Freescale Semiconductor
Application Note

Document Number: AN4455
Rev. 0, 4/2012

MC9S12G128/A240 Demonstration Lab Training

by: Jamaal Fraser
Automotive and Industrial Solutions Group

1 Introduction

This application note provides the demonstration lab software examples of MC9S12G128/A240 MCUs. The examples show how to configure and use the modules to users getting started with the MC9S12G128/A240 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 the examples. This can be downloaded onto an MC9S12G128 target such as the TWR-S12G128 demo board on which this demonstration lab is based, except for the DAC module. The DAC exists only on the MC9S12GA240 MCU.

Every module except the LIN module of the MC9S12G128/A240 has its own standalone software and is discussed within its own section of this document. For the LIN code, the user will need to get the software that comes with the DEMO9S12PFAME which is available at http://www.freescale.com.

AN4455SW.zip, containing the complete CodeWarrior projects for the lab examples, is available with this application note. The file can be downloaded from http://www.freescale.com.

Contents

2 Setup

NOTE

Before starting any of the module examples in this application note, it is important to complete the Software and Hardware setup as described in the TWR-S12G128 User Guide available on http://www.freescale.com, which accompanies the demonstration board.

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. Ensure that the PWR_SEL (JP5) jumper selects USB power.
2. Ensure that all of the JP1 jumpers are present for LED and push button operation.
3. Ensure that the CAN_EN (JP4) jumper is present.
4. Ensure that the LIN_PWR and MSTR (JP3) jumpers are present.
5. Ensure that the CAN termination (JP6) jumpers are present.
6. Ensure that the COM_EN (JP2) jumpers select the RS-232 position.
7. Connect the demonstration board to the PC via the USB cable.
8. The green power LED on the board must turn on.

3 Demonstration lab examples

3.1 CPMU clocks

This lab example shows how to produce PLL-based bus clocks using the different clock modes of the CPMU module. The example software initializes the PLL to run in default 6.25 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 LED pulse rate and the frequency can be measured by monitoring the ECLK signal (bus clock) on an oscilloscope.

3.1.1 Setup

The following procedure 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 S12G_System_Clocks.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 the file to open it.
5. Ensure the Open Source BDM option is selected as the target.
6. From the main menu, choose Project > Debug. This will compile the source code, generate an executable file, and download it to the demo board.
7. A new debugger environment will open. Once the download to the demo board is complete, close the debugger environment.

3.1.2 Instructions

Follow these instructions to run the lab example:

1. Press the Reset button. The MCU is now running in its default PEI mode with a bus clock frequency of 6.25 MHz.
2. Monitor the ECLK signal on the oscilloscope. ECLK will match the bus clock frequency. Observe the LED1 pulse rate.
3. Press the push button SW2.
4. The MCU is now running in PEI mode with a bus clock frequency of 12.5 MHz.
5. Monitor the ECLK signal on the oscilloscope. ECLK will match the bus clock frequency. Observe the LED2 pulse rate.
6. Press the Reset button and the push button SW3. The MCU is now running in PEE mode with a bus clock frequency of 32 MHz.
7. Monitor the ECLK signal on the oscilloscope. ECLK will match the bus clock frequency. Observe the LED3 pulse rate.
8. Press the Reset button and the push button SW4. The MCU is now running in PBE mode with a bus clock frequency of 4 MHz.
9. Monitor the ECLK signal on the oscilloscope. ECLK will match the bus clock frequency. Observe the LED4 pulse rate.

3.1.3 Summary

The CPMU PLL has three modes:

- PLL Engaged Internal Mode (PEI)
- PLL Engaged External Mode (PEE)
- 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 its source (PEI mode). This is the default clock mode from reset and will produce a bus clock of 6.25 MHz.

3.2 Flash overview

The flash technology module (FTM) contains program flash (P-flash) and EEPROM. P-flash is intended primarily for nonvolatile code storage. EEPROM is used for nonvolatile data storage. 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 (FSTAT) register.
3. Make sure the Command Complete Interrupt Flag is set, or CCIF=1.
4. Launch the appropriate flash commands such as Program, Erase, and Verify, via FCCOBIX and FCCOB registers.
5. Ensure that FSTAT[CCIF]=1.

A flow chart of these steps is shown below:

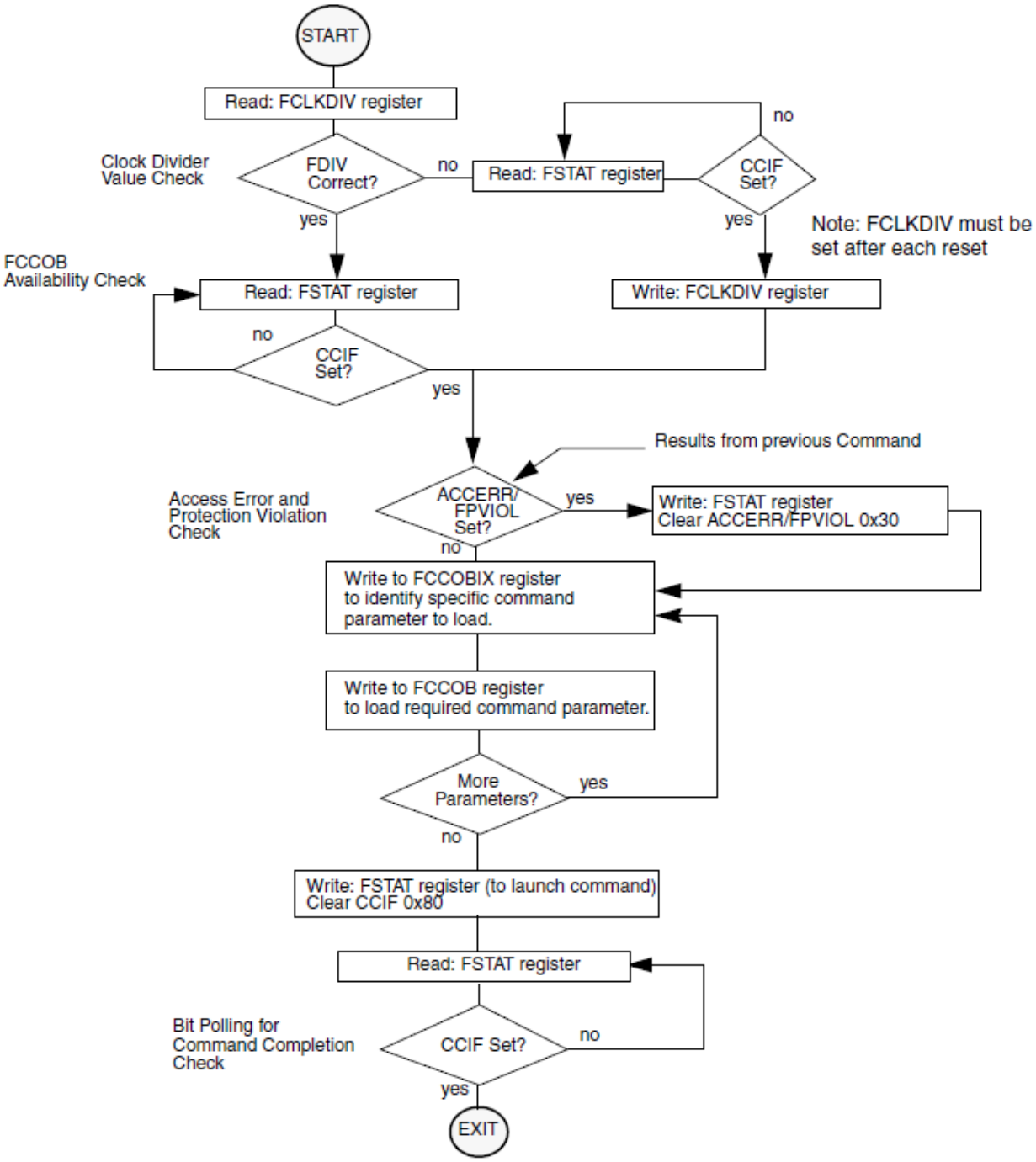

Figure 1. Flash command flow

3.2.1 Code example explanation and walkthrough

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

NOTE

If the security information byte is accidentally erased, that part will be secured and cannot be reprogrammed until it is unsecured.

The file of interest is main.c. This example demonstrates:

- Launching a flash command
- Programming and erasing flash command

This example has been written with a series of user software breakpoints and the user has to just press the Run button.

On startup, the debugger must begin the program in main.c file.

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 on the Flash command given. The flash commands are briefly described in the flash.h header file and within Chapter 26.4 "128 KByte Flash Module," of the MC9S12G Family Reference Manual, available on http://www.freescale.com. Stepping through will take the user to the function below.

```
/*****************************************************************************
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 2. Launch flash command function

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). Only four of the eight S12G128's P-Flash pages have been filled with 0xAAAA; 0x20000, 0x24000, 0x28000, 0x2C000.

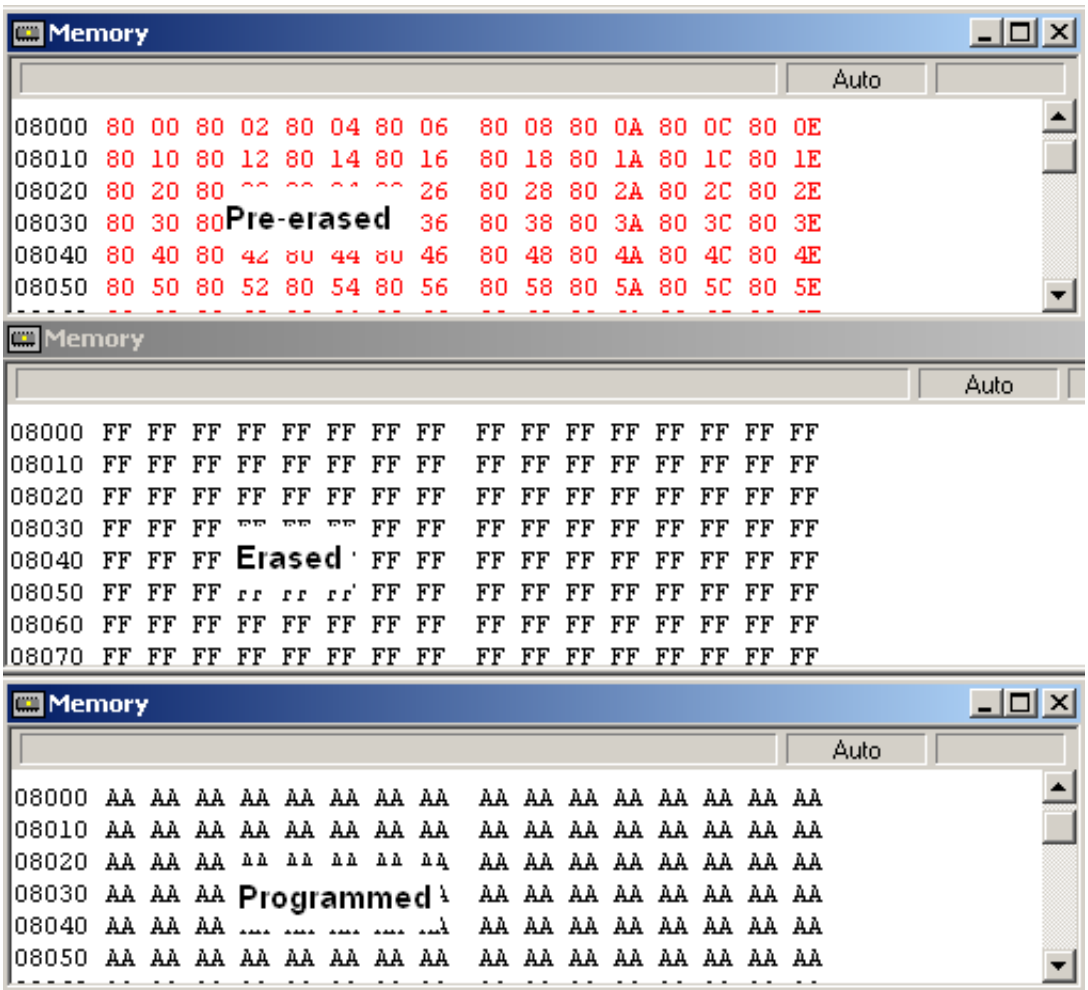

Figure 3. Flash state

At address 0x8000, the P-Flash window will show different content depending on the Program Page Index (PPAGE) register. When the programming has been done, check PPAGE 0x8-0xB by entering the character in the PPAGE register.

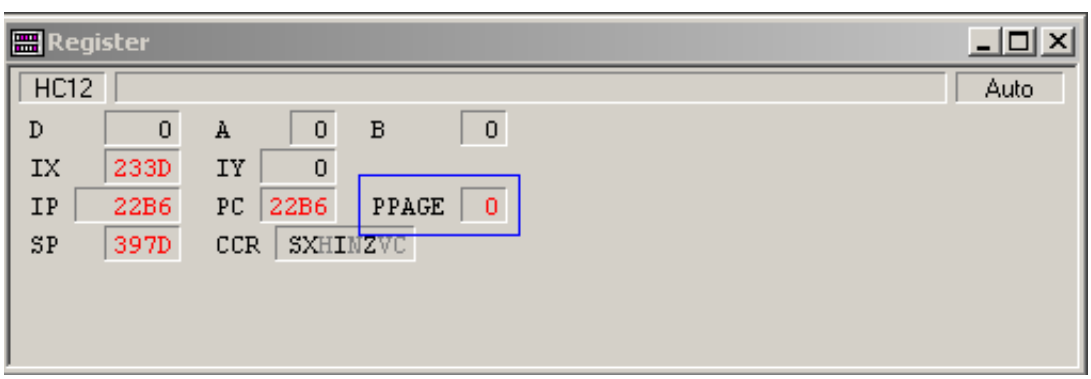

Figure 4. PPAGE register

Breakpoint 3: Launched Program Commands— Address Data

As breakpoint 2, except that the data being written to the P-Flash is different; the data written is the actual addresses of the P-Flash.

Breakpoint 4: EEPROM—Launched Program Commands

The same functions are used but will not perform activity on the EEPROM. The only difference to the LaunchFlash function is that it is now passed through EEPROM addresses and EEPROM commands.

The end of the demonstration is indicated by the LEDs on the EVB being toggled. If LED1 turns on, the test has passed and if LED4 turns on, the test has failed. This example does not reprogram the device to default; this will happen on the next reload of a program by allowing NVM erasing.

3.2.2 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. Any deviation from this may cause errors when working with the P-Flash/EEPROM. 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 LIN communications

An example of how to manage the SCI module to reconstruct a LIN bus is included on the system software CD from Softec Microsystems labeled DEMO9S12PFAME, which is supplied with the DEMO9S12PFAME board, available at http://www.freescale.com.

The software, developed by Softec Microsystems uses the LIN bus to communicate a potentiometer value from master to slave.

The LIN example software can be found on the CD in DEMO9S12PFAME\Docs\CW_Examples\DEMO9S12PFAME\C\LIN.

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 Port AD 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 Port T LEDs.

When the MSCAN module operates 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 S12G_CAN_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 the file to open it.
5. Ensure the Open Source BDM option is selected as the target.
6. From the main menu, choose Project > Debug. This will compile the source code, generate an executable file, and download it to the demo board.

7. A new debugger environment will open. Once 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 the Reset button. The MSCAN demo software will begin the execution.
2. Press SW1–SW4. The corresponding LED for each button turns on.

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 which require reliable communications.

3.5 PWM module

This lab example demonstrates 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 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 S12G_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 the file to open it.
5. Ensure the Open Source BDM option is selected as the target.
6. From the main menu, choose Project > Debug. This will compile the source code, generate an executable file, and download it to the demo board.
7. A new debugger environment will open. Once the download to the demo board is complete, close the debugger environment.

3.5.2 Instructions

Follow these instructions to run the lab example:

1. Press the Reset button. The code starts running on its own. The PWM module must output 50% duty cycle signals on Ports PP0–PP3.
2. Try probing all the 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.

3.6 Low-power modes

In addition to the default Run mode, the MC9S12G has the following three low-power modes:

- Wait/Run mode: 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.
- Pseudo Stop mode: For lower power consumption, Pseudo Stop mode halts the bus clock, but the external oscillator continues to run.
- Stop mode: 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 1. Signals for each mode

Mode	Bus clock	External oscillator
Run	Yes	Yes
Wait	Yes	Yes
Pseudo	No	Yes
Stop	No	No

The change 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.

NOTE

ECLK and EXTAL must be probed separately to avoid adding extra noise to the signals.

3.6.1 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 S12G_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 the file to open it.
5. Ensure the Open Source BDM option is selected as the target.
6. From the main menu, choose Project > Debug. This will compile the source code, generate an executable file, and download it to the demo board.
7. A new debugger environment will open. Once the download to the demo board is complete, close the debugger environment.
8. The bus clock is represented as PB0/ECLK signal (pin 25). The ECLK signal is equivalent to the MCU bus speed and can be monitored by attaching an oscilloscope probe to pin 1 of the J8 header.
9. 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. Press the Reset button. The MCU is now operating in Run mode and LED4 will flash indefinitely indicating the MCU is in Run mode.
2. Monitor the ECLK signal on the oscilloscope, which represents the bus clock. A 32 MHz square wave is observed.
3. Monitor the EXTAL signal on the oscilloscope, which indicates that the crystal oscillator is running. An 8 MHz sine wave is observed.
4. Press SW1. LED1 will flash 20 times and then the MCU will enter the Wait mode. Pressing SW4 causes the MCU to exit Wait mode back into Run mode and LED4 flashing indefinitely.
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 is observed indicating that the external oscillator continues to operate in Wait mode.
7. Press SW2. LED2 will flash 20 times and then the MCU will enter Pseudo Stop mode. Pressing SW4 causes the MCU to exit Wait mode back into Run mode and LED4 flashing indefinitely.
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 is observed indicating that the external oscillator continues to operate in Pseudo Stop mode.
10. Press SW3. LED3 will flash 20 times and then the MCU will enter Stop mode. Pressing SW4 causes the MCU to exit Wait mode back into Run mode and LED4 flashing indefinitely.
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 MC9S12G 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 MC9S12G128 has a global memory of 128 KB. This amount of memory cannot be addressed by the 16-bit MC9S12G128 MCU. 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 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 S12G_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 the file to open it.
5. Ensure the Open Source BDM option is selected as the target.
6. From the main menu, choose Project > Debug. This will compile the source code, generate an executable file, and download it to the demo board.
7. 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 PPAGE register.
3. Start the software by clicking 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 08 and the P-Flash Window shown in the Memory window displays the contents of PPAGE 08.
5. Press the Run button and observe the Memory and Register windows at the next breakpoint.
6. Now the PPAGE register is set to 09 and the P-Flash Window shown in the Memory window displays the contents of PPAGE 09.
7. Press the Run button and observe the Memory and Register windows at the next breakpoint.
8. Now the PPAGE register is set to 0A and the P-Flash window shown in the Memory window displays the contents of PPAGE 0A.
9. Press the Run button and observe the Memory and Register windows at the next breakpoint.
10. Now the PPAGE register is set to 0B and the P-Flash window shown in the Memory window displays the contents of PPAGE 0B.
11. Press the Run button and the software will loop back to return the PPAGE register to 08. The contents of PPAGE 08 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 MC9S12G128 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. Ensure that the RS-232/LIN SEL jumper (JP2) on the TWR-S12G128 demonstration board selects the RS-232 position.
2. Start CodeWarrior by selecting it in the Windows Start menu.
3. From the CodeWarrior main menu, choose File > Open and choose the S12G_ADC_Demo.mcp file.
4. Click Open. The project window opens.
5. The C code of this demonstration is contained within the main.c file. Double-click the file to open it.
6. Ensure that the Open Source BDM option is selected as the target.
7. From the main menu, choose 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. Once the download to the demo board is complete, close the debugger environment.
9. 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. Press the Reset button. The program will send out a “TWR_S12G128 Board Running” message to the terminal window.
2. Press SW3 and the ADC will perform a single 12-bit conversion on PAD0. To perform another conversion, press SW3 again.
3. Vary the conversion result by turning potentiometer RV1 on PAD0 and observe the changes in the terminal window.
4. Press SW1 and the ADC will perform continuous 8-bit conversions on PAD0.
5. Vary the conversion result by turning the potentiometer RV1 and observe the changes in the terminal window. Press SW4 to stop the continuous conversion.
6. Press SW2 and the ADC will perform continuous 12-bit conversions on PAD0 and compare the result to see if it is higher than 0x07FF. Whilst the comparison is true, the LEDs on the demo board will flash.
7. Vary the conversion result by turning potentiometer RV1 on PAD0 and observe the result in the terminal window. Notice how the LEDs only flash when the result is greater than 0x07FF. Press SW4 to stop the continuous conversion.

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 analog source to start the 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.

3.9 DAC module

The DAC module is not available on the S12G128 but is available on the S12GA240. This lab example shows how to use the DAC module in reduced voltage range and full voltage range mode using both the buffered and unbuffered outputs. It uses the ADC to verify the output values of the DAC and errors, if any, are sent to a terminal window via the RS-232 port.

3.9.1 Setup

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

1. Ensure that the RS-232/LIN SEL jumper (J13) on the TWR-S12G240 demonstration board selects the RS-232 position.
2. Place a jumper that connects AMP1 pin (jumper J5 pin 6) to the ADC1 input (jumper J5 pin 2). Place another jumper that connects DAC1 (jumper J10 pin 15) to the ADC2 input (jumper J5 pin 5).
3. Start CodeWarrior by selecting it in the Windows Start menu.
4. From the CodeWarrior main menu, choose File > Open and choose the S12G_DAC_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 the file to open it.
7. Ensure that the Open Source BDM option is selected as the target.
8. From the main menu, choose Project > Debug. This will compile the source code, generate an executable file, and download it to the demo board.
9. A new debugger environment will open. Once the download to the demo board is complete, close the debugger environment.
10. 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.

3.9.2 Instructions

Follow these instructions to run the lab example:

1. Press the Reset button. The program will send out a "TWR-S12G240 Board Running" message to the terminal.
2. Press SW4 and the DAC will perform a buffered output voltage range test and turn on LED4 while the test is running. When completed, a "Test Complete" message will be sent to the terminal and the LED will turn off. Press SW4 to run the test again.
3. Press SW5 and the DAC will perform an unbuffered output voltage range test and turn on LED5 while the test is running. When completed, a "Test Complete" message will be sent to the terminal and the LED will turn off. Press SW4 to run the test again.

3.9.3 Summary

The DAC module is a digital-to-analog converter that works with a resolution of 8 bits and generates an output voltage between VRL and VRH. The unbuffered output voltage can be routed to the external DACU pin. When enabled, the buffered voltage from an internal operational amplifier is routed to the AMP pin.

3.10 Timer module

This lab example shows how to use the Timer module to perform output compare and input capture.

3.10.1 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 S12G_Timer_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 the file to open it.
5. Ensure that the Open Source BDM option is selected as the target.
6. From the main menu, choose Project > Debug. This will compile the source code, generate an executable file, and download it to the demo board.
7. A new debugger environment will open. Once the download to the demo board is complete, close the debugger environment.

3.10.2 Instructions

Follow these instructions to run the lab example:

1. Press the Reset button. The code starts running and the Timer will perform output compares on channel 4 (Port T4) and channel 5 (Port T5), and input capture on channel 6 (Port T6). When a compare match occurs, Ports T4 and T5 will toggle.
2. Ports T4 and T5 are connected to LED1 and LED2, so that these LEDs start toggling.
3. To ensure the input capture is detecting edge transitions, remove the jumper connecting pin 13 to pin 14 of JP1 and the jumper connecting pin 15 to pin 16 of JP1. Place a jumper that connects pin 13 and pin 15 of JP1. This will connect the output of T4 to the input T6 and now, LED4 starts toggling.

3.10.3 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 the events have occurred. It is very important in the scheduling of repetitive actions and contains a variety of special functions, such as pulse accumulation.

3.11 SCI communications

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

3.11.1 Setup

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

1. Ensure that the RS-232/LIN SEL jumper (JP2) on the TWR-S12G128 board selects the RS-232 position.
2. Start CodeWarrior by selecting it in the Windows Start menu.
3. From the CodeWarrior main menu, choose File > Open and choose the S12G_SCI_Demo.mcp file.
4. Click Open. The project window will open.

3.11.2 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 the file to open it.
2. Configure the variable Baud_Rate to 9600 and make sure all other options are disabled.
3. Ensure the Open Source BDM option is selected as the target.
4. From the main menu, choose Project > Debug. This will compile the source code, generate an executable file, and download it to the demo board.
5. A new debugger environment will open.
6. 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.
7. Press F5. The code will begin execution, configuring the SCI to the selected baud rate. Its status can be confirmed on the terminal window.
8. 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.
9. Repeat steps 2 to 8 for baud rates of 19,200, 38,400, and 57,600. Alternatively modify the definition of variable Baud_Rate for a user-configured baud rate.

3.11.3 Summary

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

3.12 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.

Since none of the SPI signals are easily accessed through jumpers on the TWR-S12G128, this example is limited to transmitting data only. When a SPI master transmits data to a SPI slave, data is usually received simultaneously, synchronized by a serial clock.

3.12.1 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 S12G_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. Ensure the Open Source BDM option is selected as the target.
6. From the main menu, choose Project > Debug. This will compile the source code, generate an executable file, and download it to the demo board.
7. A new debugger environment will open. Once the download to the demo board is complete, close the debugger environment.
8. Attach scope probes to PS5, PS6, and PS7.
9. Configure the oscilloscope to trigger on the falling-edge of PS6.

3.12.2 Instructions

Follow these instructions to run the lab example:

1. Press the Reset button. The code will begin execution, configuring the SPI to transmit an incrementing byte of data at a baud rate of 15.625 kbit/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.

3.12.3 Summary

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

4 Conclusion

The MC9S12G-Family is an optimized, automotive, 16-bit microcontroller product line focused on low-cost, high-performance, and low pin-count. This family is intended to bridge between high-end 8-bit microcontrollers and high-performance 16-bit microcontrollers, such as the MC9S12XS-Family. The MC9S12G-Family is targeted at generic automotive applications requiring CAN or LIN/J2602 communication.

Typical examples of these applications include body controllers, occupant detection, door modules, seat controllers, RKE receivers, smart actuators, lighting modules, and smart junction boxes. AN4455SW.zip, containing the complete CodeWarrior projects for the lab examples, is available with this application note. The file can be downloaded from http://www.freescale.com.

5 References

This section presents a list of reference sources available on http://www.freescale.com to gather further information on the following modules used in this application note.

- CMPMU module:
  - AN3622: Comparison of the S12XS CRG Module with the S12P CPMU Module
  - MC9S12GRMV1 Reference Manual
- MSCAN module:
  - AN3034: Using MSCAN on the HCS12 Family
  - MC9S12GRMV1: MC9S12G Family Reference Manual and Data Sheet
- PWM module:
  - MC9S12GRMV1: MC9S12G Family Reference Manual and Data Sheet
- Low-power modes:
  - AN2461: Low Power Management Using HCS12 and SBC devices
  - MC9S12GRMV1: MC9S12G Family Reference Manual and Data Sheet
- MMC module:
  - MC9S12GRMV1: MC9S12G Family Reference Manual and Data Sheet
- ADC module:
  - AN2428: An Overview of the HCS12 ATD Module
  - MC9S12GRMV1: MC9S12G Family Reference Manual and Data Sheet
- DAC module:
  - MC9S12GRMV1: MC9S12G Family Reference Manual and Data Sheet
- Timer module:
  - MC9S12GRMV1: MC9S12G Family Reference Manual and Data Sheet
- SCI module:
  - AN2883: SCI as UART on HCS12 MCUs
  - MC9S12GRMV1: MC9S12G Family Reference Manual and Data Sheet
- SPI module:
  - MC9S12GRMV1: MC9S12G Family Reference Manual and Data Sheet

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor
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

Document Number: AN4455
Rev. 0, 4/2012

freescale semiconductor