Freescale Semiconductor
Application Note

Document Number: AN4418
Rev 1, 11/2013

MC9S12G64 Demonstration Lab Training

by: Jose Manuel Cisneros

Contents

1 Introduction

This document includes several module configuration examples which show how to configure and use MCUs modules to users getting started with the S12G64 device.

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. It can be downloaded onto an MC9S12G64 target such as TWR-S12G64 demo board upon which this demonstration lab is based.

Each module of the S12G64 device has its own standalone software and is discussed within its own section of this document.

A zip file, AN4418SW.zip, contains the complete CodeWarrior project for the lab examples that accompanies this application note. This software zip file can be located and downloaded from http://www.freescale.com

2 Setup

2.1 Tools setup

NOTE

Before starting this lab, it is important for the user to complete the software hardware setup as described in TWRS12G64QSG, *TWR-S12G64 Quick Start Guide*, which accompanies the demonstration board.

2.2 Board setup

The following steps provide a basic configuration for each of the module examples in this document. See Figure 1. See TWR-S12G64 schematic for details on the headers and connectors.

Table 1 shows all jumper options. The default installed jumper settings are shown in shaded boxes.

Table 1. Jumper options

Jumper (by hardware revision)		Option	Setting	Description	
Up to Rev. B	From Rev. C			Valid up to Rev. B	Valid from Rev. C
JP1	JP7	USER SWs, RVs, and LED selection	1–2	Connect PAD4 pin to SW1	Connect PAD4 pin to SW3
			3–4	Connect PAD5 pin to SW2	Connect PAD5 pin to SW4
			5–6	Connect PAD6 pin to SW3	Connect PAD6 pin to SW5
			7–8	Connect PAD7 pin to SW4	Connect PAD7 pin to SW6
			9–10	Connect PAD10 to RV1 POT1	Connect PAD10 to RV1 POT1
			11–12	Connect PAD11 to RV2 POT2	Connect PAD11 to RV2 POT2
			13–14	Connect PP0 pin to LED1	Connect PP0 pin to LED4
			15–16	Connect PP1 pin to LED2	Connect PP1 pin to LED5
			17–18	Connect PP2 pin to LED3	Connect PP2 pin to LED6
			19–20	Connect PP3 pin to LED4	Connect PP3 pin to LED7
JP2	JP6	LIN TX enable	1–3	Routes SCI (TXD0) signal to be output by LIN transceiver, chip (U4) at **J6** and **J7**	Routes SCI (TXD0) signal to be output by LIN transceiver, chip (U4) at **J8** and **J12**
		LIN RX enable	2–4	Routes SCI (RXD0) signal to be output by LIN transceiver, chip (U4) at **J6** and **J7**	Routes SCI (RXD0) signal to be output by LIN transceiver, chip (U4) at **J8** and **J12**
		UART TX enable	3–5	Routes SCI (TXD0) signal to be output by RS232 chip (U3) at **J5**	Routes SCI (TXD0) signal to be output by RS232 chip (U3) at **J11**
		UART RX enable	4–6	Routes SCI (RXD0) signal to be output by RS232 chip (U3) at **J5**	Routes SCI (RXD0) signal to be output by RS232 chip (U3) at **J11**
JP3	JP5	LIN POWER	1–2	12V for LIN Bus	12V for LIN Bus
			3–4	12V for LIN Transceiver	12V for LIN Transceiver
JP4	JP1	CAN PWR	1–2	CAN transceiver power enable	CAN transceiver power enable
JP5	JP4	PWR selector	1–2	Selects the board to be powered from the 3.3V elevator card rail	Selects the board to be powered from the 3.3V elevator card rail

Table continues on the next page...

Table 1. Jumper options (continued)

Jumper (by hardware revision)		Option	Setting	Description	
Up to Rev. B	From Rev. C			Valid up to Rev. B	Valid from Rev. C
			3–4	Selects the board to be powered from the 5V USB connector	Selects the board to be powered from the 5V USB connector
			5–6	External source selected as power source	External source selected as power source
JP6	JP3	CAN (H/L)	1–2	SPLIT termination to CAN—H line	SPLIT termination to CAN—H line
			3–4	SPLIT termination to CAN—L line	SPLIT termination to CAN—L line

NOTE

Any deviation from this basic configuration or any specific requirements for a module will be outlined in the corresponding module chapter.

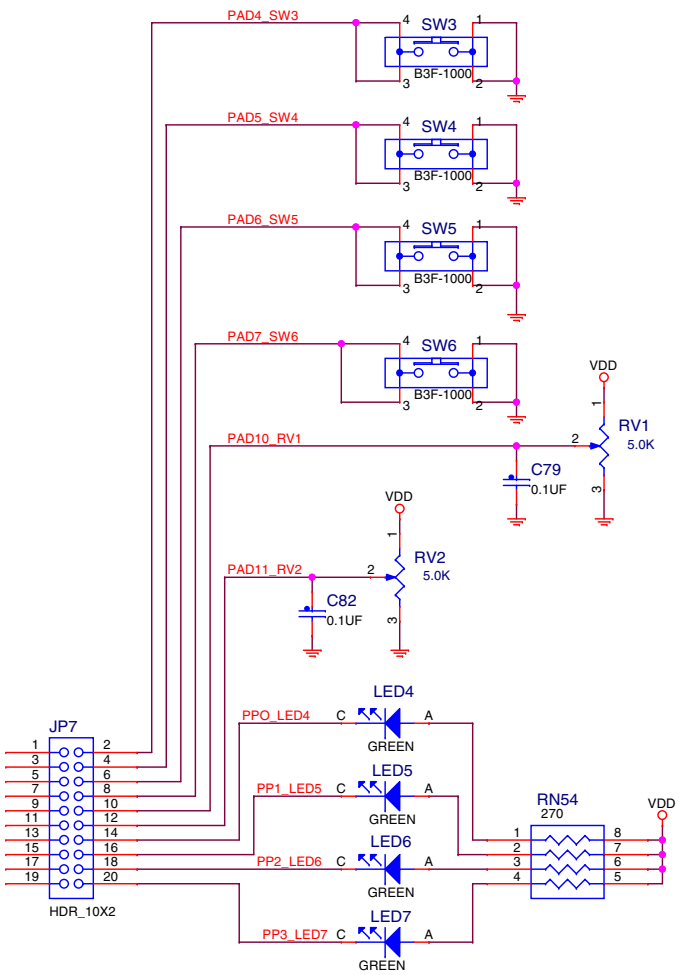

Figure 1. Basic configuration

- Ensure all jumpers at JP7 jumper connector are installed to enable the light-emitting diodes LED4–LED7, buttons SW3–SW6, and the potentiometers RV1–RV2.
- Ensure JP6 has two jumpers to shunt pins 3–5 and 4–6 to enable SCI communications.
- Ensure that the DB9 cable adapter plugged to the jumper connector J11 has been connected. This cable is included with the TWR-S12G64 demo kit.
- Ensure that LIN, and CAN_EN jumper are installed.
- Ensure that the POWER SEL jumper selects the USB position.
- Connect the demonstration board to the PC via the USB cable.
- The green POWER LED1 on the board as well as LED2 and LED3 must be turned on.

3 Demonstration lab examples

The following sections exercise the S12G64 modules in an individual way, so the user can use them as an example to follow for his own application.

3.1 Clocks and power management unit (CPMU)

This lab example shows how to produce bus clocks based on the phase-locked loop (PLL) using the CPMU module in its different modes of operation. The example software initializes the PLL to run in:

- Run mode at 8 MHz bus
- Pseudo Stop mode 16 MHz bus
- Stop mode 16 MHz bus clock

The changes in the bus clock can be observed via the pulse rate of LED4 and the frequency can be measured by monitoring the ECLK signal (bus clock) on an oscilloscope at pin 11 of J9 or, pin 59 of the MCU.

3.1.1 Setup

The following steps must be followed before running the CPMU module:

1. Open CodeWarrior by selecting it in the Windows Start menu.
2. From the CodeWarrior main menu, choose File > Open and select S12G64_CPMU_DEMO.mcp file.
3. Click Open. The project window then opens.
4. C code of this demonstration is contained in the main.c file.
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 TWR-S12G64 board with the following configuration. Open a terminal window on the PC with this configuration.
   - Baud rate = 9600
   - Data bits = 8
   - Parity = N
   - Stop bits = 1
   - Flow control = none
8. The bus clock speed is represented on pin 59 PS7/ECLK. To accomplish this, the ECLKCTL register that controls the availability of a free-running clock on the ECLK pin, must be configured accordingly. See the device reference manual available at http://www.freescale.com for further information. The ECLK signal is equivalent to the MCU bus speed and can be monitored by connecting an oscilloscope probe to pin 11 of J9 header.
9. A jump wire is required to set IRQ pin to VDD. Pin 15 of J10 is PT1/IOC1/IRQ (MCU pin 31), VDD is located at pin 3 of J13, and GND is located at pins 7, 9, and 11 of J13. This interrupt will be used in Pseudo Stop mode to wake up the MCU.

3.1.2 Instructions

Follow these instructions to run the lab example.

- **Run 8 MHz test:**
  a. Press the Reset button. The MCU is now in Run mode with a bus clock frequency of 8 MHz.
  b. Monitor the ECLK signal on the oscilloscope. The ECLK signal at pin 59 matches the bus clock frequency. Observe the LED4 pulse rate, and examine the PLL configuration on the terminal window.
- **Run 25 MHz test:**
  a. Press the Reset button while pressing SW3, but make sure to release the Reset button before releasing SW3. The MCU is now in Run mode with a bus clock frequency of 25 MHz.
  b. Monitor the ECLK signal on the oscilloscope. The ECLK signal at pin 59 matches the bus clock frequency, observe LED4 pulse rate, and examine the PLL configuration on the terminal window.
- **Pseudo Stop Mode:**
  a. Be sure to follow Step 9 of Setup.
  b. Press the Reset button while pressing SW4, but make sure to release the Reset button before releasing SW4. The MCU is now running in Pseudo Stop mode with a bus clock frequency of 16 MHz.
  c. Monitor the ECLK signal on the oscilloscope. The ECLK signal at pin 59 matches the bus clock frequency. Observe LED4 pulse rate, and examine the PLL configuration on the terminal window.
  d. Press the SW6 button for at least 2 seconds. The clock pulses will disappear and the MCU enters to Pseudo Stop mode. To wake up the microcontroller, connect the IRQ pin of the MCU to GND using the jump wire. See step 9 of Setup. The MCU will wake up and LED7 will start toggling to show that MCU is out of Pseudo Stop mode.
- **Stop Mode test:**
  a. Press the Reset button while pressing SW5, but make sure to release the Reset button before releasing SW5. The MCU is now running and prepared for Stop mode with a bus clock frequency of 16 MHz.
  b. Monitor the ECLK signal on the oscilloscope. The ECLK signal at pin 59 matches the bus clock frequency. Observe LED4 pulse rate and examine the PLL configuration on the terminal window.
  c. Press the SW6 button for at least 2 seconds. The clock pulses disappear, and the MCU enters the Stop mode. The bus clock will stop, and the MCU will stay in Stop mode until the Reset button is pressed.

3.1.3 Summary

The three CPMU modes namely Run, Pseudo Stop and Stop, were exemplified with this lab. After reset, the MCU starts from the internal 1 MHz RC oscillator and by using the PLL, bus frequency can be trimmed to different value than the source clock.

For more details, see the device reference manual located at http://www.freescale.com.

3.2 Analog-to-digital converter (ADC) module

This lab example shows how to perform 10-bit automatic compare and continuous 8-bit conversions. The ADC conversion results are output on a terminal window via the RS-232 port.

3.2.1 Setup

The following steps must be followed before running the ADC module:

1. Open CodeWarrior by selecting it in the Windows Start menu.
2. From the CodeWarrior main menu, choose File > Open and select S12G64_ADC_DEMO.mcp file.

3. Click Open. The project window then opens.
4. C code of this demonstration is contained in the main.c file.
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 ADC conversion result is sent to the RS-232 port with the following configuration. Open a terminal window on the PC with this configuration:
   - Baud rate = 9600
   - Data bits = 8
   - Parity = N
   - Stop bits = 1
   - Flow control = Hardware

3.2.2 Instructions

Follow these instructions to run the lab example:

1. Press the Reset button. The ADC will perform a single 10-bit conversion on AN10 channel (PAD10, pin 38 of S12G64). To perform another conversion, press SW6. Vary RV1, then press SW6 and observe the changes in the terminal window.
2. Press the Reset button while pressing SW3, but make sure to release the Reset button before releasing SW3. The ADC will perform continuous 8-bit conversions on AN10 channel (PAD10, pin 38 of S12G64).
3. Turn RV1 and observe the changes in the terminal window.
4. Press the Reset button while pressing SW4, but make sure to release the Reset button before releasing SW4. The ADC will perform continuous 10-bit conversions on AN10 channel (PAD10, pin 38 of S12G64) and compare the result to see if it is higher than 0x01FF. Whilst the comparison is true, LED4 on the demo board will flash.
5. Turn RV1 and observe the result in the terminal window. Notice how LED4 flashes only when the result is greater than 0x01FF.

3.2.3 Summary

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

3.3 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 TWR-S12G64 board are used to indicate port toggling due to an output compare match, or a successful input capture.

3.3.1 Setup

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

1. Ensure that RV1 enable jumper on JP7 has been removed.
2. Connect the potentiometer output (pin 10 on JP7) to Port T3 (pin 13 on J12). This will allow the potentiometer RV1 to provide stimulus to the input capture function on the Timer 1 channel IOC3.

3. Start CodeWarrior by selecting it in the Microsoft Windows Start menu.
4. From the CodeWarrior main menu, choose File > Open and select S12G64_TIM_DEMO.mcp file.
5. Click Open. The project window opens.
6. The C code of this demonstration is contained within the main.c file. Double-click the file to open it.
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 opens. After the download to the demo board is completed, close the debugger environment.

3.3.2 Instructions

In order to drive LEDs directly from Timer output channels, the software reroutes the Timer 0 channels IOC0 and IOC1 from Port T0–T1 to Port P0–P1. In addition, to allow a connection to an input capture channel, the software also reroutes Timer 1 channel IOC3 from Port T3 to Port P3. The rerouting also gives the user access to the signals at J10 header.

Follow these instructions to run the lab example:

1. Press the Reset button. The code must run and reroute the Timer channels.
2. Timer 0 will perform output compares on channels IOC0 (Port P0 rerouted from Port T0) and IOC1 (Port P1 rerouted from Port T1). When a compare match occurs on IOC0, Port P0 will toggle. When a match compare occurs on IOC1, Port P1 will toggle.
3. Timer 1 will perform input capture on both rising and falling edges on channel IOC3 (Port P3 rerouted from Port T3).
4. Use an oscilloscope to view the toggling Timer 0 channels IOC0 and IOC1 on Port T0–T1 pins (J10 header pins 16 and 17). In case an oscilloscope is not available, the LEDs associated with
5. Port P0 (LED4) and Port P1 (LED5) toggle in sync with the Timer channels.
6. Use the potentiometer RV1 to create rail-to-rail rising and falling edges on Timer 1 channel IOC3.
7. To ensure the input capture is detecting edge transitions, observe how LED7 toggles with each rising or falling edge caused by RV1.

3.3.3 Summary

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

3.4 SCI module

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

3.4.1 Setup

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

1. Ensure that both the jumpers on JP6 are installed connecting the pins 3–5 and 4–6.
2. Start CodeWarrior by selecting it in the Microsoft Windows Start menu.
3. From the CodeWarrior main menu, choose File > Open and select S12G64_SCI_DEMO.mcp file.
4. Click Open. The project window opens.

3.4.2 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 baud rate to 9600 and make sure all other options are disabled.
3. From the main menu, choose Project > Debug. This will compile the source code, generate an executable file, and download it to the demo board.
4. A new debugger environment opens.
5. The software uses the RS-232 port to interact with the user. Open a terminal window with the following configuration to see the RS-232 data:
   - Baud rate = 9600
   - Data bits = 8
   - Parity = N
   - Stop bits = 1
   - Flow control = Hardware
6. Press F5. The code begins execution and configures 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. Select some options and observe the SCI register configurations.
8. The user can select options B–F in the terminal window to configure different baud rate in run time. Follow the instructions indicated to change the baud from the menu.

3.4.3 Summary

The SCI module can be used to communicate with peripheral devices or other MCUs. This module can also be configured to accomplish several bit rate speeds. For details on the special setting of this module, see the S12G64 reference manual available at http://www.freescale.com.

3.5 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.

Even when there are two SPI modules available on the TWR-S12G64, this example is limited to transmitting data only through the SPI0.

3.5.1 Setup

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

1. Start CodeWarrior by selecting it in the Microsoft Windows Start menu.
2. From the CodeWarrior main menu, choose File > Open and select S12G64_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 the file to open it.
5. From the main menu, choose Project > Debug. This will compile the source code, generate an executable file, and download it to the demo board.
6. A new debugger environment will open. After the download to the demo board is completed, close the debugger environment.
7. Attach scope probes to signals PS5/MOSI0, PS6/SCK0, and PS7/SS0 on pins 9, 10, and 11 of header J9.
8. Configure the oscilloscope to trigger on the falling-edge of PS7.

3.5.2 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 12 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. In Figure 2, the Slave Select signal is represented in orange, Data Transmitted in purple, and Serial Clock in blue color.

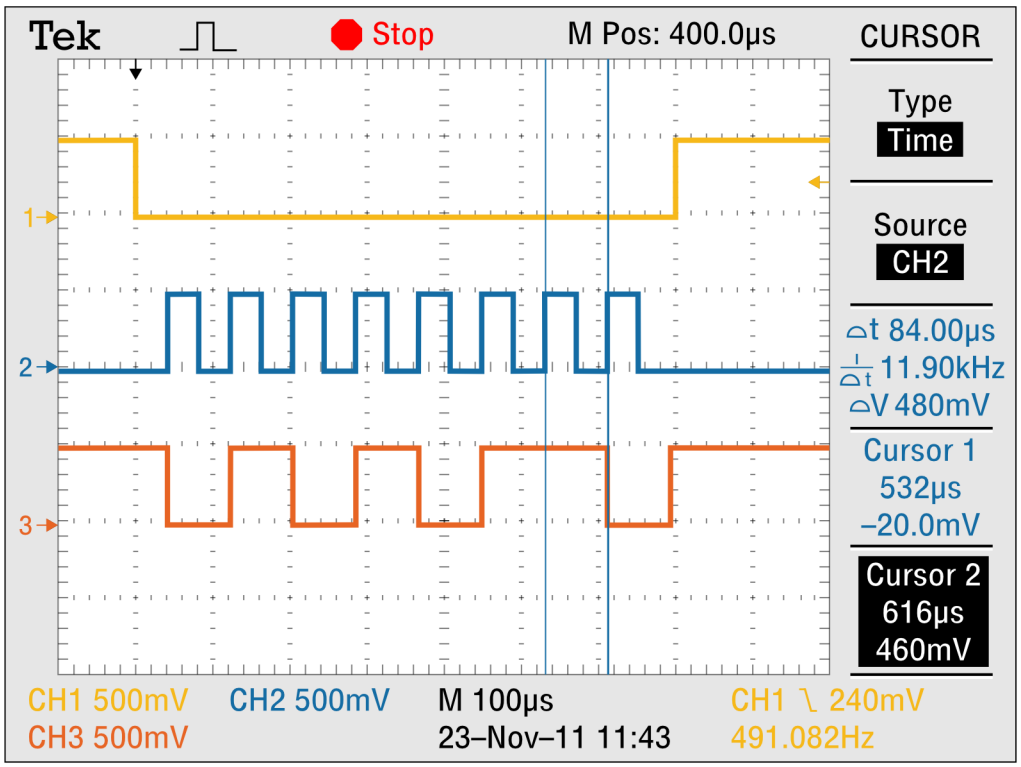

Figure 2. SPI protocol signals

3.5.3 Summary

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

3.6 PWM module

This lab provides an example of how to setup four out of six PWM channels available on the S12G64 device. This demo code configures the PWM module as follows:

1. Create a 50% duty cycle output with different polarity and alignment settings on each of the channels, PWMCH0–PWMCH3.
2. Configure these four channels to operate from Timer A. Timer B, SA, and SB are also available, so each channel can selectively use different time clocks. See the chapter on PWM in the S12G64 reference manual on http://www.freescale.com for more details.
3. Configure the PWM channels as follows:

- PWM0: Left-Aligned, Polarity = 0 at 100 KHz
- PWM1: Left-Aligned, Polarity = 1 at 100 KHz
- PWM2: Center-Aligned, Polarity = 0 at 50 KHz
- PWM3: Center-Aligned, Polarity = 1 at 50 KHz

NOTE

All the four channels above are referenced to the same clock source, however, as PWM2 and PWM3 are center-aligned, the period of these two channels is double than PWM0 and PWM1.

3.6.1 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 select S12G64_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. From the main menu, choose Project > Debug. This will compile the source code, generate an executable file, and download it to the demo board.
6. A new debugger environment will open. After the download to the demo board is completed, close the debugger environment.

3.6.2 Instructions

The PWM signals PWM0–PWM3 can be accessed at the pins 17–20 of J10 header.

Follow these instructions to run the lab example:

1. Press the Reset button. 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 noticing the difference in channel settings such as center alignment and polarity to be more apparent.

3.6.3 Summary

The PWM is a common module on many microcontrollers. It often finds use in applications that need to vary frequency or intensity, such as in light control, motor speed control, or other more complex circuits that require a regulated output.

3.7 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, SW3–SW6, 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 transmitter (Tx) and Receiver (Rx) pins are held high.

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 select S12G64_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 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.

3.7.2 Instructions

Follow these instructions to run the lab example:

1. Press the Reset button. The MSCAN demo software begins execution.
2. Press various combinations of SW3, SW4, SW5, and SW6 switches. The LEDs must match their configuration.

3.7.3 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 suits a wide variety of uses that require reliable communications.

3.8 MMC

This demo code is divided into two demonstration codes:

- The first part shows how the MMC module is used to page through the memory and demonstrates how direct memory access can be used.
- The second part demonstrates how invalid memory access causes a reset event and how the user can acknowledge that illegal access happened and take failsafe action.

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 select S12G64_MMC_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 the file to open it.
5. Make sure that the MMC_Test variable is equal to the test the user wants to run. It could be either MMC_VALID_ACCESS or, MMC_INVALID_ACCESS.
6. From the main menu, choose Project > Debug. This compiles the source code, generates an executable file, and downloads it to the demo board.
7. A new debugger environment will open.

3.8.2 Instructions

See the desired section of this demo code depending on what selection has been made in step 5 of Setup.

- **MMC VALID ACCESS**

a. After the code has been downloaded to TWR-S12G64 board, click Run and the debugger will stop after the BGND instruction is reached. In Figure 3, in the debugger, the Memory:1 window shows the current page content starting from 0x8000 and at Register window, the PPAGE register shows 0x0C.

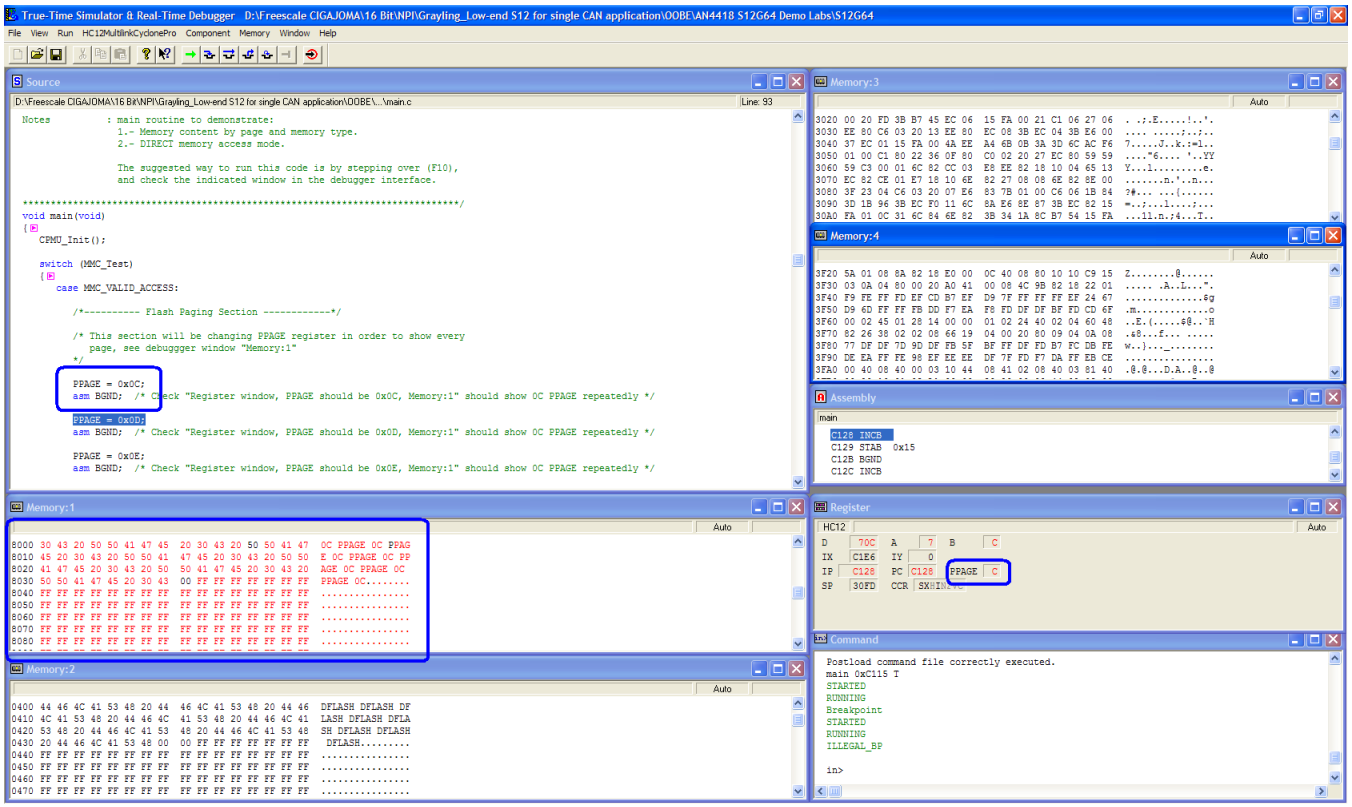

Figure 3. Page 0C shown on PPAGE

b. The same procedure is repeated for pages 0x0D, 0x0E, and 0x0F.

S12G64 device has four pages of 16 KB each, however the content on pages 0x0D and 0x0F can be accessed at addresses 0x4000–0x7FFF and 0xC000–0xFFF respectively without requiring the PPAGE register; this means that data stored on page 0x0D can be accessed at address 0x8000–0xBFFF, if page 0x0D is used, and at address 0x4000–0x7FFF directly. The content on page 0x0F can be accessed at address 0xC000–0xFFFF. See Figure 4 and Table 2 as reference.

c.

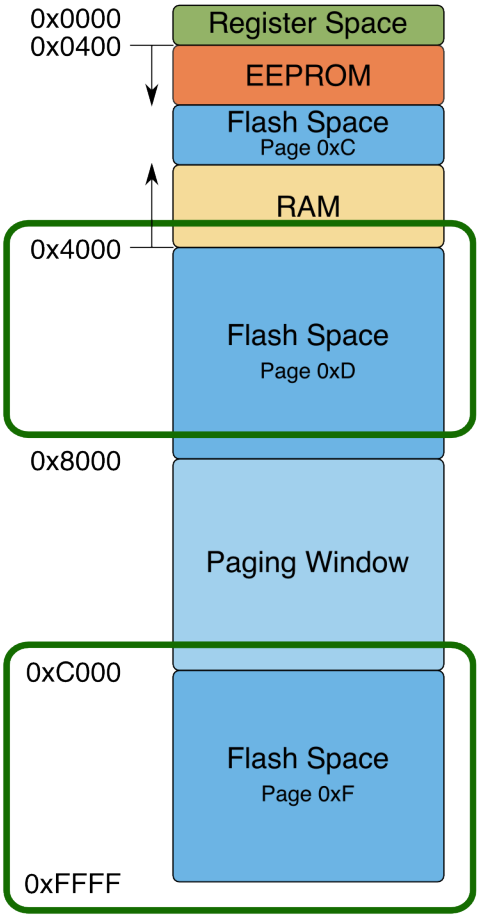

Figure 4. S12G family memory map

Table 2. Local memory map addresses

Memory	Address	Block size (KB)	Pages
EEPROM	0x400 to 0xBFF	2	--
RAM	0x3000 to 0x3FFF	4	--
FLASH	0x8000 to 0xBFFF	16	4
Note: S12G64 has two banks of flash memory that are mirrored with two pages at address 0x8000–BFFF, bank at page 0x0D (16 KB from 0x4000–0x7FFF, and bank 0x0F (16 KB from 0xC000–0xFFFF).			

d. The emulated EEPROM content can be seen at the Memory: 2nd window in the debugger at address 0x0400; S12G64 includes 2 KB of D-Flash and does not require paging for this reason. The total D-Flash space can be accessed at any time from address 0x0400 to 0xBFF. D-Flash string has been stored at programming time to address 0x400–0x437.

- **Direct Memory Addressing**

  This section demonstrates how direct memory addressing is used.

  The user can step over this section of assembly code in debugger to verify how the registers involved with the test are getting updated.
  - The first test is used to show access to RAM by setting Direct Mode register to 30 and using a load offset of 0x20 to have an effective load address of 0x3020. See Y index register value after LDY assembly instruction is used.

- The user must be careful while trying to access memory outside the 256 direct boundary reach. In this second example of direct memory addressing, it is required to set Direct register value to 0x00. So the PPAGE register at address 0x15 can be set to a new value, in this case, 0x0C. For this case, more instructions are required to configure the Direct mode register to address from the desired 0x8000 address page 0x0C.
- The third example of direct memory address loads data from EEPROM at address 0x0400.

- **MMC INVALID ACCESS**

  This section demonstrates the functionality of the invalid memory access for the S12G devices.

  The test occurs as follows:

  a. Load the code compiled with MMC_Test = MMC_INVALID_ACCESS.
  b. Using the debugger, set a breakpoint at asm JMP $A000 and verify address 0x3F20 is cleared. The software verifies whether any illegal access is used right after reset by checking ILAF. If:
     - ILAF = 1; Fault occurred.
     - ILAF = 0; No fault occurred.
  c. If ILAF = 0, meaning that no fault occurred, then the illegal access will be executed by forcing the program counter to read from unimplemented address 0xA000, causing an illegal memory access. Click Run and the debugger must reach the break point.
  d. Click Step and the MCU will reset and ILAF will be equal to 1. To show that the fault has occurred, LED4–LED7 will start toggling at the same time, and the debugger will be disconnected after the reset event.
  e. Close the debugger and while LED4–LED7 are toggling, press the Reset button and only LED4 is toggling now. This is the signal that indicates ILAF has been cleared and now the MCU is running fine even if the Reset button is pressed again.
  f. Since the code is prepared to keep control variables even after reset, the user needs to power off the TWR-S12G64 by turning off SW2, to repeat the test and repeat the process from Step 5.

3.8.3 Summary

The MMC module performs several functions:

- Controls the proper memory access and serves as an arbitrator to the different modules while trying to access MCU resources
- Provides different ways of addressing like the direct mode, which makes memory access very efficient in several cases
- Provides the mode control of the MCU, which makes it possible to access different resources depending on the current run mode to be Special Single (SS) chip or, Normal Single (NS) chip.
- Checks the illegal memory access and protects the user application from unimplemented memory access.

3.9 Analog comparator (ACMP)

The ACMP module is included only in the S12GN16, S12GN32, S12GN48, S12G48, and S12G64 devices. This module provides a circuit for comparing two analog inputs. This module is particularly useful in applications where a change in an analog signal over an specific set point needs to be detected, like in a high-temperature protection application or, a battery overvoltage.

3.9.1 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 select S12G64_ACMP_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 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.

3.9.2 Instructions

Follow these instructions to run the lab example:

1. Click Run in the debugger window.
2. Verify LED4 status.
3. If LED4 is on, then turn off RV1.
4. LED4 will turn on again when RV1 voltage is greater than RV2 voltage.
5. Try with different set points of RV2. Then use RV1 to make ACMP0 transition from high to low and cause LED4 to be on.

3.9.3 Summary

The ACMP module is useful in motor control applications where a quick action requires to be taken after two voltages are compared. S12G64's ACMP interrupts complement this functionality and provide the flexibility to configure interrupts at rising, falling, or rising and falling edges of the comparator output.

Another complementary feature of this ACMP module is that the comparator output signal can be accessed at an output pin as well as it can be used to trigger an internal timer of the MCU.

4 Useful reference material

The following material is available at http://www.freescale.com

- Software development tools
  - CodeWarrior 5.1 for HCS12(X) Microcontrollers.
- Application notes
  - AN4302: Introduction to the S12G Family EEPROM
  - AN4320: Introduction to the 16-bit Tower Boards Using the MC9S12GN32
  - AN4303: Light control and diagnostics using a MC9S12GN32 MCU
  - 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
- Reference Manual
  - MC9S12GRM Reference Manual
  - S12GFS S12G Family Fact Sheet.

5 Revision history

Revision	Date	Substantive changes
0	12/2011	Initial release

Table continues on the next page...

Revision history

Revision	Date	Substantive changes
1	11/2013	• Included Table 1 • Updated Figure 1 • Revised document text to use Rev. C jumper, switch, and LED numbers as specified in Table 1 (was Rev. B and prior numbers). For example, changed JP1 to JP7, and SW1 to SW3 throughout.

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number AN4418
Revision 1, 11/2013