FF FF FF
00458 00 1A 00 0A	00558 00 39 00 29 .9.)	00634 FF FF FF FF
0045C 00 1B 00 1B	0055C 00 3A 00 3A .:.:	00638 FF FF FF FF
00460 00 1B 00 0B	00560 00 3A 00 2A .:.*	0063C FF FF FF FF

3.3.7 Breakpoint 5 — Sector Swap

Now that the active sectors have been completely filled, the next record write will only occur after a new active sector has been created. This is a two-stage process, firstly all the records from the first active sector are copied to the first available alternative sector — in this case, data and ID records from 0x400-0x499 are copied to 0x600. Secondly, the sector at 0x400 is erased and becomes a new alternative sector. Notice on the new alternative sector, it begins with 0xFFFF0001.

00400	FF FF 00 01		00500	FA CF 00 00		005DC	00 4A 00 4A	.J.J	006F0	00 2E 00 1E
00404	FF FF FF FF		00504	00 2F 00 2F	././	005E0	00 4A 00 3A	.J.:	006F4	CE CE CE CE
00408	FF FF FF FF		00508	00 2F 00 1F	./..	005E4	00 4B 00 4B	.K.K	006F8	CE CE 00 10
0040C	FF FF FF FF		0050C	00 30 00 30	.0.0	005E8	00 4B 00 3B	.K.;	006FC	FF FF FF FF
00410	FF FF FF FF		00510	00 30 00 20	.0.	005EC	00 4C 00 4C	.L.L	00700	FF FF 00 00
00414	FF FF FF FF		00514	00 31 00 31	.1.1	005F0	00 4C 00 3C	.L.<	00704	FF FF FF FF
00418	FF FF FF FF		00518	00 31 00 21	.1.!	005F4	00 4D 00 4D	.M.M	00708	FF FF FF FF
0041C	FF FF FF FF		0051C	00 32 00 32	.2.2	005F8	00 4D 00 3D	.M.=	0070C	FF FF FF FF
00420	FF FF FF FF		00520	00 32 00 22	.2."	005FC	FF FF FF FF		00710	FF FF FF FF
00424	FF FF FF FF		00524	00 33 00 33	.3.3	00600	FA CF 00 00		00714	FF FF FF FF
00428	FF FF FF FF		00528	00 33 00 23	.3.#	00604	00 11 00 11		00718	FF FF FF FF
0042C	FF FF FF FF		0052C	00 34 00 34	.4.4	00608	00 11 00 01		0071C	FF FF FF FF
00430	FF FF FF FF		00530	00 34 00 24	.4.4	0060C	00 12 00 12		00720	FF FF FF FF
00434	FF FF FF FF		00534	00 35 00 35	.5.5	00610	00 12 00 02		00724	FF FF FF FF
00438	FF FF FF FF		00538	00 35 00 25	.5.%	00614	00 13 00 13		00728	FF FF FF FF
0043C	FF FF FF FF		0053C	00 36 00 36	.6.6	00618	00 13 00 03		0072C	FF FF FF FF
00440	FF FF FF FF		00540	00 36 00 26	.6.&	0061C	00 14 00 14		00730	FF FF FF FF
00444	FF FF FF FF		00544	00 37 00 37	.7.7	00620	00 14 00 04		00734	FF FF FF FF
00448	FF FF FF FF		00548	00 37 00 27	.7.'	00624	00 15 00 15		00738	FF FF FF FF
0044C	FF FF FF FF		0054C	00 38 00 38	.8.8	00628	00 15 00 05		0073C	FF FF FF FF
00450	FF FF FF FF		00550	00 38 00 28	.8.(	0062C	00 16 00 16		00740	FF FF FF FF
00454	FF FF FF FF		00554	00 39 00 39	.9.9	00630	00 16 00 06		00744	FF FF FF FF
00458	FF FF FF FF		00558	00 39 00 29	.9.)	00634	00 17 00 17		00748	FF FF FF FF
0045C	FF FF FF FF		0055C	00 3A 00 3A	..:.	00638	00 17 00 07		0074C	FF FF FF FF
00460	FF FF FF FF		00560	00 3A 00 2A	..*.	0063C	00 18 00 18			

The process described whereby the sectors are being filled, copied, and erased will continue through an application, such as an odometer for example. When power to the application is lost, the data is stored in the D-flash and is easily read.

3.3.8 Breakpoint 6 — Reading EEE and Erase

```
asm_BGND;
/* Software breakpoint 6 - Active sector swapped. The code below will read and then re-program the
/* This reads the data with Data ID 0x01. The record is read from the cache table if EED_CACHETABLE.

returnCode=FSL_ReadEeprom(DATA_ID_ONE, &readAddr);
if (EED_OK != returnCode)
{
    ErrorTrap();
}
else
{
    for(i = 0; i < EED_DATA_VALUE_SIZE ; i+=EED_MIN_PROG_SIZE)
    {
        /* ... */
    }
}
```

There are read functions after breakpoint 6 that are responsible for reading the EEE. This will read the record with a specified data and address.

This demonstration program will complete by erasing the D-flash sectors and will end in the while loop.

00400	FF FF FF FF		00500	FF FF FF FF		005DC	FF FF FF FF		006F0	FF FF FF FF	
00404	FF FF FF FF		00504	FF FF FF FF		005E0	FF FF FF FF		006F4	FF FF FF FF	
00408	FF FF FF FF		00508	FF FF FF FF		005E4	FF FF FF FF		006F8	FF FF FF FF	
0040C	FF FF FF FF		0050C	FF FF FF FF		005E8	FF FF FF FF		006FC	FF FF FF FF	
00410	FF FF FF FF		00510	FF FF FF FF		005EC	FF FF FF FF		00700	FF FF FF FF	
00414	FF FF FF FF		00514	FF FF FF FF		005F0	FF FF FF FF		00704	FF FF FF FF	
00418	FF FF FF FF		00518	FF FF FF FF		005F4	FF FF FF FF		00708	FF FF FF FF	
0041C	FF FF FF FF		0051C	FF FF FF FF		005F8	FF FF FF FF		0070C	FF FF FF FF	
00420	FF FF FF FF		00520	FF FF FF FF		005FC	FF FF FF FF		00710	FF FF FF FF	
00424	FF FF FF FF		00524	FF FF FF FF		00600	FF FF FF FF		00714	FF FF FF FF	
00428	FF FF FF FF		00528	FF FF FF FF		00604	FF FF FF FF		00718	FF FF FF FF	
0042C	FF FF FF FF		0052C	FF FF FF FF		00608	FF FF FF FF		0071C	FF FF FF FF	
00430	FF FF FF FF		00530	FF FF FF FF		0060C	FF FF FF FF		00720	FF FF FF FF	
00434	FF FF FF FF		00534	FF FF FF FF		00610	FF FF FF FF		00724	FF FF FF FF	
00438	FF FF FF FF		00538	FF FF FF FF		00614	FF FF FF FF		00728	FF FF FF FF	
0043C	FF FF FF FF		0053C	FF FF FF FF		00618	FF FF FF FF		0072C	FF FF FF FF	
00440	FF FF FF FF		00540	FF FF FF FF		0061C	FF FF FF FF		00730	FF FF FF FF	
00444	FF FF FF FF		00544	FF FF FF FF		00620	FF FF FF FF		00734	FF FF FF FF	
00448	FF FF FF FF		00548	FF FF FF FF		00624	FF FF FF FF		00738	FF FF FF FF	
0044C	FF FF FF FF		0054C	FF FF FF FF		00628	FF FF FF FF		0073C	FF FF FF FF	
00450	FF FF FF FF		00550	FF FF FF FF		0062C	FF FF FF FF		00740	FF FF FF FF	
00454	FF FF FF FF		00554	FF FF FF FF		00630	FF FF FF FF		00744	FF FF FF FF	
00458	FF FF FF FF		00558	FF FF FF FF		00634	FF FF FF FF		00748	FF FF FF FF	
0045C	FF FF FF FF		0055C	FF FF FF FF		00638	FF FF FF FF		0074C	FF FF FF FF	
00460	FF FF FF FF		00560	FF FF FF FF		0063C	FF FF FF FF				

3.3.9 Summary

The demonstration software has shown how to initialize the D-flash for EEE operation by producing active and alternative sectors via the 'FSL_InitEeprom' function. The functions for writing, 'FSL_WriteEeprom' and reading, 'FSL_ReadEeprom' are required to write/read the appropriate data and accompanying ID records and hence emulate EEPROM. In an application it will be the two latter functions that will be relied upon. Moreover, the software is capable of dealing with brownout events as well as dead sectors. For developing applications with this code, it is advisable to read the EEE driver user's guide included Emulated EEPROM software pack, available from www.freescale.com.

3.4 MSCAN Module

This lab example uses the MSCAN module in loopback mode to transmit and receive a byte of data using standard length identifiers and four 16-bit filters. The status of the four switches, SW1 to SW4, is read and transmitted by the MSCAN module. When the MSCAN module receives its own transmission, the data in the message is read and displayed on the four LEDs.

When the MSCAN module is operated in loopback mode, no CAN signals are transmitted externally. Both the Tx and Rx pins are held high.

3.4.1 Setup

The following steps should be completed before running the lab example:

1. Start CodeWarrior™ by selecting it in the Microsoft Windows Start menu.
2. From the CodeWarrior main menu, select **File > Open** and select the **S12HY MSCAN Demo.mcp** file.
3. Click on the **Open** button. The project window will open.
4. The C code of this demonstration is contained within the **main.c** file. Double-click on the file to open it.

5. From the main menu, select **Project > Debug**. This will compile the source code, generate an executable file, and download it to the demo board.
6. A new debugger environment will open. After the download to the demo board is completed, close the debugger environment.

3.4.2 Instructions

Follow these instructions to run the lab example:

1. Hit RESET. The MSCAN demo software will begin execution.
2. Press various combinations of the SW1, SW2, SW3, and SW4 switches. The LEDs should match their configuration.

3.4.3 Summary

The MSCAN module is a serial data bus communication controller implementing the CAN 2.0A/B protocol as defined in the Bosch specification dated September 1991. It is not limited to automotive applications and is suited to wide variety of uses that require reliable communications.

For further information on the MSCAN module, refer to the following documentation that is available at www.freescale.com.

- Application note titled *Using MSCAN on the HCS12 Family* (document AN3034)
- *MC9S12HY64 Reference Manual* (document MC9S12HY64RM)

3.5 PWM Module

This lab provides an example of how to setup and use the PWM module to create a 50% duty cycle output with different polarity and alignment settings. This behavior is best illustrated if all of the PWM signals can be displayed simultaneously on a four channel oscilloscope.

3.5.1 Setup

The following steps should be completed before running the lab example:

1. Start CodeWarrior™ by selecting it in the Windows Start menu.
2. From the CodeWarrior main menu, select **File > Open** and select the **S12HY PWM Demo.mcp** file.
3. Click on the **Open** button. The project window will open.
4. The C code of this demonstration is contained within the **main.c** file. Double-click on the file to open it.
5. From the main menu, select **Project > Debug**. This will compile the source code, generate an executable file, and download it to the demo board.
6. A new debugger environment will open. After the download to the demo board is completed, close the debugger environment.

3.5.2 Instructions

In order to give access to PWM signals on the J1 header, the software re-routes PWM[3:0] from PP[3:0] to PS[7:4].

Follow these instructions to run the lab example:

1. Hit RESET. The code should run and re-route the PWM channels. The PWM module should output 50% duty cycle signals on ports PS7–PS4.
2. Try probing all four signals simultaneously if possible. This allows the difference in settings such as center alignment and polarity to be more apparent.

3.5.3 Summary

The PWM is a common module on many microcontrollers. It often finds use in applications that have a need to vary frequency or intensity, such as with lighting.

For further information on the PWM module, refer to the following documentation that is available at www.freescale.com.

- *MC9S12HY64 Reference Manual* (document MC9S12HY64RM)

3.6 S12HY Low Power Modes

In addition to the default Run mode, the MC9S12HY has three low power modes, Wait, Pseudo Stop, and Stop.

Wait mode is similar to Run mode except that CPU execution is halted and it is possible to selectively disable some modules so that only necessary modules are clocked.

For lower power consumption, Pseudo Stop mode halts the bus clock, but the external oscillator continues to run.

Stop Mode disables the external oscillator for the lowest power consumption.

This lab example shows how to enter each mode and the differences between them.

The table below summarizes the signals present in each mode.

Mode	Bus Clock	External Oscillator
Run	Y	Y
Wait	Y	Y
Pseudo Stop	N	Y
Stop	N	N

The changes in the MCU operating mode can be observed via the LEDs and by monitoring the ECLK signal (Bus clock) and EXTAL signal (Crystal) on an oscilloscope.

Care should be taken to probe ECLK and EXTAL separately to avoid adding extra noise to the signals.

3.6.1 Setup

The following steps should be completed before running the lab example:

1. Start CodeWarrior™ by selecting it in the Microsoft Windows Start menu.
2. From the CodeWarrior main menu, select **File > Open** and select the **S12HY Low Power Modes.mcp** file.
3. Click on the **Open** button. The project window will open.
4. The C code of this demonstration is contained within the **main.c** file. Double-click on the file to open it.
5. From the main menu, select **Project > Debug**. This will compile the source code, generate an executable file, and download it to the demo board.
6. A new debugger environment will open. After the download to the demo board is completed, close the debugger environment.
7. The bus clock speed is represented on pin 59 PH2/ECLK. The ECLK signal is equivalent to the MCU bus speed and can be monitored by attaching an oscilloscope probe to pin 23 of the DS1 header.
8. The oscillator can be monitored by attaching a scope probe to the EXTAL side of the Y1 crystal.

3.6.2 Instructions

Follow these instructions to run the lab example:

1. Hit RESET. The MCU is now operating in Run mode. The LEDs will flash indefinitely indicating the MCU is in Run mode.
2. Monitor the ECLK signal on the oscilloscope. ECLK represents the bus clock. A 32 MHz square wave should be observed.
3. Monitor the EXTAL signal on the oscilloscope. EXTAL indicates that the crystal oscillator is running. An 8 MHz sine wave should be observed.
4. Hit RESET whilst pressing down SW1. LED1 will flash twenty times indicating Run mode and then the MCU will enter into Wait mode. Pressing SW4 causes the MCU to exit Wait mode back into Run mode. LED1 will flash twenty times before the MCU returns into Wait mode again.
5. Monitor the ECLK signal on the oscilloscope. The 32 MHz square wave representing the bus clock will be present in both Run and Wait modes.
6. Monitor the EXTAL signal on the oscilloscope. In both Run and Wait modes, an 8 MHz sine wave should be observed indicating that the external oscillator continues to operate in Wait mode.
7. Hit RESET whilst pressing down SW2. LED1 will flash twenty times indicating Run mode and then the MCU will enter into Pseudo Stop mode. Pressing SW4 causes the MCU to exit Pseudo Stop mode back into Run mode. LED1 will flash twenty times before the MCU returns into Pseudo Stop mode again.
8. Monitor the ECLK signal on the oscilloscope. The 32 MHz square wave representing the bus clock is only present in Run mode. In Pseudo Stop mode, the bus clock is stopped to save power.

9. Monitor the EXTAL signal on the oscilloscope. In both Run and Pseudo Stop modes, an 8 MHz sine wave should be observed indicating that the external oscillator continues to operate in Pseudo Stop mode.
10. Hit RESET whilst pressing down SW3. LED1 will flash twenty times indicating Run mode and then the MCU will enter into Stop mode. Pressing SW4 causes the MCU to exit Stop mode back into Run mode. LED1 will flash twenty times before the MCU enters returns into Stop mode again.
11. Monitor the ECLK signal on the oscilloscope. The 32 MHz square wave representing the bus clock is only present in Run mode. In Stop mode, the bus clock is stopped to save power.
12. Monitor the EXTAL signal on the oscilloscope. The 8 MHz sine wave is only present in Run mode. In Stop mode, the external oscillator is stopped to save power.

3.6.3 Summary

The MC9S12HY Family can be configured in a variety of ways to achieve low power consumption. The three low power modes offer different solutions for user applications.

For further information on low power modes, refer to the following documentation available at www.freescale.com.

- Application note titled *Low Power Management Using HCS12 and SBC Devices* (document AN2461)
- *MC9S12HY64 Reference Manual* (document MC9S12HY64RM)

3.7 MMC Program Flash Paging Window

The MC9S12HY64 has Flash memory of 64 KB. Whilst this amount of memory can be addressed by the 16-bit MC9S12HY64 MCU, there is insufficient local address space to accommodate all of the Flash memory, the RAM memory, and the register space. Instead, a paging system which maps 16 KB blocks of memory into the local memory map from address 0x8000 to 0xBFFF is used.

This lab example shows how to use the paging capability of the MMC module to access global memory addresses within the local memory map.

3.7.1 Setup

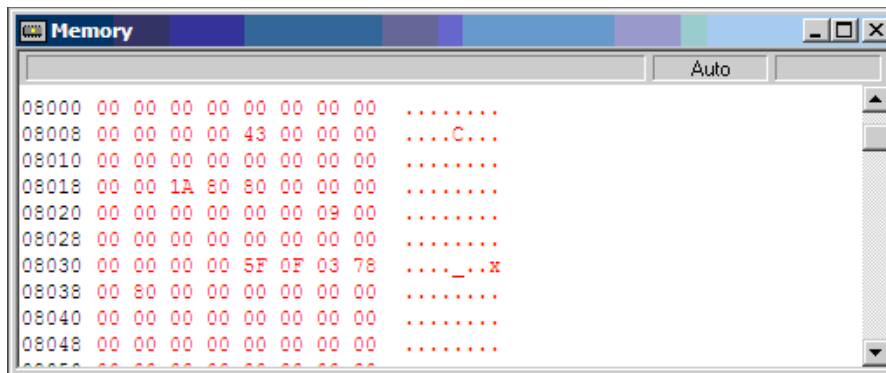
The following steps should be completed before running the lab example:

1. Start CodeWarrior™ by selecting it in the Microsoft Windows Start menu.
2. From the CodeWarrior main menu, select **File > Open** and select the **S12HY MMC Demo.mcp** file.
3. Click on the **Open** button. The project window will open.
4. The C code of this demonstration is contained within the **main.c** file. Double-click on the file to open it.
5. From the main menu, select **Project > Debug**. This will compile the source code, generate an executable file, and download it to the demo board.
6. A new debugger environment will open. Do not close the debugger environment.

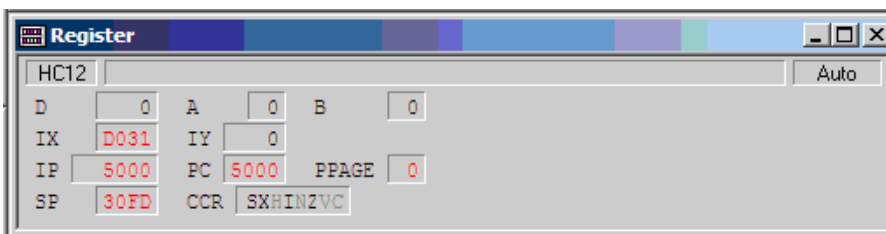
3.7.2 Instructions

Follow these instructions to run the lab example:

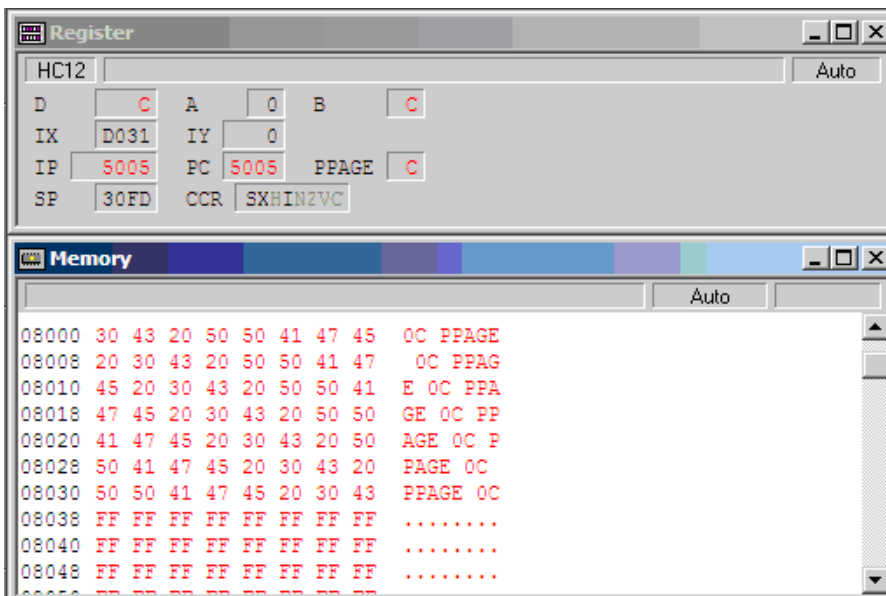
1. The **Memory** window in the debugger environment is configured to show the first few locations of the P-flash window at address 0x8000.



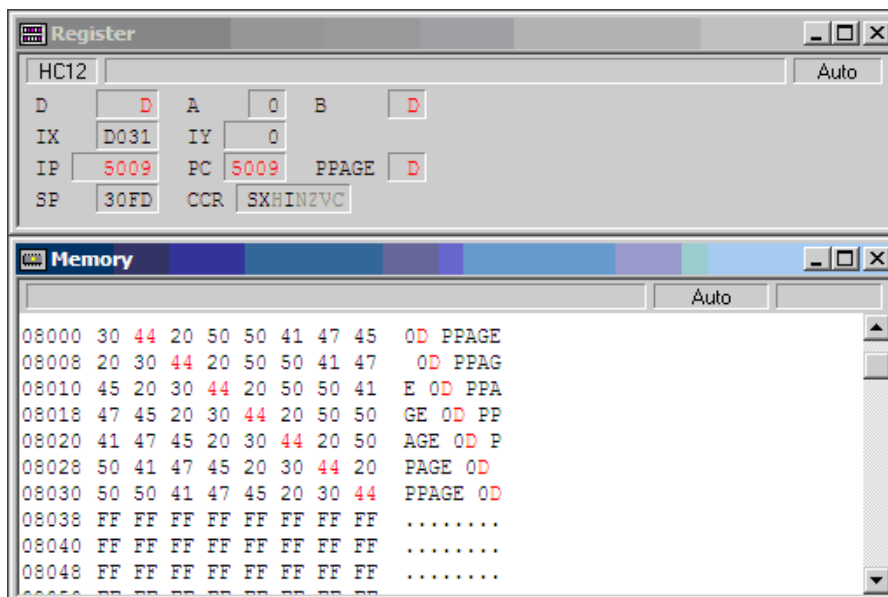
2. The **Register** window in the debugger environment shows the setting of the Program Page Index Register (PPAGE).



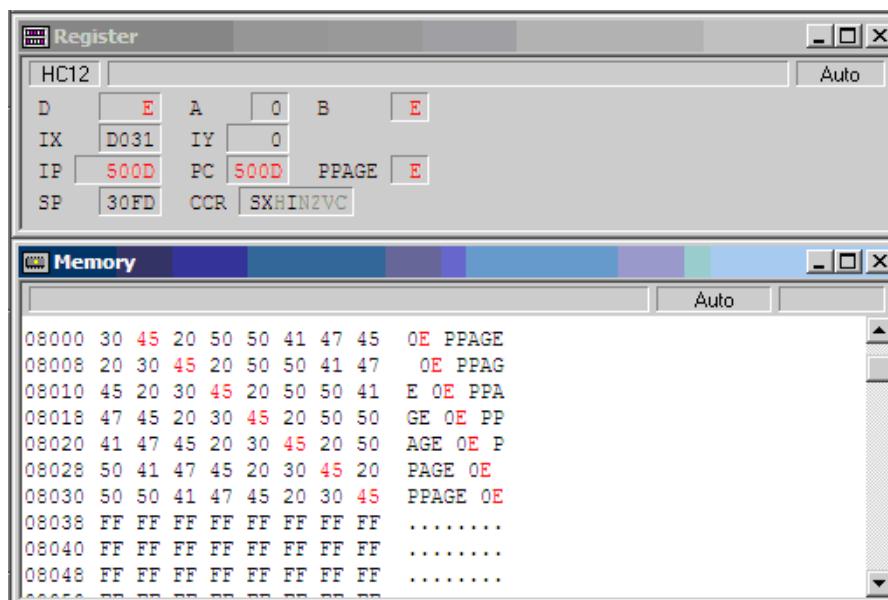
3. Start the software by clicking on the Run  button.
4. When the software hits the first breakpoint, examine the contents of the **Memory** and **Register** windows. The PPAGE register is set to 0C and the P-flash window shown in the **Memory** window displays the contents of PPAGE 0C.



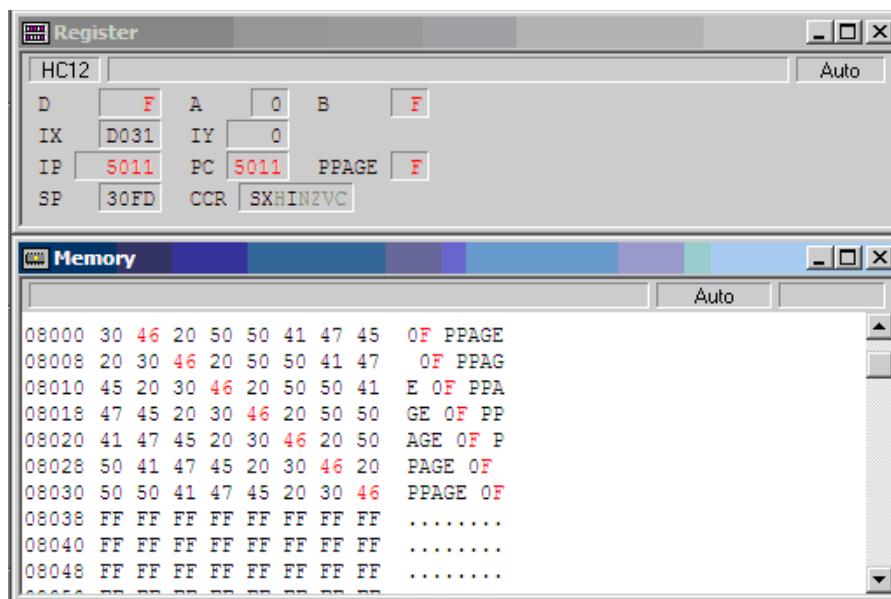
5. Hit the **Run** button and observe the **Memory** and **Register** windows at the next breakpoint.
6. Now the PPAGE register is set to 0D and the P-flash window shown in the **Memory** window displays the contents of PPAGE 0D.



7. Hit the **Run** button and observe the **Memory** and "Register" windows at the next breakpoint.
8. Now the PPAGE register is set to 0E and the P-Flash Window shown in the "Memory" window displays the contents of PPAGE 0E.



9. Hit the **Run** button and observe the **Memory** and **Register** windows at the next breakpoint.
10. Now the PPAGE register is set to 0F and the P-flash window shown in the **Memory** window displays the contents of PPAGE 0F.



11. Hit the **Run** button and the software will loop back to return the PPAGE register to 0C. The contents of PPAGE 0C can be seen in the **Memory** window.

3.7.3 Summary

The MMC module can be used to expand the accessible amount of memory of the MC9S12HY64 MCU by paging the expanded global memory into a window in local memory.

For further information on the MMC module, refer to the following documentation available at www.freescale.com.

- *MC9S12HY64 Reference Manual* (document MC9S12HY64RM)

3.8 ADC Module

This lab example shows how to use the ADC module to perform single conversions, continuous conversions, and automatic compare. The ADC conversion results are output on a terminal window via the RS-232 port.

3.8.1 Setup

The following steps should be completed before running the lab example:

1. Start CodeWarrior™ by selecting it in the Microsoft Windows Start menu.
2. From the CodeWarrior main menu, select **File > Open** and select the **S12HY ADC Demo.mcp** file.
3. Click on the **Open** button. The project window will open.
4. The C code of this demonstration is contained within the **main.c** file. Double-click on the file to open it.
5. From the main menu, select **Project > Debug**. This will compile the source code, generate an executable file, and download it to the demo board.

6. A new debugger environment will open. After the download to the demo board is completed, close the debugger environment.
7. The ADC conversion result is sent to the RS-232 port (baud rate = 9600, data bits = 8, parity = N, stop bits = 1, Flow control = Hardware). Open a terminal window on the PC with this configuration.

3.8.2 Instructions

Follow these instructions to run the lab example:

1. Hit RESET. The ADC will perform a single 12-Bit conversion on PAD00. To perform another conversion press SW4.
2. Vary the conversion result by turning potentiometer RV1 on PAD00 and observe the changes in the terminal window.
3. Hit RESET whilst pressing down SW1. The ADC will perform continuous 8-bit conversions on PAD00.
4. Vary the conversion result by turning potentiometer RV1 on PAD00 and observe the changes in the terminal window.
5. Hit RESET whilst pressing down SW2. The ADC will perform continuous 12-bit conversions on PAD00 and compare the result to see if it is higher than 0x07FF. Whilst the comparison is true, LED1 on the demo board will flash.
6. Vary the conversion result by turning potentiometer P501 on PAD00 and observe the result in the terminal window. Notice how LED1 only flashes when the result is greater than 0x07FF.

3.8.3 Summary

The ADC module is highly autonomous with an array of flexible conversion sequences and resolution. It can be configured to select which analogue source to start conversion on, how many conversions to perform, and whether these should be on the same or multiple input channels. An automatic compare can be used to liken the conversion result against a programmable value for higher than or less than/equal to matching. Any conversion sequence can be repeated continuously without additional MCU overhead.

For further information on the ADC module, refer to the following documentation available at www.freescale.com.

- Application note titled *An Overview of the HCS12 ATD Module* (document AN2428)
- *MC9S12HY64 Reference Manual* (document MC9S12HY64RM)

3.9 Timer Module

This lab example shows how to use the Timer module to perform output compare and input capture. In case an oscilloscope is unavailable, the LEDs associated with Port R on the DEMO9S12HY64 board are used to indicate port toggles due to an output compare match, or a successful input capture.

3.9.1 Setup

The following steps should be completed before running the lab example:

1. Ensure that the LED4 and POT jumpers on JP14 have been removed.
2. Connect the potentiometer output (pin 18 on JP14) to Port R3 (pin 7 on JP14). This will allow the potentiometer RV1 to provide stimulus to the input capture function on IOC1_7.
3. Start CodeWarrior™ by selecting it in the Microsoft Windows Start menu.
4. From the CodeWarrior main menu, select **File > Open** and select the **S12HY Timer Demo.mcp** file.
5. Click on the **Open** button. The project window will open.
6. The C code of this demonstration is contained within the **main.c** file. Double-click on the file to open it.
7. From the main menu, select **Project > Debug**. This will compile the source code, generate an executable file, and download it to the demo board.
8. A new debugger environment will open. After the download to the demo board is completed, close the debugger environment.

3.9.2 Instructions

In order to drive LEDs directly from Timer output channels, the software re-routes the Timer 0 channels IOC0_7 and IOC0_6 from Port T7:6 to Port R1:0. In addition, to allow a connection to an input capture channel, the software also re-routes Timer 1 channel IOC1_7 from Port T3 to Port R3. The re-routing also gives the user access to the signals on the J1 header

Follow these instructions to run the lab example:

1. Hit RESET. The code should run and re-route the Timer channels.
2. Timer 0 will perform output compares on channels IOC0_7 (Port R1 re-routed from Port T7) and IOC0_6 (Port R0 re-routed from Port T6). When a compare match occurs on IOC0_7, Port R1 will toggle. When a match compare occurs on IOC0_6, Port R0 will toggle.
3. Timer 1 will perform input capture on both rising and falling edges on channel IOC1_7 (Port R3 re-routed from Port T3).
4. Use an oscilloscope to view the toggling Timer 0 channels IOC0_7 and IOC0_6 on Port pins R1 and R0 (J1 header pins 11 and 9). In case an oscilloscope is not available, the LEDs associated with Port R1 (LED2) and Port R0 (LED1) toggle in sync with the Timer channels.
5. Use the potentiometer RV1 to create rail to rail rising and falling edges on Timer 1 channel IOC1_7.
6. To ensure the input capture is detecting edge transitions, observe LED3 toggles with each rising or falling edge.

3.9.3 Summary

The timer is a very useful module in that it provides a trigger for events to occur at a specific time, or captures when events have occurred. It is very important in the scheduling of repetitive actions and contains a variety of special functions, such as pulse accumulation.

For further information on the timer module, refer to the following documentation available at www.freescale.com.

- *MC9S12HY64 Reference Manual* (document MC9S12HY64RM)

3.10 SCI Communications

This lab example shows how to configure the SCI module to transmit and receive data using different baud rates.

3.10.1 Setup

The following steps should be followed before running the lab example:

1. Ensure that both the BCOM_EN jumpers on JP11 have been removed.
2. Start CodeWarrior™ by selecting it in the Microsoft Windows Start menu.
3. From the CodeWarrior main menu, select **File > Open** and select the **S12HY SCI Demo.mcp** file.
4. Click on the **Open** button. The project window will open.

3.10.2 Instructions

Follow these instructions to run the lab example:

1. The C code of this demonstration is contained within the **main.c** file. Double-click on the file to open it.
2. Configure the variable "Baud_Rate" to 9600 and make sure all other options are disabled.

```
void main(void) {  
    char RegisterCase;  
    unsigned int BaudRatePrescaler, x,y;  
  
    unsigned long Baud_Rate = 9600;  
    // unsigned long Baud_Rate = 19200;  
    // unsigned long Baud_Rate = 38400;  
    // unsigned long Baud_Rate = 57600;
```

3. From the main menu, select **Project > Debug**. This will compile the source code, generate an executable file, and download it to the demo board.
4. A new debugger environment will open.
5. The software uses the RS-232 port to interact with the user. Open a terminal window (baud rate = 9600, data bits = 8, parity = N, stop bits = 1, Flow control = Hardware) to see the RS-232 port data.
6. Hit F5. The code will begin execution, configuring the SCI to the selected baud rate. It's status can be confirmed on the terminal window.
7. The SCI register configurations can be confirmed by selecting an option displayed on the terminal window. Select some options and observe the SCI register configurations.

8. Repeat steps 2 to 9 for baud rates of 19200, 38400 and 57600. Alternatively, modify the definition of variable "Baud_Rate" for a user configured baud rate.

3.10.3 Summary

The SCI module can be used to communicate with peripheral devices or other MCUs.

For further information on the SCI module please refer to the following documentation which is available at www.freescale.com.

- Application note titled *Serial Communication Interface as UART on HCS12 MCUs* (document AN2883)
- *MC9S12HY64 Reference Manual* (document MC9S12HY64RM)

3.11 SPI Communications

This lab example shows how to set up and use the SPI module in master mode to transmit an incrementing byte of data.

As there is only one SPI module available on the DEMO9S12HY64 board, this example is limited to transmitting data only. When an SPI master transmits data to an SPI slave, data is usually received simultaneously, synchronized by a serial clock.

3.11.1 Setup

An oscilloscope and three scope probes are required for this demo. The following steps should be completed before running the lab example.

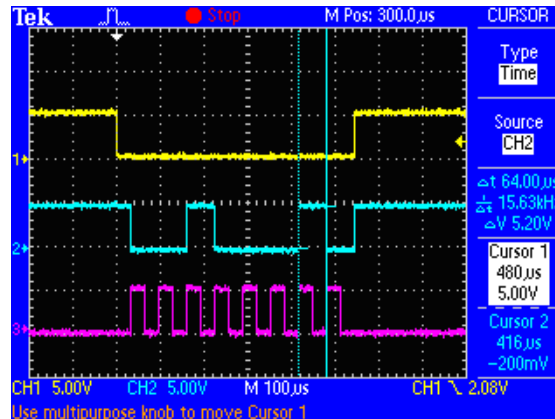
1. Start CodeWarrior™ by selecting it in the Microsoft Windows Start menu.
2. From the CodeWarrior main menu, select **File > Open** and select the **S12HY SPI Demo.mcp** file.
3. Click on the **Open** button. The project window will open.
4. The C code of this demonstration is contained within the **main.c** file. Double-click on the file to open it.
5. From the main menu, select **Project > Debug**. This will compile the source code, generate an executable file, and download it to the demo board.
6. A new debugger environment will open. After the download to the demo board is completed, close the debugger environment.
7. Attach scope probes to signals PS4, PS6, and PS7 on header J1.
8. Configure the oscilloscope to trigger on the falling edge of PS7.

3.11.2 Instructions

Follow these instructions to run the lab example:

1. Hit RESET. The code will begin execution, configuring the SPI to transmit an incrementing byte of data at a baud rate of 15.625 kbits/s.

2. Monitor the SPI transmission on the oscilloscope to see the relationship between Slave Select (PS7), data transmitted on MOSI (PS4), and the Serial Clock (PS6) signals.



3.11.3 Summary

The SPI module can be used to allow duplex synchronous serial communication between peripheral devices and the MCU.

For further information on the SPI module, refer to the following documentation available at www.freescale.com.

- *MC9S12HY64 Reference Manual* (document MC9S12HY64RM)

3.12 Motor Control Module

This demonstration code has been constructed to explain how to set up the motor control module and operate an external stepper motor. In order to operate this demo, the user will require connecting an external motor to PTU0-3.

3.12.1 Setup

1. Connect a stepper motor to Motor 0 pins, PTU0-3 -J3 on the demo board.
2. Start CodeWarrior™ by selecting it in the Microsoft Windows Start menu.
3. From the CodeWarrior main menu, select **File > Open** and select the **S12HY_MC_DEMO.mcp** file.
4. Click on the **Open** button. The project window will open.
5. The C code of this demonstration is contained within the **main.c** file. Double-click on the file to open it.
6. From the main menu, select **Project > Debug**. This will compile the source code, generate an executable file, and download it to the demo board.

The motor control module is set up via the 'MC_init' function where the motor control registers are configured. The module can be set to operate in various modes; in this case it is operated in dual full H-bridge, which is ideal for controlling stepper motors. The physical set-up of the motor requires 4

connections, whereby each PWM channels has two connections. See S12HY reference manual, section 17.4.1.1.1, “Dual Full H-Bridge Mode.”

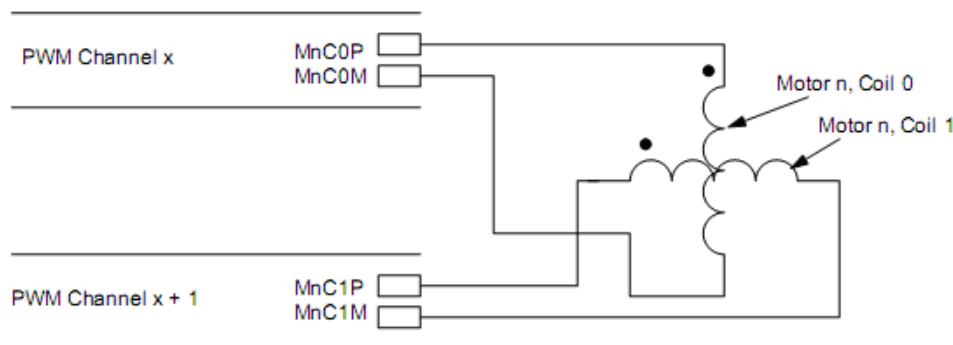


Figure 3. Dual full H-bridge configuration

3.12.2 Instructions

This is a self-contained example in which the user requires no intervention. On running the program, the motor will initialize by returning to zero (RTZ) position. The motor is simply controlled by adjusted the potentiometer RV1 on the board.

3.12.3 Summary

This demonstration has shown that it is possible to control the movement of a stepper motor and this basic example can be applied to many types of applications.

3.13 LCD Module

The demonstration code incorporates an example of an odometer display and trip meters, which increments in real time. There are also other common items displayed and updated on the LCD to emulate a dashboard unit display. This document serves to describe the software and explain the operation of the LCD module.

3.13.1 Setup

1. Start CodeWarrior™ by selecting it in the Microsoft Windows Start menu.
2. From the CodeWarrior main menu, select **File > Open** and select the **S12HY_LCD_DEMO.mcp** file.
3. Click on the **Open** button. The project window will open.
4. The C code of this demonstration is contained within the **main.c** file. Double-click on the file to open it.
5. From the main menu, select **Project > Debug**. This will compile the source code, generate an executable file and download it to the demo board.

In order to begin using the LCD, the pins which operate the front-planes and backplanes must be configured. This is made easy for the user because when the LCD module is enabled, performed in the

LCD_Init function, the 44 pins will output an LCD driver waveform based on the DUTY and BIAS settings in LCDCCR0.

```
void LCD_init()
{
    CONFIG_CLKSOURCE();           //Configure clock source
    SET_LCDCR1_REG();             //Configure operation in stop/wait
    SET_LCD_FRAME_FREQU();        //Configure frame frequency
    SET_DUTY();
    SET_BIAS();
    ENABLE_FP();                   //Enable Frontplanes
    ENABLE_LCD(ON);                //Enable LCD
}
```

Figure 4. LCD initialization functions

The frequency at which the LCD glass should be operated at will be given by the glass manufacturer and in the case of this demo it is 61 Hz. This is obtained by setting the frame frequency via the LCD clock prescaler bits. Table 16-8 of the S12HY/HA reference manual provides some information of LCD clock vs. frame frequency.

The LCD module has a dedicated 20 bytes of RAM at 0x208 which contains the data that is displayed on the 160 segment LCD. This LCD RAM interfaces with the internal address and data buses of the MCU. During any type of power cycle, the contents of the RAM can be indeterminate, therefore it is recommended to set the 20 bytes of RAM to a known state prior to exercising it.

3.13.2 Instructions

This is a self-contained example, in which the user requires no intervention. On program start-up, the LCD displays 'S12HY' and will proceed into a never-ending for loop, which is responsible for the updating and animation of the LCD display.

A port interrupt has also been used on analog ports that are connected to the switches. On pressing SW1 on the hardware, this allows the user to switch the LCD numerical display between ODO, TRIPA, and TRIPB.

3.13.3 Summary

This demonstration has shown that controlling the LCD is a matter of updating the dedicated RAM and manipulating the backplane and frontplane pins. This demonstration does not save the information during power down, so the user will find the odometer and trip information have been reset. A further exercise to this example would be to use the emulated EEPROM driver, available freely from Freescale, to enable recovery of this information, and by continuously reading and writing the data to the device's D-flash.

4 Conclusion

The S12HY Family of MCUs offers the enhanced features of 16-bit performance at the value of 8-bit MCUs. The S12HY family bridges the gap between 8- and 16-bit MCUs and serves as an entry point into Freescale's 16-bit family offerings, giving customers the flexibility to enhance or cost reduce their applications.

References

The S12HY family is ideal for a wide range of central body control applications, such as low-end instrument clusters, window lifts, seat controllers, sunroofs, door modules, low-end Anti-lock Brake Systems (ABS), Electronic Power Steering (EPS), and watchdog control.

A zip file, AN4021SW.zip, containing the complete CodeWarrior projects for the lab examples accompanies this application note. The file can be downloaded from www.freescale.com.

5 References

The following material is available at www.freescale.com.

Software Development Tools

- CodeWarrior for HCS12(X) Microcontrollers

Application Notes

- Application note titled *Stepper Stall Detect Software for S12HY* (document AN4024)
- Application note titled *XGATE Library: TN/STN LCD Driver* (document AN3219)
- Application note titled *Comparison of the S12XS CRG Module with S12P CPMU Module* (document AN3622)
- Application note titled *Using MSCAN on the HCS12 Family* (document AN3034)
- Application note titled *PWM Generation Using HCS12 Timer Channels* (document AN2612)
- Application note titled *An Overview of the HCS12 ATD Module* (document AN2428)
- Application note titled *Serial Communication Interface as UART on HCS12 MCUs* (document AN2883)
- Application note titled *Low Power Management Using HCS12 and SBC Devices* (document AN2461)

Reference Manual

- *MC9S12HY64 Reference Manual* (document MC9S12HY64RM)
- *Automotive Cluster Demo Guide* (document S12HY64ACDUG)

Useful Information

- [Tips for driving LCDs](#)
- [S12HY64 Video Demonstration](#)
- [Emulated EEPROM software driver](#)

THIS PAGE IS INTENTIONALLY BLANK

How to Reach Us:

Home Page:

www.freescale.com

Web Support:

<http://www.freescale.com/support>

USA/Europe or Locations Not Listed:

Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

Europe, Middle East, and Africa:

Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
www.freescale.com/support

Japan:

Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:

Freescale Semiconductor China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:

Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. "Typical" parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including "Typicals", must be validated for each customer application by customer's technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

RoHS-compliant and/or Pb-free versions of Freescale products have the functionality and electrical characteristics as their non-RoHS-compliant and/or non-Pb-free counterparts. For further information, see <http://www.freescale.com> or contact your Freescale sales representative.

For information on Freescale's Environmental Products program, go to <http://www.freescale.com/epp>.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.
© Freescale Semiconductor, Inc. 2009. All rights reserved.