Freescale Semiconductor
Application Note

Document Number: AN4236
Rev. 0, 11/2010

MC9S12XHY-Family Demonstration Lab Training

by: José Cisneros
Luis Hernandez
Hugo Osornio
Microcontroller Solutions Group, Mexico

1 Introduction

This publication serves to document demonstration lab software examples. The examples show how to configure and use the modules to users getting started with the MC9S12XHY family of MCUs.

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

The complete code is available for all examples. This can be downloaded onto an MC9S12XHY256 target such as the DEMO9S12XHY256 demo board, which this demonstration lab is based on.

Each module of the MC9S12XHY family has its own stand alone software and is discussed within its own section of this document.

Contents

2 Setup

2.1 Tools setup

NOTE

Before starting any of the module examples in this document, it is important to install CodeWarrior™ Development Studio and CodeWarrior for Micro-controllers as described in the *DEMO9S12XHY256 Quick Start Guide* which accompanies the demonstration board.

2.2 Board setup

The steps listed below 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 DEMO9S12XHY256 board must be configured with the default jumper settings as shown in the *DEMO9S12XHY256 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 DEMO9S12XHY256 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 +5 V LED above the ON/OFF switch will illuminate.

3 Demonstration Lab Examples

3.1 Clocks and reset generator

This lab example shows how to produce PLL based bus clocks using the CRG module in its different modes of operation. The example software initializes the PLL to run in RUN Mode at 4 MHz bus, Run Mode 40 MHz bus, Pseudo Stop 8 MHz bus, and Stop Mode 16 MHz bus clock. The changes in the 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 must be followed before running the lab example.

1. Start CodeWarrior by selecting it in the Windows Start menu.
2. From the CodeWarrior main menu, choose File > Open and choose the S12XHY_CRG_Demo.mcp file.
3. Click Open. The project window then opens.
4. 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 choose Project > Debug. This compiles the source code, generates an executable file, and downloads it to the demo board.
6. A new debugger environment opens. After the download process is finished, close the debugger environment.
7. The PLL configuration is sent through the serial communications port on the DEMO9S12XHY256 board (baud rate = 9600, data bits = 8, parity = N, stop bits = 1, flow control = none). Open a terminal window on the PC with this configuration.
8. The bus clock speed is represented on pin 59 PH2/ECLK (to accomplish this demonstration code clear the ECLKCTL register and see the MCU reference manual for further information). 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. This signal is not available in J1 header because it is part of the LCD bus.

3.1.2 Instructions

Follow these instructions to run the lab example.

Run 4 MHz test

1. Press RESET, the MCU is now in run mode with a bus clock frequency of 4 MHz.
2. Monitor the ECLK signal on the oscilloscope. The ECLK matches the bus clock frequency, observe the LED pulse rate, and examine the PLL configuration on the terminal window.

Run 40 MHz test

3. Press RESET while pressing down SW1, make sure to release the RESET switch first. The MCU is now in run mode with a bus clock frequency of 40 MHz.
4. Monitor the ECLK signal on the oscilloscope. The ECLK will match the bus clock frequency, observe the LED pulse rate, and examine the PLL configuration on the terminal window.

Pseudo stop mode test

5. Press RESET while pressing down SW2, make sure to release the RESET switch first. The MCU is now running in Pseudo Stop with a bus clock frequency of 8MHz.
6. Monitor the ECLK signal on the oscilloscope. The ECLK will match the bus clock frequency, observe the LED pulse rate, and examine the PLL configuration on the terminal window.
7. Press the SW4 button, clock pulses will disappear, and the MCU will restart after WD expires with its default clock configuration (4 MHz bus clock).

Stop mode test

8. Press RESET while pressing down SW3, make sure to release RESET switch first. The MCU is now running and prepared for stop mode with a bus clock frequency of 16 MHz.
9. Monitor the ECLK signal on the oscilloscope. ECLK will match the bus clock frequency. Observe the LED pulse rate and examine the PLL configuration on the terminal window.
10. Press the SW4 button, the clock pulses disappear, the MCU enters STOP mode, the bus clock will stop, and the MCU will stay in STOP mode until the RESET switch is pressed.

3.1.3 Summary

The CRG PLL has four operation modes, self-clock mode (which is a fail soft action when the MCU looses the main CLK, see the MCU manual for further information regarding this mode), run mode, pseudo stop, and full stop mode.

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

3.2 Flash programming example

The flash technology module 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 the CCIF=1.

A flow chart of these steps is shown below **Figure 3-1.**

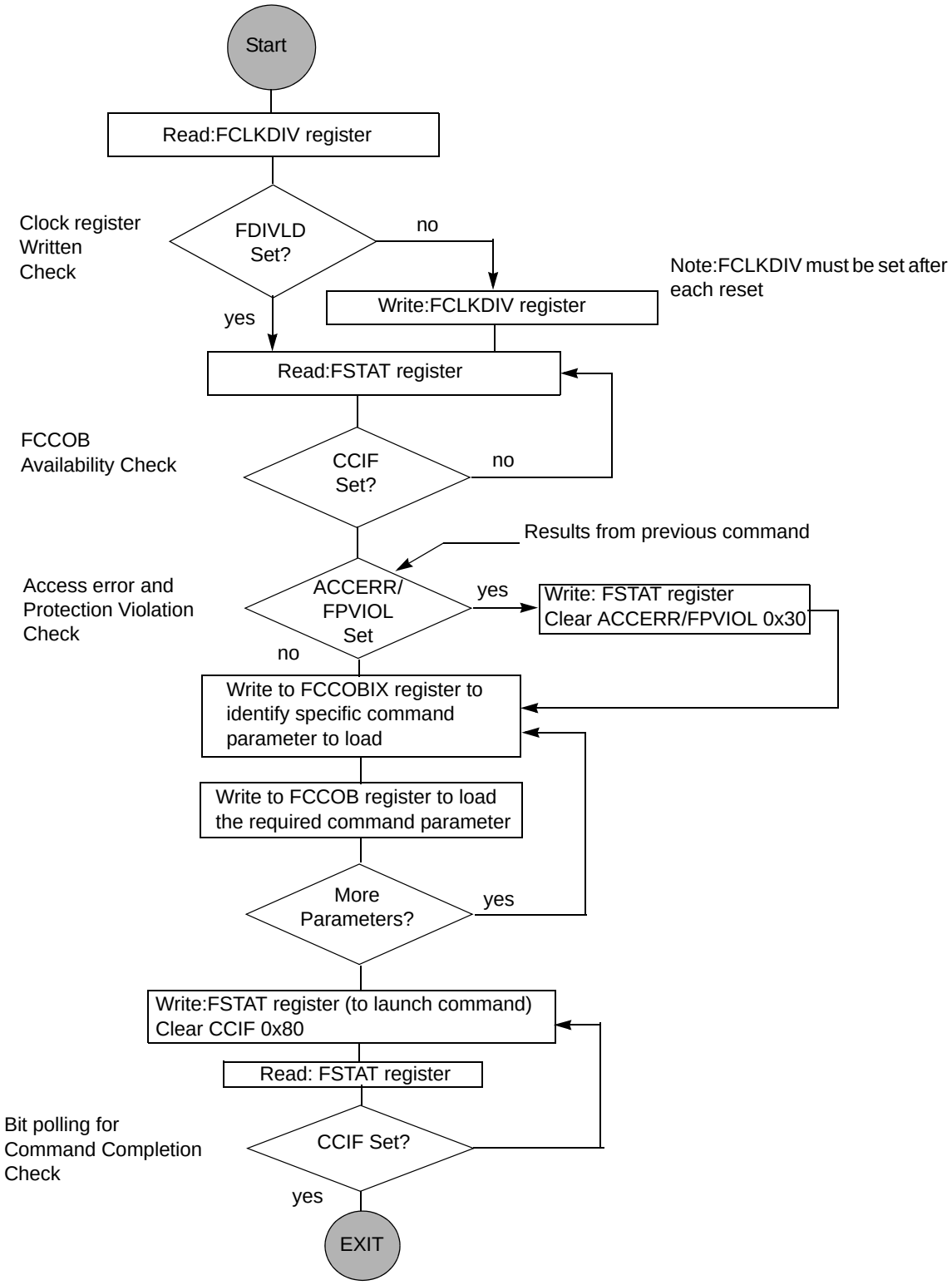


Figure 3-1. Flash programming interface flow chart

3.2.1 Setup

1. Start CodeWarrior by selecting it in the Windows Start menu.
2. From the CodeWarrior main menu, choose File > Open and choose the S12XHY_Flash_Demo.mcp file.
3. Click Open. The project window opens.
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 choose Project > Debug. This compiles the source code, generates an executable file, and downloads it to the demo board.
6. A new debugger environment will open.

3.2.2 Instructions

Notice that this program is compiled and runs from RAM because sections of the flash will be erased in this example. The security information (0x7F_FF0F) in the flash configuration field is not erased.

NOTE

If this is accidently erased the part will be secured and cannot be re-programmed until it is unsecured. For more information on how to unsecure go to the P&E Micro website.

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

- Launch a flash command
- Demonstrate programming and erasing of flash

This example has been written with a series of user software breakpoint.Therefore, the only thing that has to be done is to press the Run button  .

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

3.2.2.1 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 *S12XHY Reference Manual*.

Stepping through will take the user to the function below Figure 3-2.

```
/*****************************************************************************
Function Name : LaunchFlashCommand
Engineer      :   b06320
Date          :   08/06/09
Arguments :
Return    :
Notes     : This function does not check if the Flash is erased.
            This function does not explicitly verify that the data has been
            sucessfully 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 3-2. (Code Snippet) Launch flash command function

3.2.2.2 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). P-Flash pages from F0 to FB have been filled with 0XAAAA. Consider that the store process, particularly in sectors as big as the P-Flash, may take ~20s to be completed Figure 3-3.

D-flash

P-flash

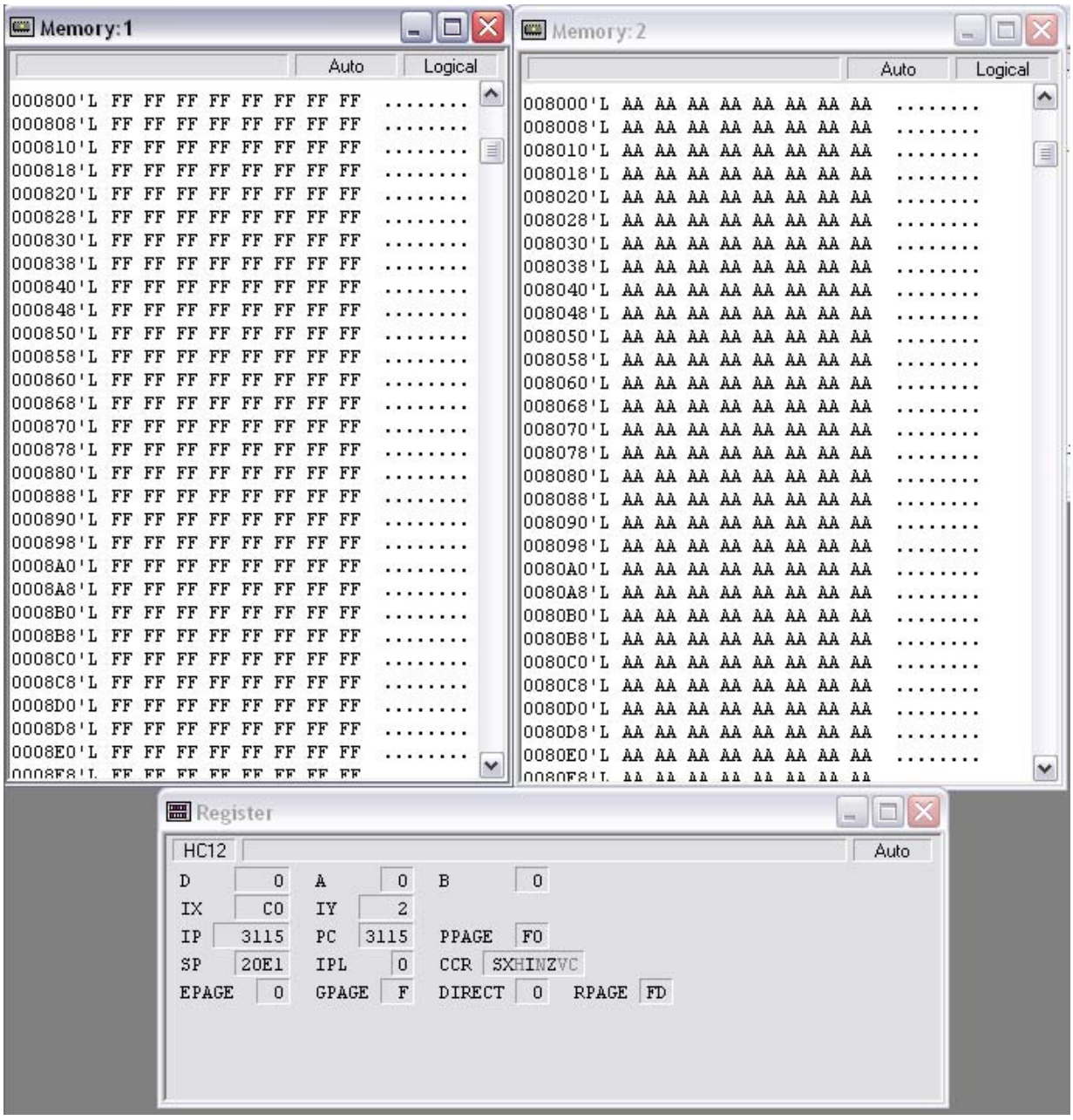


Figure 3-3. The memory maps real time erasing and programming, P-Flash filled with 0xAAAA values

The P-flash window (0x8000 to 0xBFFF) shows 16 K of the P-Flash content, to navigate through all programmed pages change the PPAGE index. Programmed pages in this example are F0 to FB.

The D-flash window (0x0800 to 0x0BFF) shows 1 K of the D-Flash, navigation through the D-Flash's 8 K space is possible by changing EPAGE index from 0 to 7 (8 1K pages) (Figure 3-4).

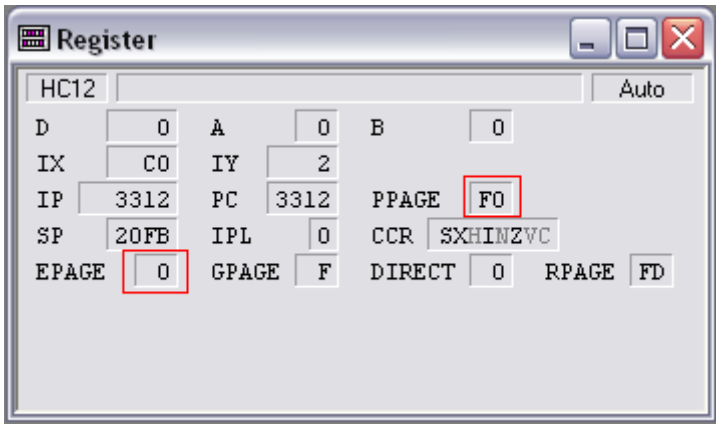


Figure 3-4. Changing P and D flash page indexes

3.2.2.3 Breakpoint 3 — Launched program commands — address data

The store procedure is the same as in previous step, except the data being written to the P-flash is different. The data written is the actual addresses of the P-flash (Figure 3-5).

D-flash

P-flash

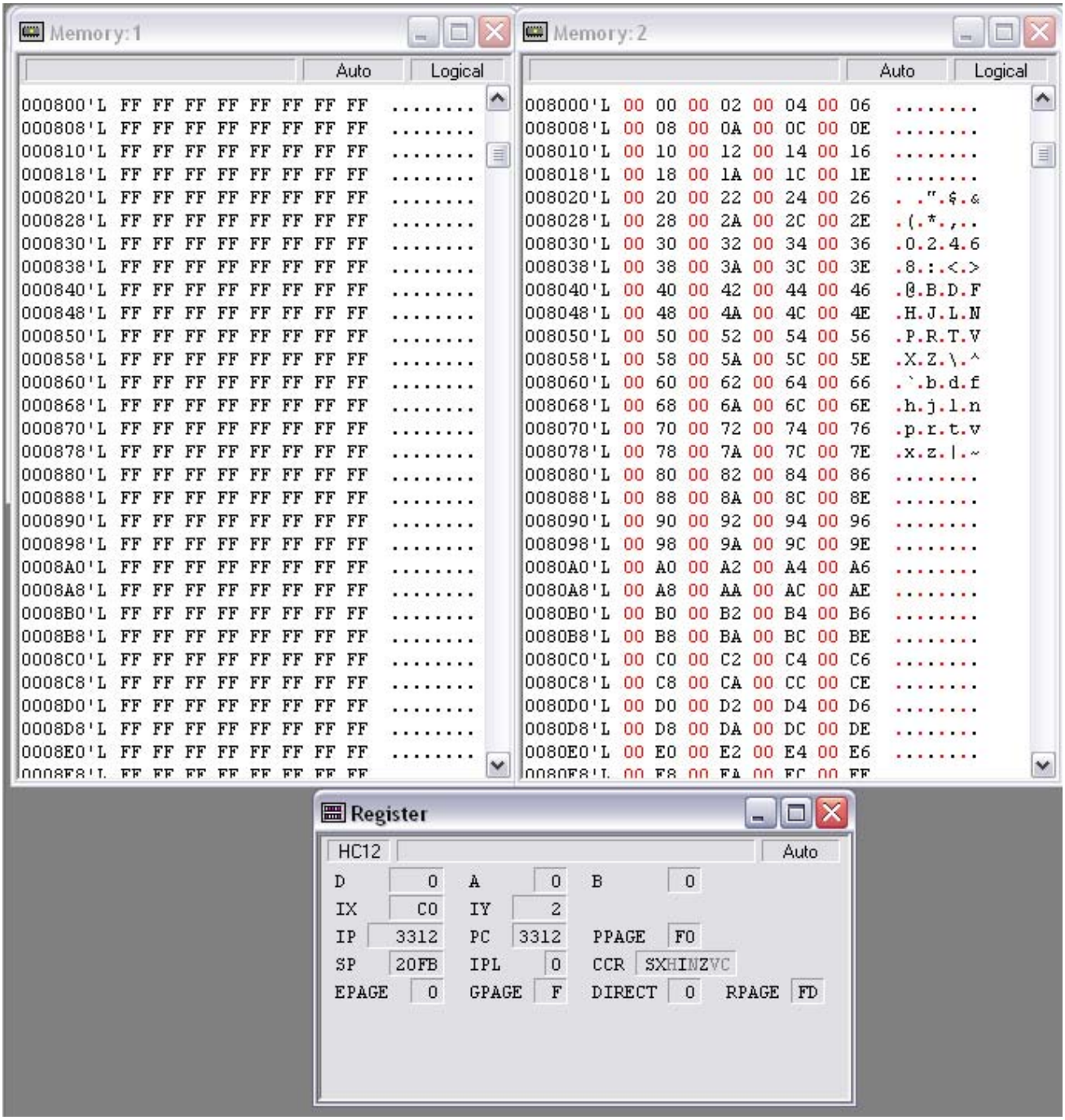


Figure 3-5. Address values stored in P-flash

3.2.2.4 Breakpoint 4 — D-flash — launched program commands

The same functions are used but now they perform activity on the D-flash. The only differences to the flash command function are the memory addresses issued and flash commands, that is D-flash instead of P-flash.

The end of the demonstration is indicated by the LEDs on the evaluation board (EVB) being toggled.

This example does not re-program the device to default, this happens on the next re-load of a program by allowing NVM erasing.

3.2.3 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

The 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 make 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 the S12XHY implements the fixed-length data record scheme emulation on a split gate flash. The emulated EEPROM functionalities are organizing data records, initializing and de-initializing EEPROM, and reporting EEPROM status reading and writing data records.

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

3.3.1 Setup

1. Start CodeWarrior by selecting it in the Windows Start menu.
2. From the CodeWarrior main menu, choose File > Open and choose the S12XHY_EEE_Demo.mcp file.
3. Click Open. The project window opens.
4. The C code of this demonstration is contained within the NormalDemo.c file. Double click on the file to open it.
5. From the main menu choose Project > Debug. This compiles the source code, generates an executable file, and downloads it to the demo board.
6. A new debugger environment opens.

3.3.2 Instructions

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 is continually read and written to, sectors would be copied, and swapped and erased. This example has been written with a series of user software breakpoint, therefore the only thing that has to be done is to press the Run button  .

On start-up, the debugger should begin the program in main as shown in **Figure 3-6.**

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

Figure 3-6. Main function

3.3.2.1 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)
```

Figure 3-7. Breakpoint 1

When selecting the run button, the first software breakpoint is reached. Breakpoint 1 stops at the function responsible for initializing the D-flash. This function erases and assigns the physical D-flash which is used for EEE. Notice that the selected D-flash (0x800, 0x900, 0xA00, 0xB00) is in the erased state.

3.3.2.2 Breakpoint 2 — Initialize the active and alternative sectors

```
returnCode = FSL_InitEeprom();   /* initilise 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();
```

Figure 3-8. Breakpoint 2

The D-flash has now been erased and has to be arranged as active and alternative sectors. This software driver requires that at least 2 alternative sectors be available. This deals with any brownout or dead sector situations. The FSL_InitEeprom function initializes the two sectors at location 0x100000 and 0x100100 to *active* and the remaining sectors 0x100200, 0x100300, and 0x100400 to *alternative*. On completion, the active sectors are defined by 0xFACF0000 and the alternative sectors are defined by 0xFFFF0000.

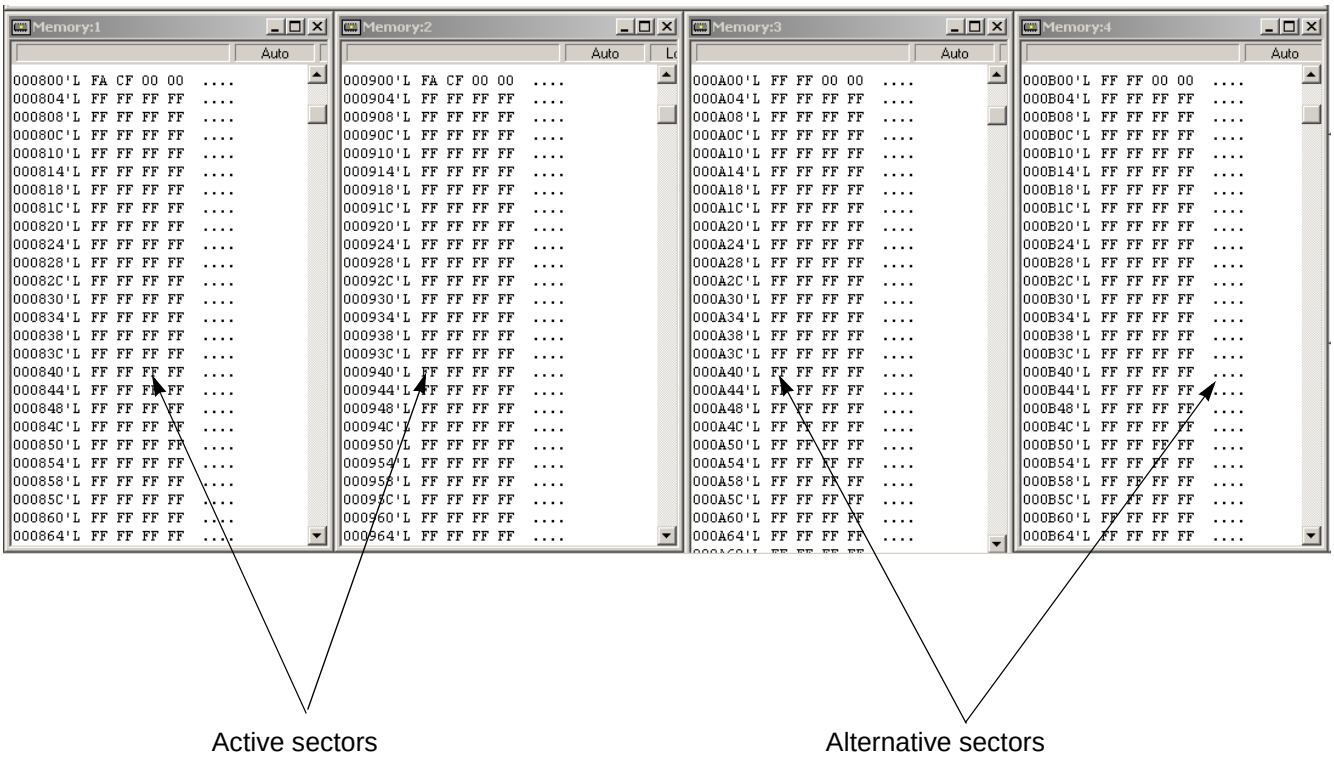
The active sectors are displayed on the memory windows as follows (Table 3-1).

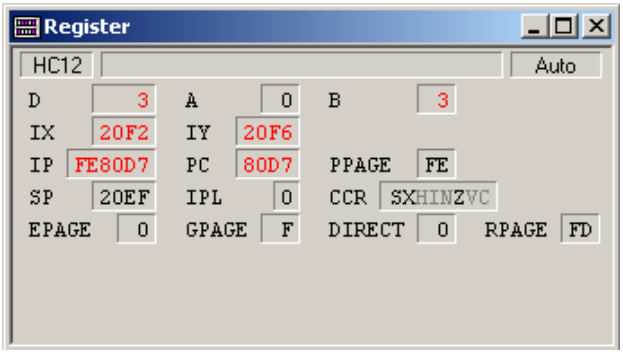
Table 3-1. Active sectors on memory windows

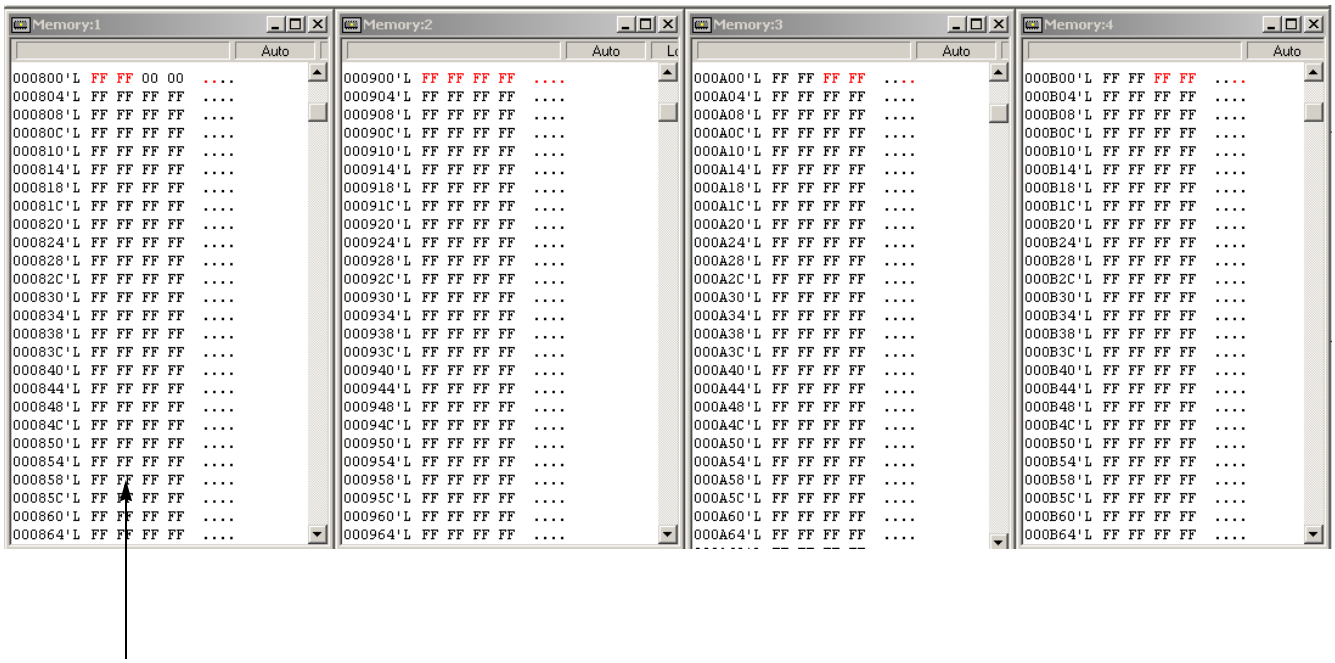
Window	EPAGE	Local Map		Global Address
Memory:1	0	0x800		0x100000
Memory:2	0	0x900		0x100100
Memory:3	0	0xA00		0x100200
Memory:4	0	0xB00		0x100300
Memory:1	1	0x800		0x100400

NOTE

For being able to see the content of the 0x100400 alternative record change the EPAGE register on the Register window as indicated on Figure 3-9.


Memory:1
Memory:2
Memory:3
Memory:4
Auto
Active sectors
Alternative sectors


Register
HC12
Auto
D 3 A 0 B 3
IX 20F2 IY 20F6
IP FE80D7 PC 80D7 PPAGE FE
SP 20EF IPL 0 CCR SXHINZVC
EPAGE 0 GPAGE F DIRECT 0 RPAGE FD



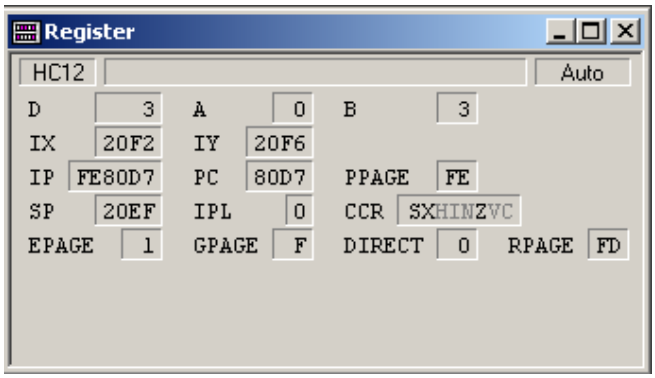


Figure 3-9. EPAGE register change

3.3.2.3 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();
```

Figure 3-10. Breakpoint 3

The data to be written is defined within a header file (Figure 3-11) – 0x10. This function when executed writes the data 0x10 and assigns 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.

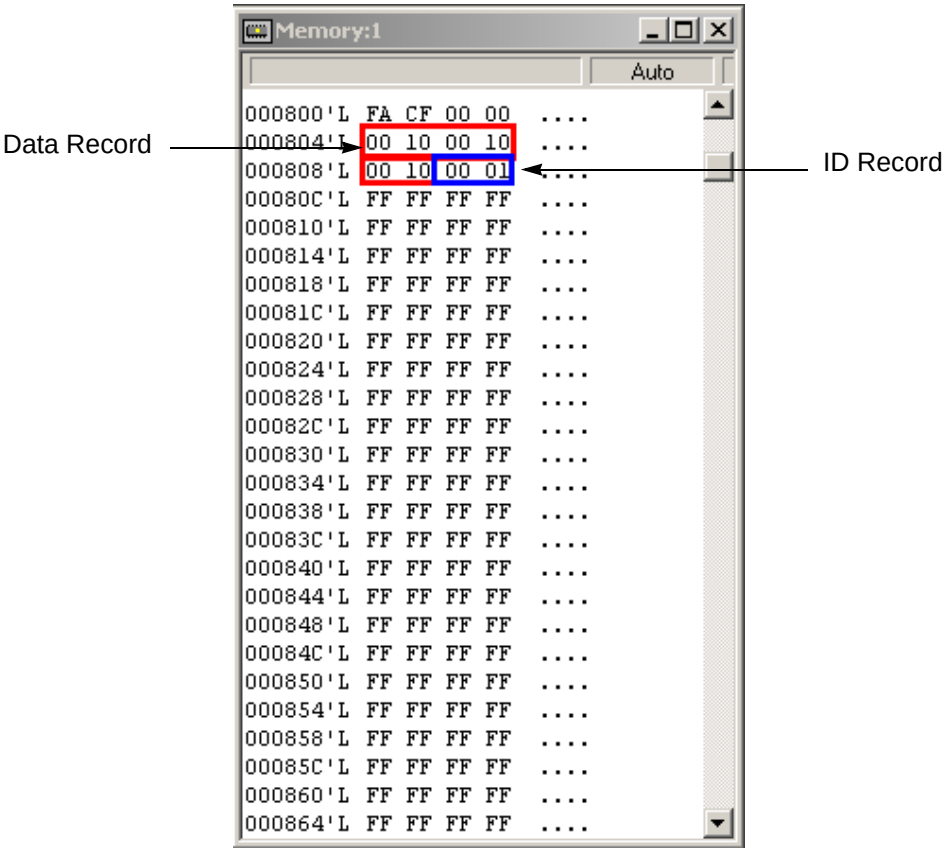


Figure 3-11. Header file content

3.3.2.4 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)
    {
```

Figure 3-12. Breakpoint 4

This writes data records so that the active block is completely filled. The next write after the loop causes a swap. The active sectors at 0x800 and 0x900 have been filled because the code executes a loop until it reaches the end of the second active sector.

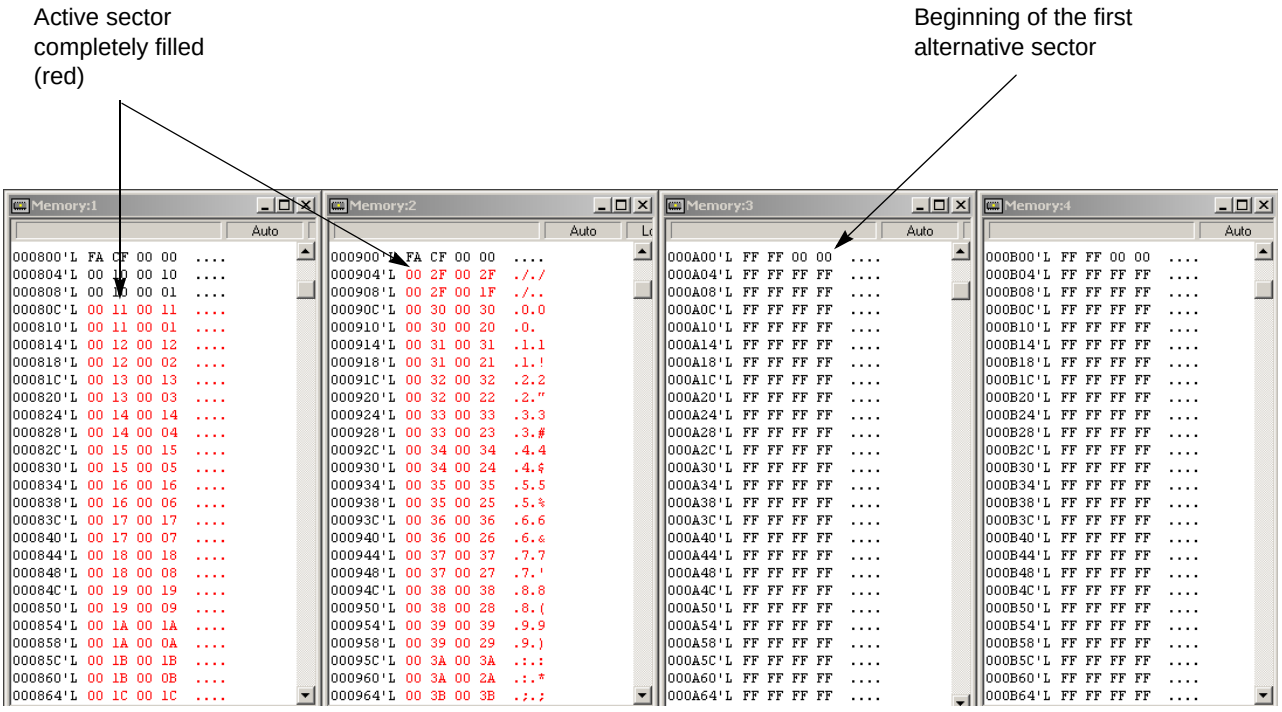


Figure 3-13. Sector filling

3.3.2.5 Breakpoint 5 — Sector swap

Now that the active sectors have been completely filled, the next record write only occurs after a new active sector has been created. This is a two-stage process. First, all the records from the first active sector are copied to the first available alternative sector, in this case, data and ID records from 0x800-0x899 are

copied to 0xA00. Second, the sector at 0x800 is erased and becomes a new alternative sector. Notice that on the new alternative sector it begins with 0xFFFF0001.

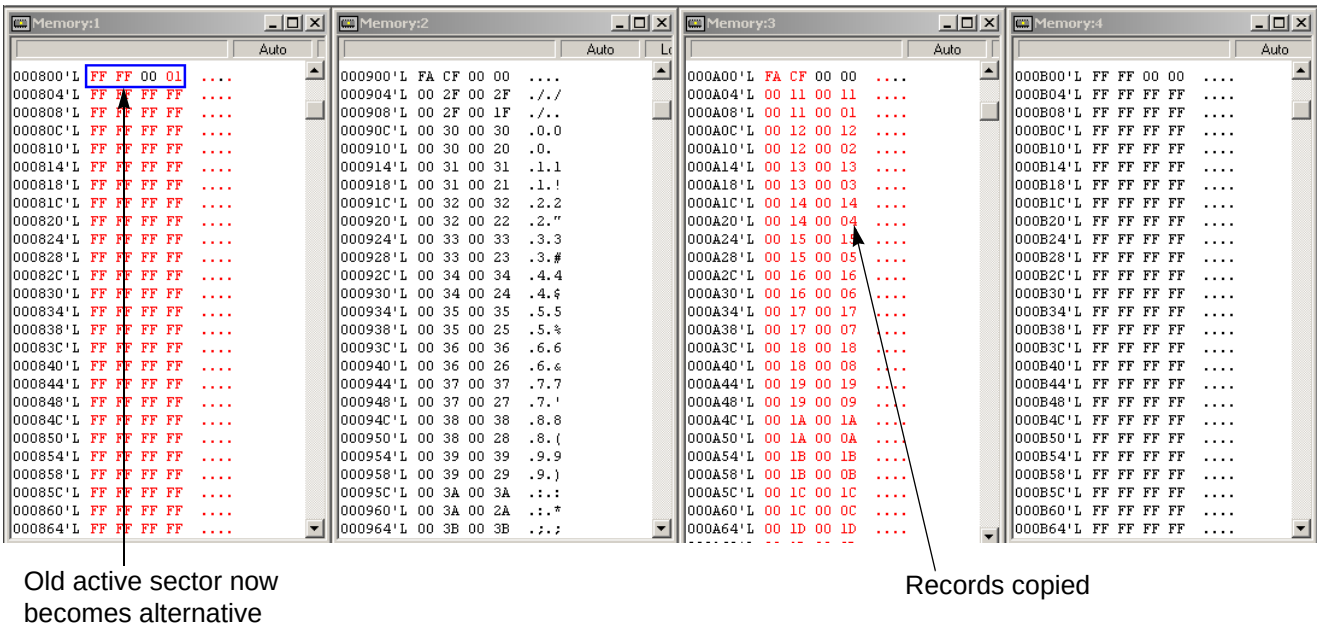


Figure 3-14. Sector SWAP

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

3.3.2.6 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)
     {
```

Figure 3-15. Breakpoint 6

Functions after breakpoint 6 read a specific EEE address from the stored data tables.

This demonstration program completes by erasing the D-flash sectors and ends while in the loop.

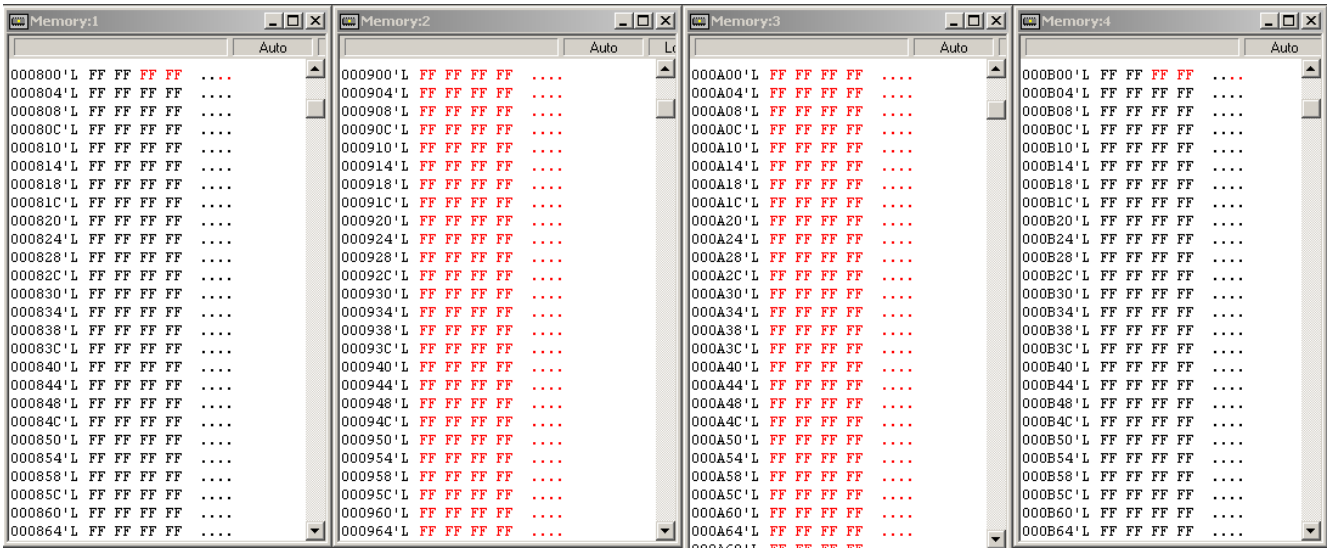


Figure 3-16. EEE data read

3.3.3 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 is the two latter functions that are relied upon. Moreover, the software is capable of dealing with brownout events as well as dead sectors. For developing applications with this code read the *EEE Driver User's Guide* included in this pack.

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 must be followed before running the lab example.

1. Start CodeWarrior by selecting it in the Windows Start menu.
2. From the CodeWarrior main menu, choose File > Open and choose the S12XHY_MSCAN_Demo.mcp file.
3. Click Open. The project window opens.

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 choose Project > Debug. This compiles the source code, generates an executable file, and downloads it to the demo board.
6. A new debugger environment will open. After the download to the demo board is complete, close the debugger environment.

3.4.2 Instructions

Follow these instructions to run the lab example.

1. Press RESET. The MSCAN demo software begins execution.
2. Press various combinations of the SW1, SW2, SW3, and SW4 switches. The LEDs must 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 a wide variety of uses that require reliable communications.

3.5 PWM module

This lab provides an example of how to setup and use the pulse width modulation (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 must be completed before running the lab example.

1. Start CodeWarrior by selecting it in the Windows Start menu.
2. From the CodeWarrior main menu, choose File > Open and choose the S12XHY_PWM_Demo.mcp file.
3. Click Open. 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 choose Project > Debug. This compiles the source code, generates an executable file, and downloads it to the demo board.
6. A new debugger environment will open. After the download to the demo board is complete, close the debugger environment.

3.5.2 Instructions

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. Press RESET. The code must run and re-route the PWM channels. The PWM module must 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 lighting.

3.6 Low Power Modes

In addition to the default run mode, the MC9S12XHY has three low power modes, wait, pseudo stop and stop.

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

Self clock mode is used for safety purposes. It provides a reduced functionality in case a loss of clock is causing severe system conditions.

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.

Table 3-2. Power modes

Mode	Bus Clock	External Oscillator
Run	Y	Y
Wait	Y	Y
Self Clock	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 must be taken to probe the ECLK and EXTAL separately to avoid adding extra noise to the signals.

3.6.1 Setup

The steps below must be followed before running the lab example.

1. Start CodeWarrior by selecting it in the Windows Start menu.
2. From the CodeWarrior main menu, choose File > Open and choose the S12XHY_Low_Power_Modes.mcp file.
3. Click Open. 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 choose Project > Debug. This compiles the source code, generates an executable file, and downloads it to the demo board.
6. A new debugger environment will open. After the download to the demo board is complete, 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, this signal is not available in the J1 header because it is part of the LCD bus.
8. The oscillator can be monitored by attaching a scope probe to the EXTAL side of the Y1 crystal.

NOTE

If the CLK is monitored in the XTAL, additional noise is input to the device and may cause the MCU not to run properly.

3.6.2 Instructions

Follow these instructions to run the lab example:

Run mode test

1. Press RESET. The MCU is now operating in run mode. The LEDs flash indefinitely indicating the MCU is in run mode.
2. Monitor the ECLK signal on the oscilloscope. ECLK represents the bus clock. A 40 MHz signal should be observed.
3. Monitor the EXTAL signal on the oscilloscope. The EXTAL indicates that the crystal oscillator is running. An 8 MHz sine wave is observed.

Wait mode test

1. Press RESET while pressing down SW1 and first release RESET then SW1. LED1 flashes twenty times indicating run mode and then the MCU enters wait mode. Pressing SW4 causes the MCU to exit wait mode and go back into run mode. LED1 flashes twenty times before the MCU returns to wait mode.
2. Monitor the ECLK signal on the oscilloscope. The 40 MHz clock signal representing the bus clock is present in both run and wait modes.
3. Monitor the EXTAL signal on the oscilloscope. In both run and wait modes, an 8 MHz sine wave is observed indicating that the external oscillator continues to operate in wait mode.

Pseudo stop mode test

1. Press RESET while pressing down SW2 and first release RESET then SW2. LED1 flashes twenty times indicating run mode and then the MCU enters into pseudo stop mode. Pressing SW4 causes the MCU to exit pseudo stop mode and back into run mode. LED1 flashes twenty times before the MCU returns to pseudo stop mode.
2. Monitor the ECLK signal on the oscilloscope. The 40 MHz clock signal representing the bus clock is present in run mode. In pseudo stop mode, the bus clock is stopped to save power.
3. Monitor the EXTAL signal on the oscilloscope. In both run and pseudo stop modes, an 8 MHz sine wave is observed indicating that the external oscillator continues to operate in pseudo stop mode.

Stop mode test

1. Press RESET while pressing down SW3 and first release RESET then SW3. LED1 flashes twenty times indicating run mode and then the MCU enters into stop mode. Pressing SW4 causes the MCU to exit stop mode and back into run mode. LED1 flashes twenty times before the MCU enters and returns into stop mode.
2. Monitor the ECLK signal on the oscilloscope. The 40 MHz clock signal representing the bus clock is only present in run mode. In stop mode, the bus clock is stopped to save power.
3. 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 MC9S12XHY 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.

3.7 MMC program flash paging window

The MC9S12XHY256 has a flash memory of 256 KB. Because the 256 KB of memory cannot be addressed by the 16-bit MC9S12XHY256 MCU, there is insufficient local addressing space to accommodate all of the flash memory, the RAM memory, and the register space. Instead, a paging system is used which maps 16 KB flash memory blocks, 4 KB RAM memory blocks, and 1 KB Emulated EEPROM memory blocks. The paging system maps the blocks into the local memory map as shown in the next table.

Table 3-3. Local memory map addresses

Memory	Local Memory Map Address
EEPROM	0x800 to 0xBFF
RAM	0x1000 to 1FFF
FLASH	0x8000 to 0xBFFF

The MC9S12XHY256 supports Global addressing access. Global addresses are 23 bit addresses that cover an 8 MB address space, ranging from addresses 0x000000 to 0x7FFFFF. In this linear global address space all memory resources are grouped and the GPAGE register can be used to access all RAM, EEPROM, and

FLASH locations, as well as external memory space. See the memory map included in the MC9S12XHY256 Reference Manual.

This lab example shows how to use the paging capability of the MMC module to access global memory addresses within the local memory map. As well as how to use the Global Addressing Access for the different memory resources.

3.7.1 Setup

The following steps must be followed before running the lab example

1. Start CodeWarrior by selecting it in the Windows Start menu.
2. From the CodeWarrior main menu, choose File > Open and choose the S12XHY_MMC_Demo.mcp file.
3. Click Open. 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 choose Project > Debug. This compiles the source code, generates an executable file, and downloads 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:1 window in the debugger environment is configured to show the first few locations of the P-Flash Window at address 0x8000. Note that the P-Flash Memory mapping on this example does not map the pages FF nor FD. This is made intentional due to that the memory locations are non banked. Even though these non banked locations are still accessible by the paging system.

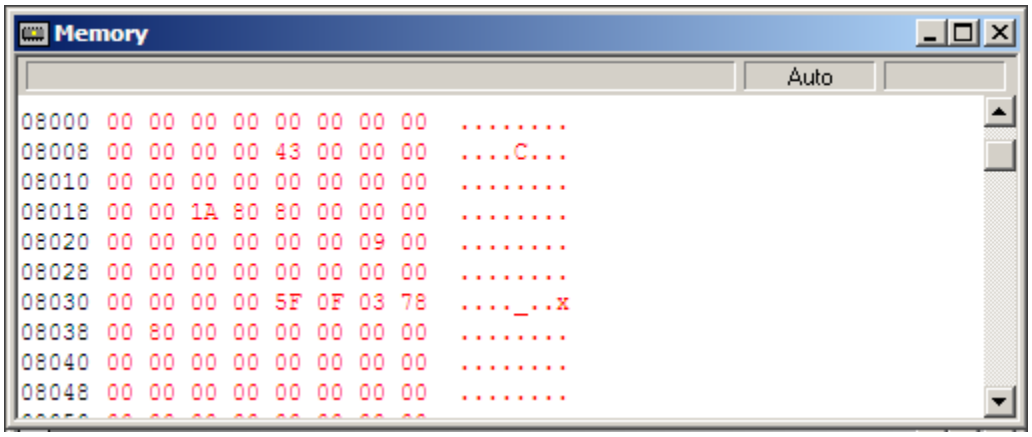


Figure 3-17. P-Flash window

2. The Register window in the debugger environment shows the setting of the program page index register (PPAGE).

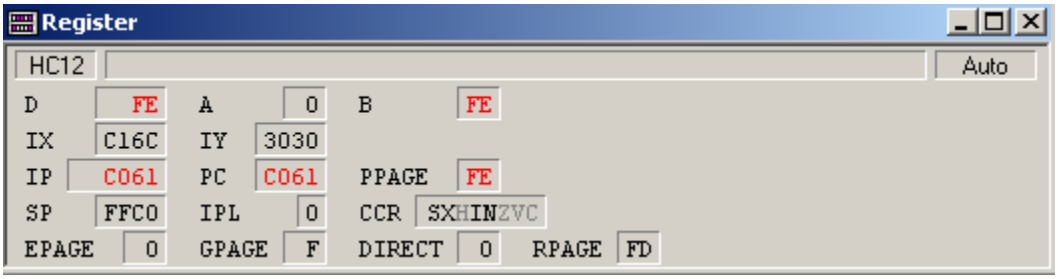


Figure 3-18. PPAGE register

3. Start the software by clicking on the Run button  .
4. When the software reaches the first software breakpoint, examine the contents of the Memory and Register windows. The PPAGE register is set to FE and the P-Flash Window shown in the Memory window displays the contents of PPAGE FE.

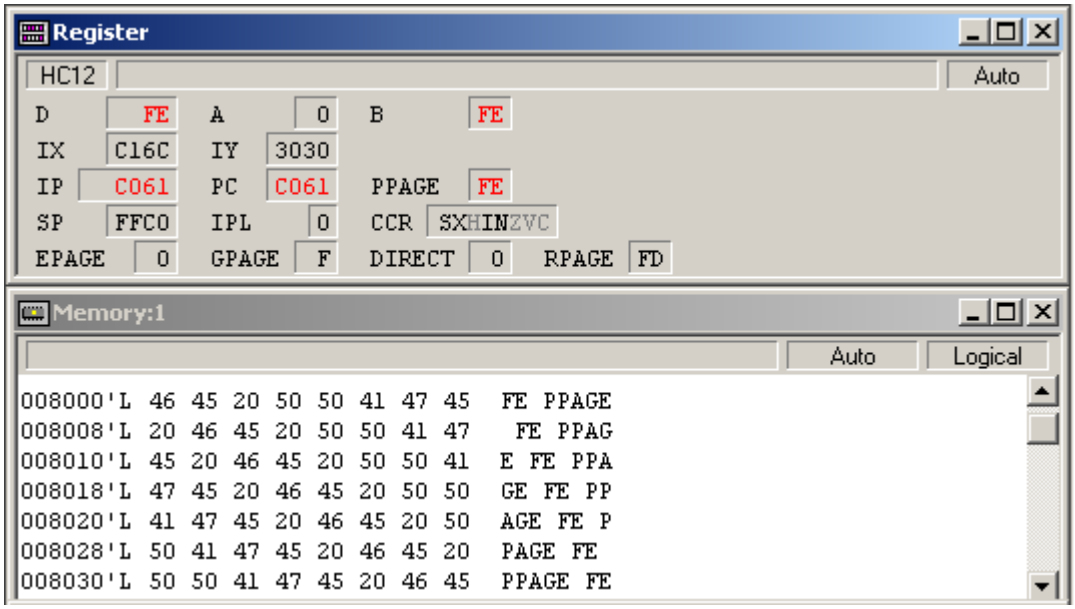


Figure 3-19. PPAGE FE contents

5. Press the Run button and observe the Memory and Register windows at the next breakpoint.
6. Now the PPAGE register is set to FC and the P-Flash Window shown in the Memory window displays the contents of PPAGE FC.

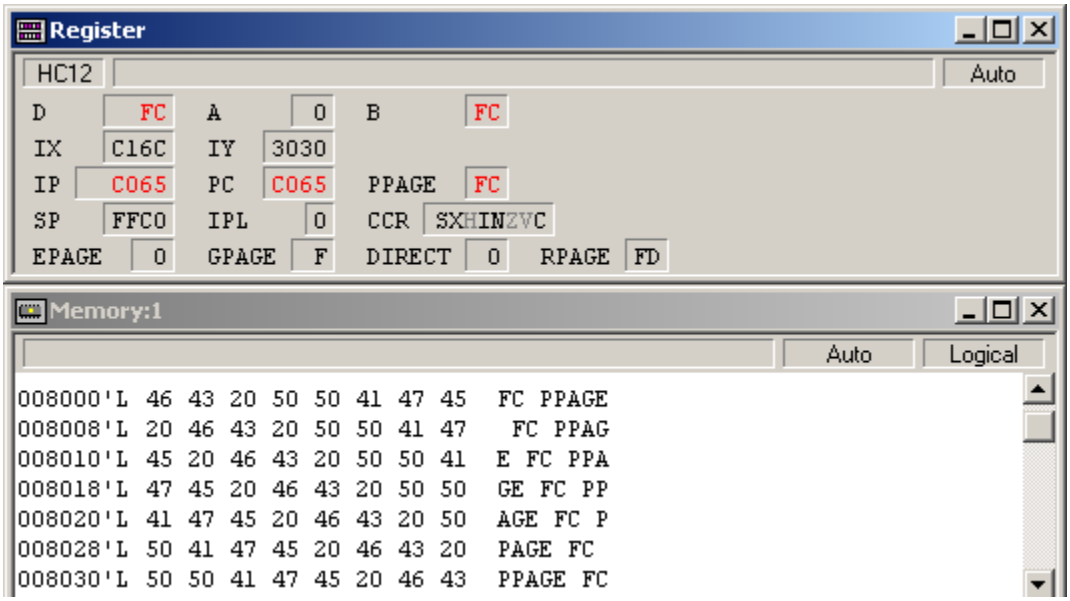


Figure 3-20. PPAGE FC contents

7. Press the Run button as many times as necessary to reach the PPAGE F0 and observe that every time that you reach a breakpoint the Memory and Register windows are updated accordingly.
8. Now, after the PPAGE register is set to F0 and the P-Flash Window shown in the Memory window displays the contents of PPAGE F0.
9. The Memory:2 window in the debugger environment is configured to show the first few locations of the RAM Window at address 0x1000. Notice that the RAM Memory mapping in this example maps the pages from FF to FD. The sections that correspond to the pages FF and FE are non banked. Even though these non banked locations are still accessible by the paging system.
10. Press the Run button to start running the RPAGE section after reaching the first breakpoint, observe that register RPAGE is set to FF and the contents in Memory:2 are unknown data, this is because this page was not set with information. This page is non banked memory, but it is still accessible through the RAM paging system.

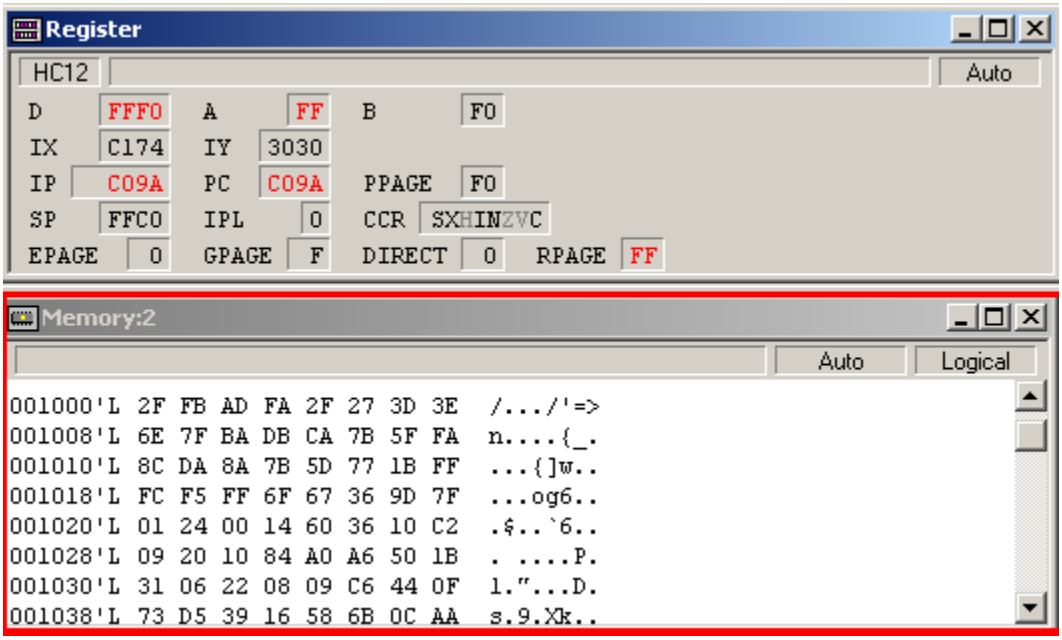


Figure 3-21. RPAGE FF Contents

11. Press the Run button as many times as necessary to reach the RPAGE FD and observe that every time a breakpoint is reached the Memory and Register windows are updated accordingly.
12. Now the RPAGE register is set to FD and the Window shown in the Memory:2 window, displays the contents of RPAGE FD.

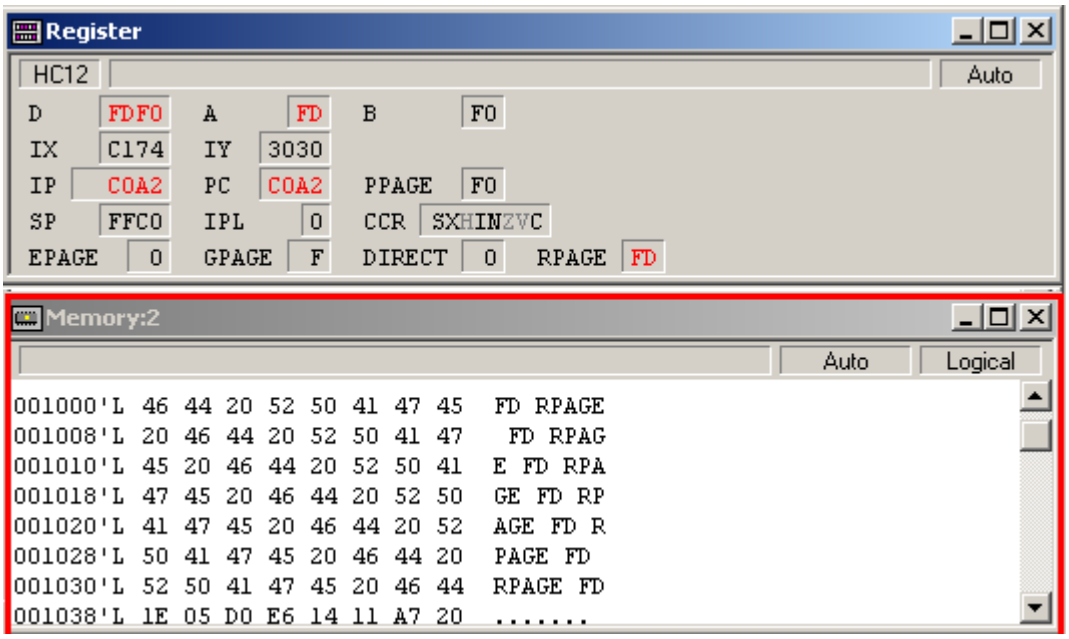


Figure 3-22. RPAGE FD contents

13. After the RPAGE register is set to FD and the RAM window, the Memory window displays the contents of RPAGE FD.

14. The Memory:3 window in the debugger environment is configured to show the first few locations of the Emulated EEPROM window at address 0x1000. Notice that the Emulated EEPROM Memory mapping in this example maps the pages from 00 to 0F. The sections that correspond to the pages 00 and 0F are banked.
15. Press the Run button to start running the EPAGE section after reaching the first breakpoint. Observe, that the EPAGE register is set to 7 and the contents in Memory:3 are EPAGE 07 data.

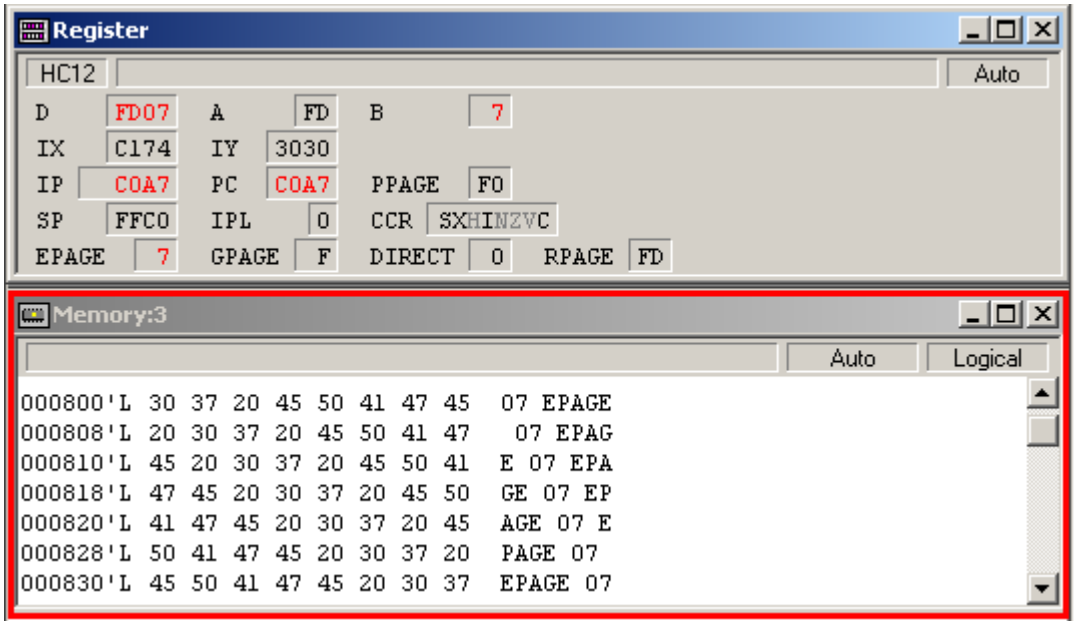


Figure 3-23. EPAGE 7 contents

16. Press the Run button as many times as necessary to reach the EPAGE 00 and notice that every time the breakpoint is reached the Memory and Register windows are updated accordingly.
17. The EPAGE register is now set to 00 in the window shown. The Memory:3 window displays the contents of EPAGE 00.

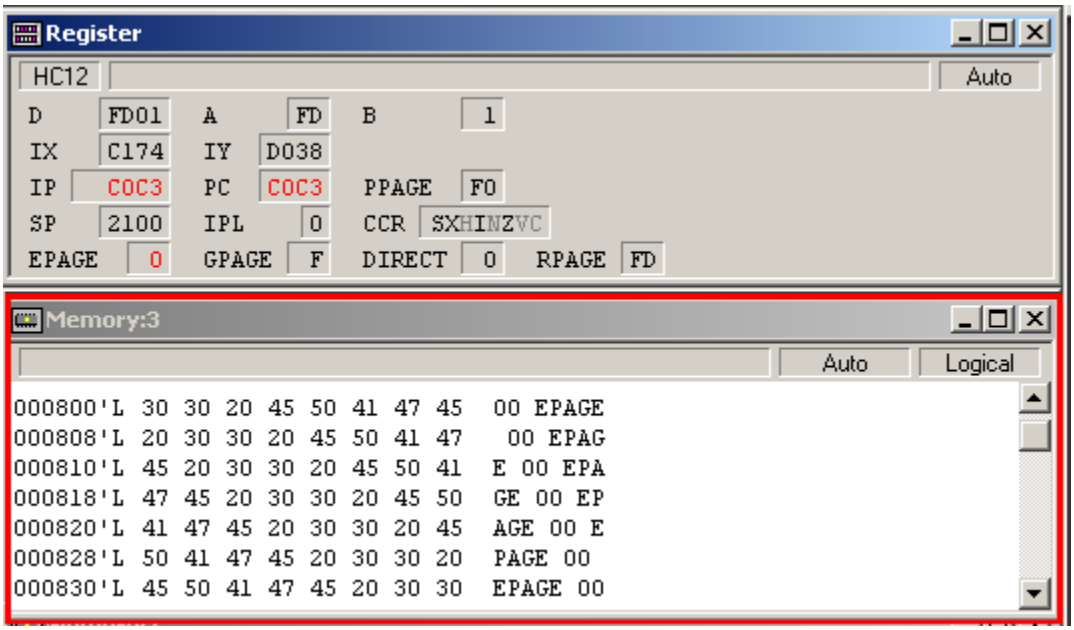


Figure 3-24. EPAGE 0 contents

18. After the EPAGE register is set to 00 and the EEPROM Window shown. The Memory window displays the contents of EPAGE 00.
19. Press the Run button to start running the GPAGE section after reaching the first breakpoint, notice that the GPAGE register is set to 7 and the contents in the Memory:4 window are PPAGE F0 data. The data of window Memory:4 was read from the P-Flash PPAGE F0. This data was accessed using Global addressing.

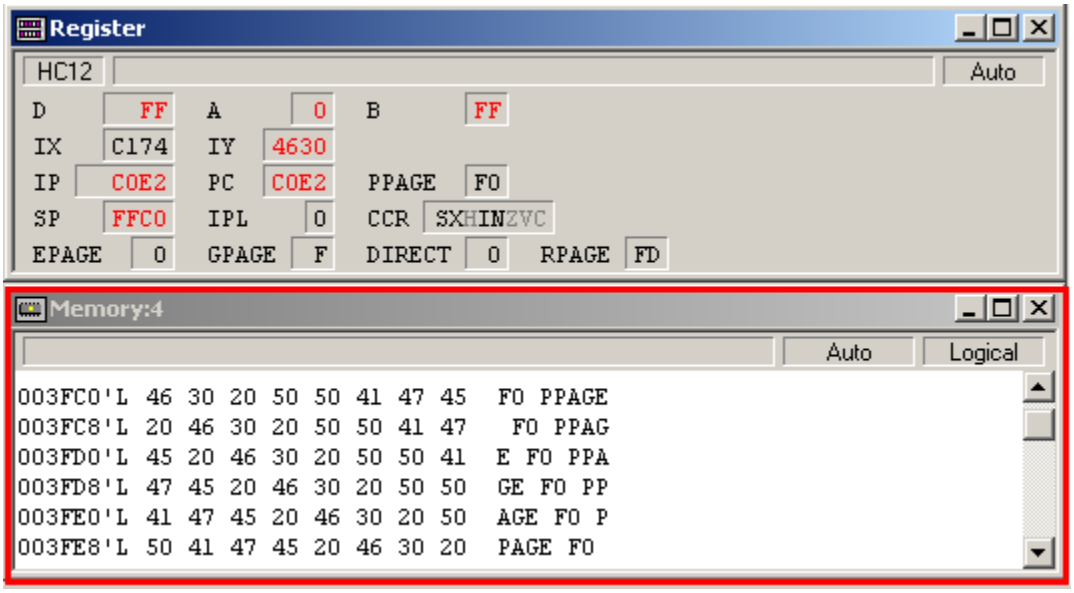


Figure 3-25. Global memory addressing access to P-FLASH

20. Press the Run button and the second breakpoint is reached, observe that the GPAGE register is set to 7 and the contents in the Memory:4 window are RPAGE FD data. The data of window Memory:4 was read from the RAM RPAGE FD. This data was accessed using Global addressing.

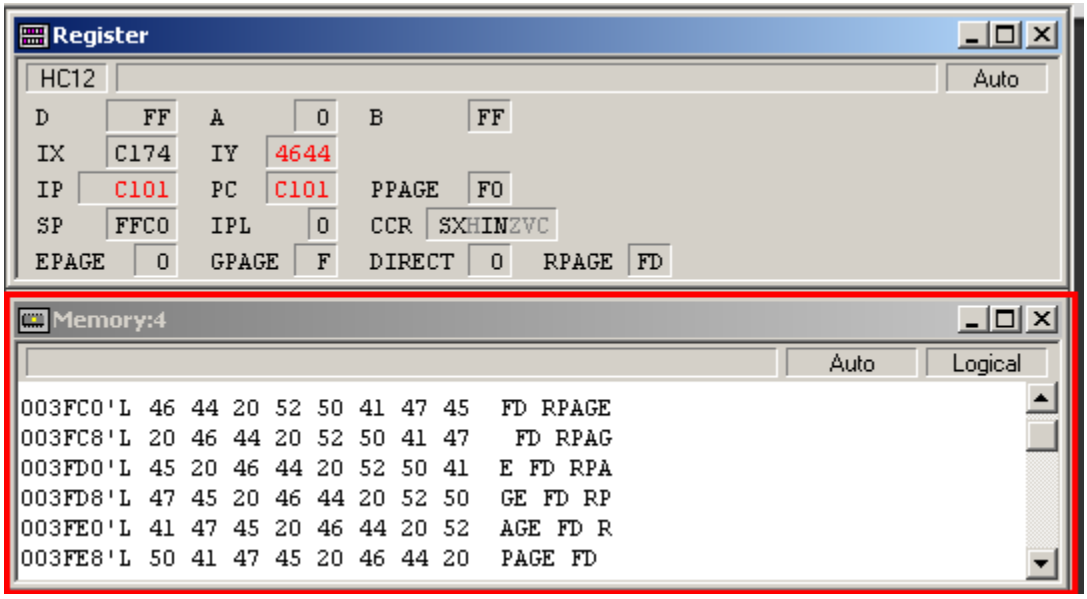


Figure 3-26. Global memory addressing access to RAM

21. Press the Run button and the third breakpoint is reached. Notice that the GPAGE register is set to 7 and the contents in Memory:4 window are EPAGE 00 data.

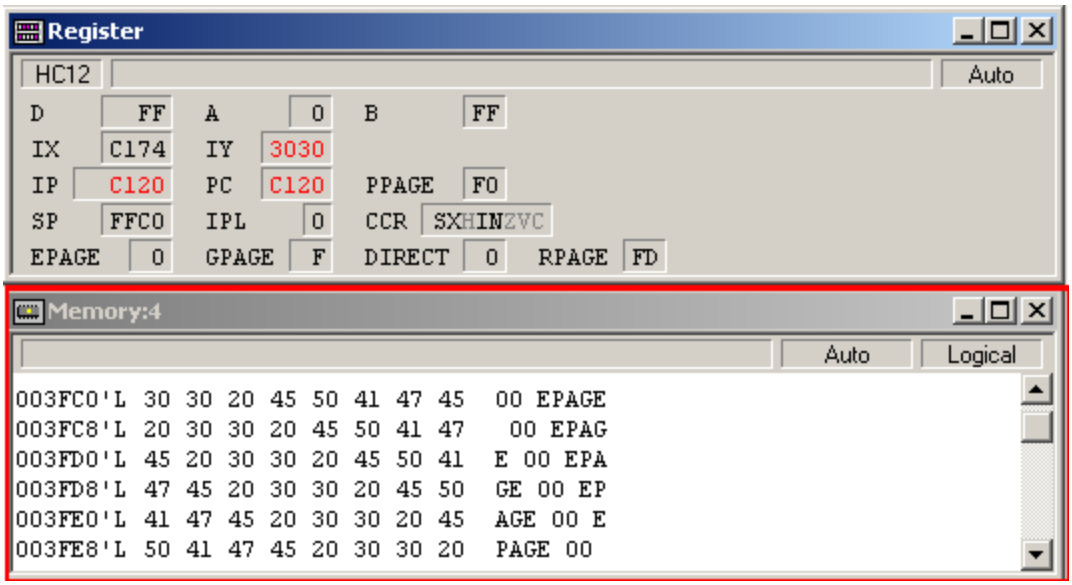


Figure 3-27. Global memory addressing access to Emulated EEPROM

3.7.3 Summary

The MMC module can be used to expand the accessible amount of memory of the MC9S12XHY256 MCU by paging the expanded global memory into a window in the local memory.

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 must be followed before running the lab example.

1. Start CodeWarrior by selecting it in the Windows Start menu.
2. From the CodeWarrior main menu, choose File > Open and choose the S12XHY_ADC_Demo.mcp file.
3. Click Open. 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 choose Project > Debug. This compiles the source code, generates an executable file, and downloads it to the demo board.
6. A new debugger environment will open. After the download to the demo board is complete, 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 = none). Open a terminal window on the PC with this configuration.

3.8.2 Instructions

Follow these instructions to run the lab example.

1. Press 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 the potentiometer RV1 on PAD00 and observe the changes in the terminal window.
3. Press RESET while pressing down SW1 and release the switches, following the next sequence RESET then 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. Press RESET while pressing down SW2 and release the switches, following the next sequence RESET then SW2. The ADC will perform continuous 12-bit conversions on PAD00 and compare the result to see if it is higher than 0x07FF. While the comparison is true, LED1 on the demo board flashes.
6. Vary the conversion result by turning potentiometer RV1 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 must be on the same or multiple input channels. An automatic compare can be used to compare the conversion result against a programmable value for higher than, less than, or equal to matching. Any conversion sequence can be repeated continuously without additional MCU overhead.

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

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 DEMO9S12XHY256 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 must be completed before running the lab example

1. Ensure that the LED4 and POT jumpers on the JP14 have been removed.
2. Connect the potentiometer output (pin 18 on JP14) to port R3 (pin 7 on JP14). This allows the potentiometer RV1 to provide a stimulus to the input capture function on IOC1_7.
3. Start CodeWarrior by selecting it in the Windows Start menu.
4. From the CodeWarrior main menu, choose File > Open and choose the S12XHY_Timer_Demo.mcp file.
5. Click Open. 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 choose Project > Debug. This compiles the source code, generates an executable file, and downloads it to the demo board.
8. A new debugger environment will open. After the download to the demo board is complete, close the debugger environment.

3.9.2 Instructions

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. Press RESET. The code runs and re-routes the timer channels.

2. Timer 0 performs 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 (LED2) will toggle. When a match compare occurs on IOC0_6, port R0 will toggle.
3. Timer 1 performs 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, notice LED3 toggles with each rising or falling edge.

3.9.3 Summary

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

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 must 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 Windows Start menu.
3. From the CodeWarrior main menu, choose File > Open and choose the S12XHY_SCI_Demo.mcp file.
4. Click Open. 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;
```

Figure 3-28. Baud rate configuration section

3. From the main menu choose Project > Debug. This compiles the source code, generates an executable file, and downloads 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 = None) to see the RS-232 port data.
6. Press Run. The code begins execution and configuring the SCI to the selected baud rate. Its 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. Choose 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 the 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.

3.11 SPI Communications

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

As there is only one SPI module available on the DEMO9S12XHY256 board this example is limited to transmitting data only. When an SPI master transmits data to an SPI slave, data is usually received simultaneously and synchronized by a serial clock.

3.11.1 Setup

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

1. Start CodeWarrior by selecting it in the Windows Start menu.
2. From the CodeWarrior main menu, choose File > Open and choose the S12XHY_SPI_Demo.mcp file.
3. Click Open. 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 choose Project > Debug. This compiles the source code, generates an executable file, and downloads it to the demo board.
6. A new debugger environment will open. After the download to the demo board is complete, close the debugger environment.
7. Attach scope probes to signals PS5, 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. Press RESET. The code begins 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 (PS5), and the Serial Clock (PS6) signals.

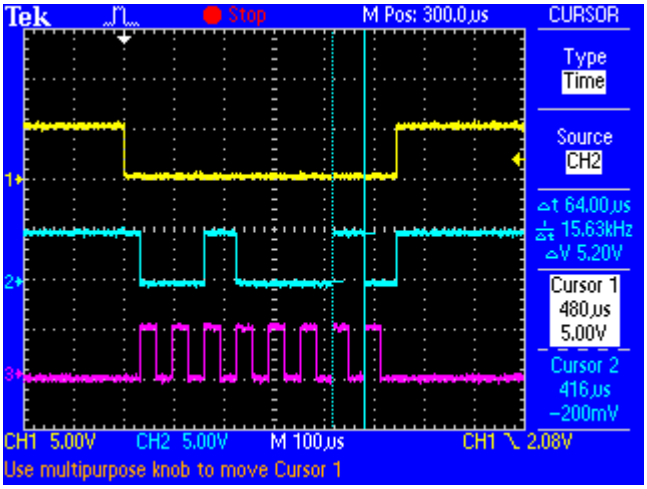


Figure 3-29. SPI protocol signals

3.11.3 Summary

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

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. To operate this demo, connecting an external motor to PTU0-3 is required.

3.12.1 Set-up

1. Connect a stepper motor to motor 0 pins, PTU0-3 –J3 on the demo board.
2. Start CodeWarrior by selecting it in the Windows Start menu.
3. From the CodeWarrior main menu, choose File > Open and choose the S12XHY_MC_DEMO.mcp file.
4. Click Open. 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.
6. From the main menu choose Project > Debug. This compiles the source code, generates an executable file, and downloads 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 four connections (Figure 3-30), where each PWM channels has two connections. Refer to the section *Motor Controller* in the *MC9S12XHY256 Reference Manual*.

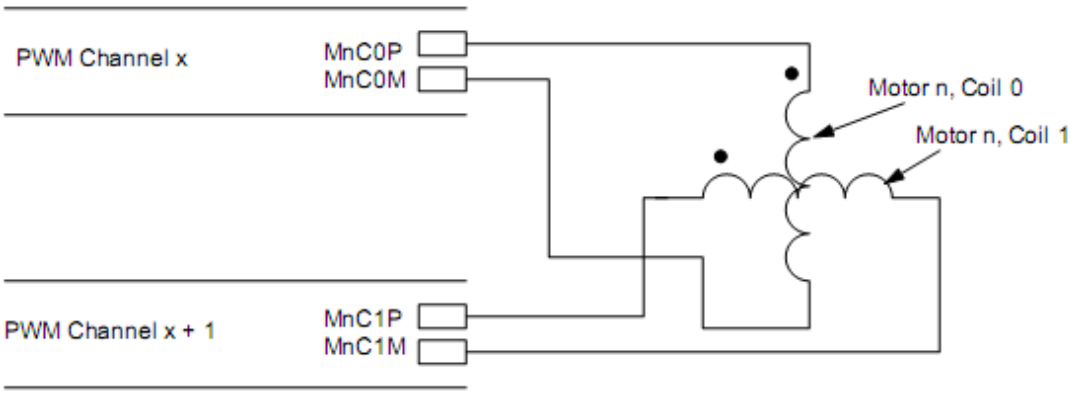


Figure 3-30. —Dual full H-bridge configuration

3.12.2 Instructions

This is a self-contained example, which requires no intervention. On running the program, the motor initializes by returning to zero (RTZ) position. The motor is controlled by adjusting 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, that increments in real time. There are also other common items displayed and updated on the LCD to emulate a dashboard unit display. This document describes the software and explains the operation of the LCD module.

3.13.1 Set-up

1. Start CodeWarrior by selecting it in the Windows Start menu.
2. From the CodeWarrior main menu, choose File > Open and choose the S12XHY_LCD_DEMO.mcp file.
3. Click Open. 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 choose Project > Debug. This compiles the source code, generates an executable file, and downloads it to the demo board.

To begin using the LCD, the pins that 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 output an LCD driver waveform based on the DUTY and BIAS settings in LCDCR0.

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

Figure 3-31. LCD initialization functions

The frequency at which the LCD glass must be operated is determined 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. Refer to the section *Liquid Crystal Display* in the *MC9S12XHY256 Reference Manual* which provides information on the LCD clock vs. the frame frequency.

The LCD module has a dedicated 20 bytes of RAM at 0x208 which contains 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.

3.13.2 Instructions

This is a self-contained example and requires no intervention. On program start-up, the LCD displays S12XHY and will proceed into a never-ending loop which is responsible for the updating and animation of the LCD display.

A port interrupt has also been used on analogue ports which 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, notice then that the odometer and trip information have been reset. A further exercise to this example is to use the emulated EEPROM driver, to enable recovery of this information, by continuously reading and writing the data to the device's D-flash, available for free on the Freescale website.

3.14 Stepper stall detect module

This demonstration code has been constructed to demonstrate SSD module capabilities using a back EMF signal to determine stall detection on a stepper motor. This module consists of an internal hardware block included in the S12XHY microcontroller family. This hardware module shares the same pins with the motor control module (MCM), thus no additional wiring is required to use SSD and MCM for stepper motor control.

To operate this demo, it is required to connect an external stepper motor to MOTOR0 pins on the DEMO9S12XHY256 board.

3.14.1 Set-up

6. Connect a stepper motor to MOTOR0 pins, PTV0–3 –J3 on the demo board.
7. Start CodeWarrior by selecting it in the Windows Start menu.
8. From the CodeWarrior main menu, choose File > Open and choose the S12XHY_SSD_DEMO.mcp file.
9. Click Open. The project window will open.
10. The C code of this demonstration is contained within the main.c file. Double click on the file to open it.
11. From the main menu choose Project > Debug. This compiles the source code, generates an executable file, and downloads it to the demo board.

The stepper stall detection module is set-up via the SSD0_Step function. The SSD configuration registers are modified in sequence depending on the SSD stage, see Figure 3-32 for more details. The module can be used to calibrate a stepper motor, use the visualization tool included in this software package to modify stall detection values on the run, and find the adequate stall value to fit your HW set up, a full CodeWarrior license is required to use this feature. The physical set-up of the motor requires four connections, use the Motor Control Diagram connection (shown in the section *Motor Controller* in the *MC9S12XHY256 Reference Manual*) as a guide.

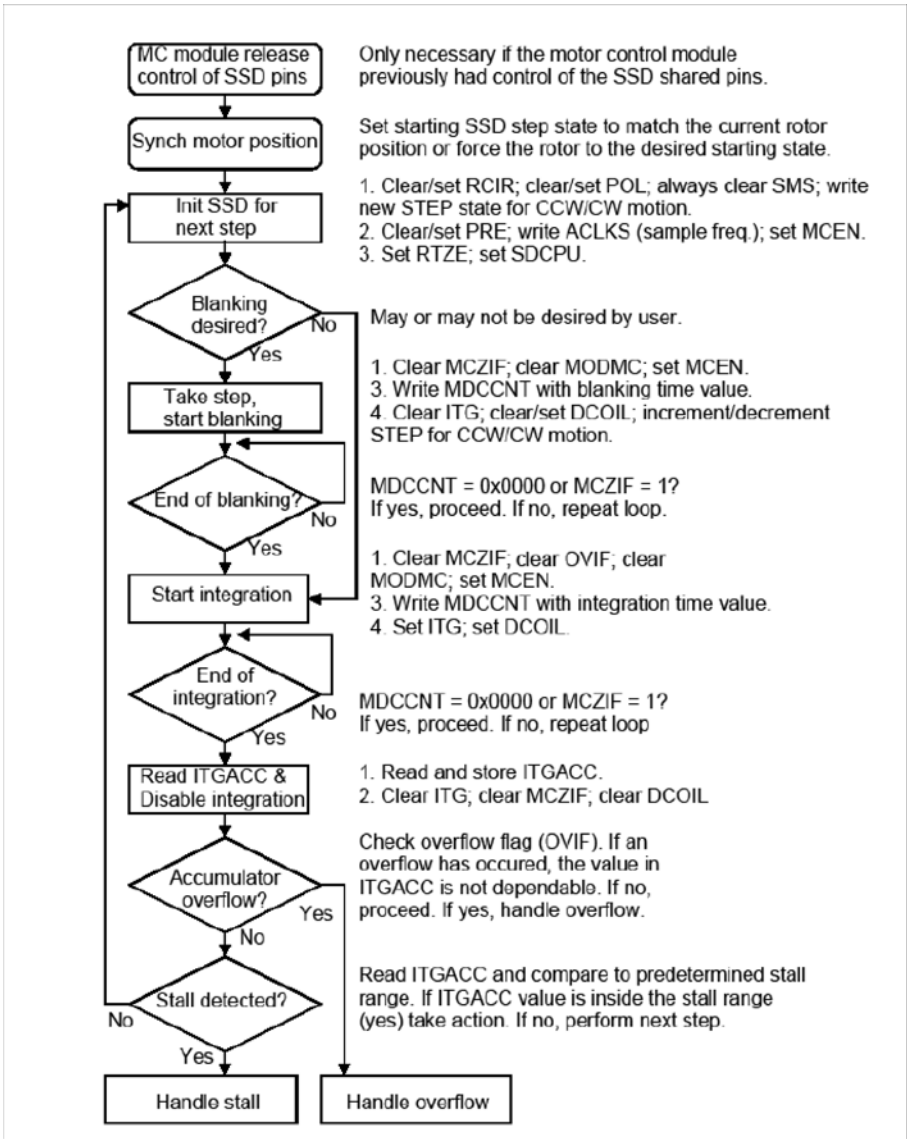


Figure 3-32. SSD software flow chart

NOTE

Be aware that parameters calibration may be required to successfully run this application code, the calibration values depend on the stepper motor type. Use Motor_Calibarion.vtl (Figure 3-33) to help in finding the correct values that meet your Motor parameters and be able to determine stall state.

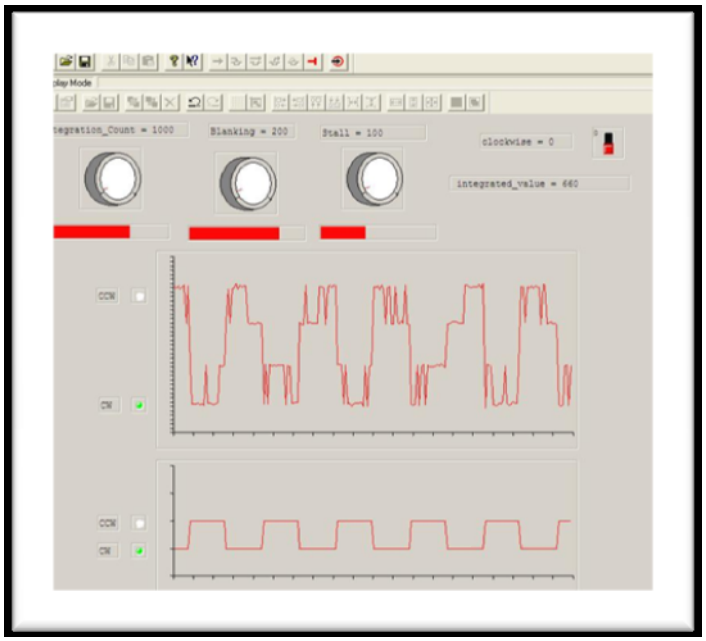


Figure 3-33. Visualization tool application (available, a full CodeWarrior license is required)

3.14.2 Instructions

This project is written to work with a small stepper motor connected to the SSD0 module pins on the DEMO9S12XHY256. It is assumed the stepper motor has a pointer attached to the shaft coming out of the motor. When the project is working, the user will see the pointer rotate in one direction until it bumps into an object (stall detected). When the pointer bumps into an object, the SSD accumulator yields an integration result with a lower magnitude value that is between the stall magnitude and zero. The code may interpret this as a stall, and reverse the rotational direction. The process repeats by beginning another sequence of steps until another stall is detected and the rotational direction is again reversed.

3.14.3 Summary

This demonstration has shown that it is possible to determine stall state using the SSD hardware module of a stepper motor and this basic example can be applied to many types of applications.

4 Conclusion

The S12XHY family of microcontrollers (MCUs) offers the enhanced features of 16-bit performance at a value of 8-bit MCUs. The S12XHY is an extension of the S12HY featuring higher performance and additional modules.

The S12XHY family is ideal for a wide range of central body control applications, such as low-end instrument clusters.

A zip file, AN4236SW.zip, containing the complete CodeWarrior projects for the lab examples accompanies this application note. The file can be downloaded from www.freescale.com.

5 Useful Reference Material

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

Software development tools

- CodeWarrior 5.1 for HCS12(X) Microcontrollers

Application notes

- AN3330 — *Introduction to the Stepper Stall Detector Module*
- AN3219 — *XGATE Library: TN/STN LCD Driver*
- AN3622 — *Comparison of the S12XS CRG Module with S12P CPMU Module*
- AN3034 — *Using MSCAN on the HCS12 Family*
- AN2612 — *PWM Generation Using HCS12 Timer Channels*
- AN2428 — *An Overview of the HCS12 ATD Module*
- AN2883 — *Serial Communication Interface as UART on HCS12 MCUs*
- AN2461 — *Low Power Management using HCS12 and SBC devices*
- AN4201 — *Migrating Applications from S12HY to S12XHY*
- AN4024 — *High Speed Stall Detection on S12HY and S12XHY*

Reference manual

- *MC9S12XHY256 Reference Manual*
- *S12XHY256ACDUG – Automotive Cluster Demo Guide*

Useful links

- LCD_TIPS
- S12HXY256_Cluster_DEMO_VIDEO
- EEPROM_Software_Driver

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

Document Number: AN4236
Rev. 0
11/2010

freescale™
semiconductor