# Dragon12-Light Trainer

For Freescale HCS12 microcontroller family

## User's Manual for Rev. D board

Version 1.03

# Table OF Contents

## Note: For users who will use CodeWarrior IDE with serial monitor:

This manual is written for the board that is pre-installed with bootloader and D-Bug12 monitor. If you ordered the board with Freescale serial monitor for CodeWarrior, the board would be pre-installed with the serial monitor and a factory test program. The software installation on the page 12 is not needed. Once the serial monitor is installed, the board will not work with AsmIDE or other terminal emulation programs.

The state of the left switch of the 2-position DIP switch (SW7) is tested by the serial monitor for selecting RUN or LOAD mode during power up or reset, and the 8 port B LED indicators will flash sequentially from right to left and the speaker will chirp once to indicate that the serial monitor is functioning. If the left switch is placed in the "LOAD" mode (the "low" position) the monitor will wait for a command from a PC. If the left switch is placed in the "RUN" mode (the "up" position) the port B LED indicators will flash sequentially again from left to right to indicate that the program execution is diverted to the user code (the factory test program or your code).

The left DIP switch of the SW7 has been set in the “up” position as a factory default setting for running the test program.

The CodeWarrior communicates with Freescale serial monitor only in the LOAD mode and so in order to interface with the CodeWarrior you have to place the left switch in the “low “position”. The port B LED indicators will flash sequentially from right to left and the speaker will chirp once when the board is powered up.

If your board does not communicate with CodeWarrior, the first thing you should check is that if the left DIP switch of the SW7 is in the LOAD mode (in the "low" position).

# Chapter 1. Introduction

## 1.1 Welcome

Thank you very much for purchasing our Dragon12_light_d trainer. The Dragon12_light_d trainer is a low-cost, feature-packed training board for the Freescale HCS12 microcontroller family. It incorporates many on-board peripherals that make this board a popular trainer in universities around the world.

For engineers, it is a convenient prototype system suitable for designers who want to rapidly develop and prototype HCS12 applications. For students, it can not only to be used as a general trainer for freshman and sophomore students, but also as a versatile platform for senior projects as well. The new features of the Dragon12_light_d board create a new potential for students at every level.

The Dragon12_light_d trainer kit comes with the following items:

1. Dragon12_light_d board with solderless breadboard.
2. Software and User manual downloadable from our web site
3. A USB mini-B cable

If you miss any part of the kit, please contact trainer4edu@gmail.com or call 630 913-1440 for help.

The new Dragon12_light_d board is fully backward compatible to the Dragon12-Plus2 board. Most programs written for the Dragon12-Plus2 board will run on the new Dragon12_light_d board without modifications.

Please carefully examine the default jumper settings before turning on the board:

1. The J40 should have a jumper to enable port B LEDs.
2. The J26 should have a jumper installed on the two up pins labeled with “PT5”, so the speaker will be driven by PT5. The speaker can be driven by timer (PT5) or PWM (PP5) or DAC. It defaults for PT5. Without a jumper installed on J26 the speaker won’t sound.
3. The J25 should have a jumper on the left 2 pins for testing your small motor with USB power. If you need external power supply for your motor, change the jumper to two right pins.

## 1.2 **MC9S12DG256 features and memory map:**

The Dragon12_light_d board comes with the MC9S12DG256CVPE installed. The MC9S12DG256 is a replacement for the MC9S12DP256 since the latter has been discontinued by Freescale. The only difference between DG256 and DP256 is the number of CAN ports. The DG256 has 2 CAN ports, but the DP256 has 5 CAN ports. Other than the different number of CAN ports these two microcontrollers have the same features. If you don't use more than 2 CAN ports these two chips are identical and **all datasheets and manuals** for the DP256 can be used for the DG256.

The MC9S12DG256 microcontroller consists of a powerful 16-bit CPU (central processing unit), 256K bytes of flash memory, 12K bytes of RAM, 4K bytes of EEPROM and many on-chip peripherals.

The main features of the MC9S12DG256 are listed below:

- Powerful 16-bit CPU
- 256K bytes of flash memory
- 12K bytes of RAM
- 4K bytes of EEPROM
- SCI ports
- SPI ports
- CAN 2.0 ports
- $I^2C$ interface
- 8-ch 16-bit timers
- 8-ch 8-bit or 4-ch 16 bit PWM
- 16-channel 10-bit A/D converter
- Fast 25 MHz bus speed via on-chip Phase Lock Loop
- BDM for in-circuit programming and debugging
- 112-pin LQFP package offers up to 91 I/O in a small footprint

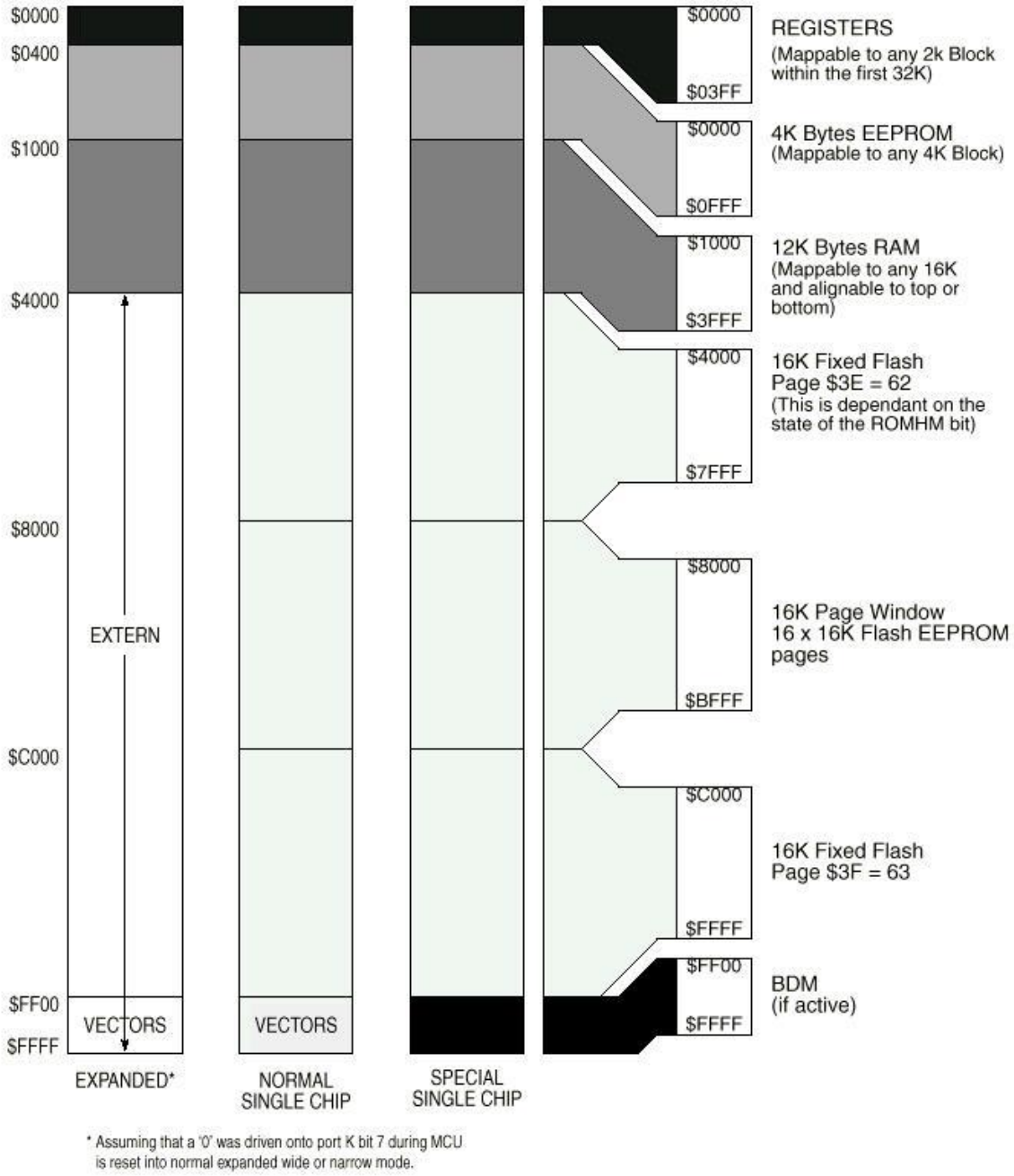



**Fig 1-1: MC9S12DG256 Memory map**

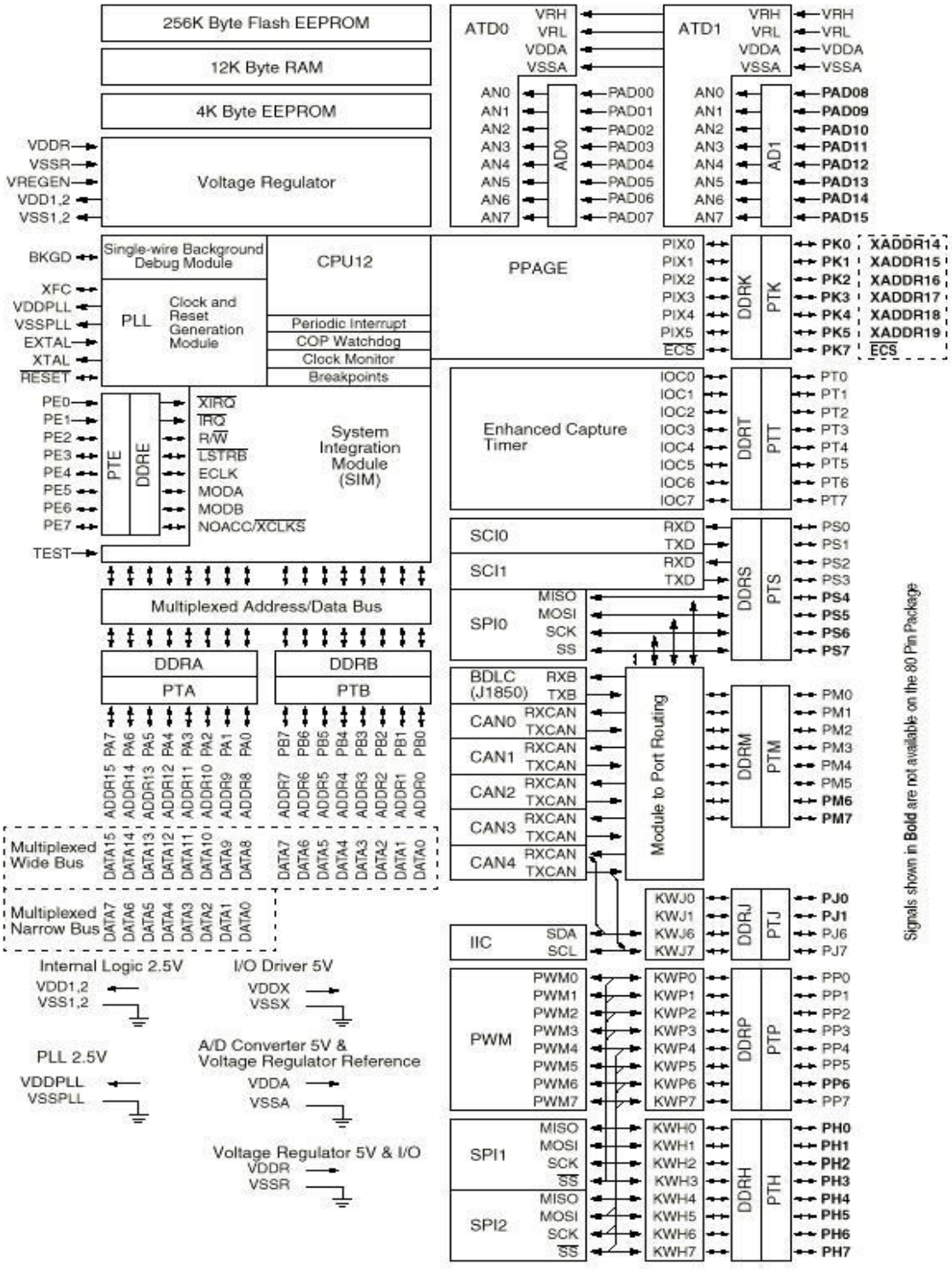



**Fig 1-2: MC9S12DG256 MCU block diagram**

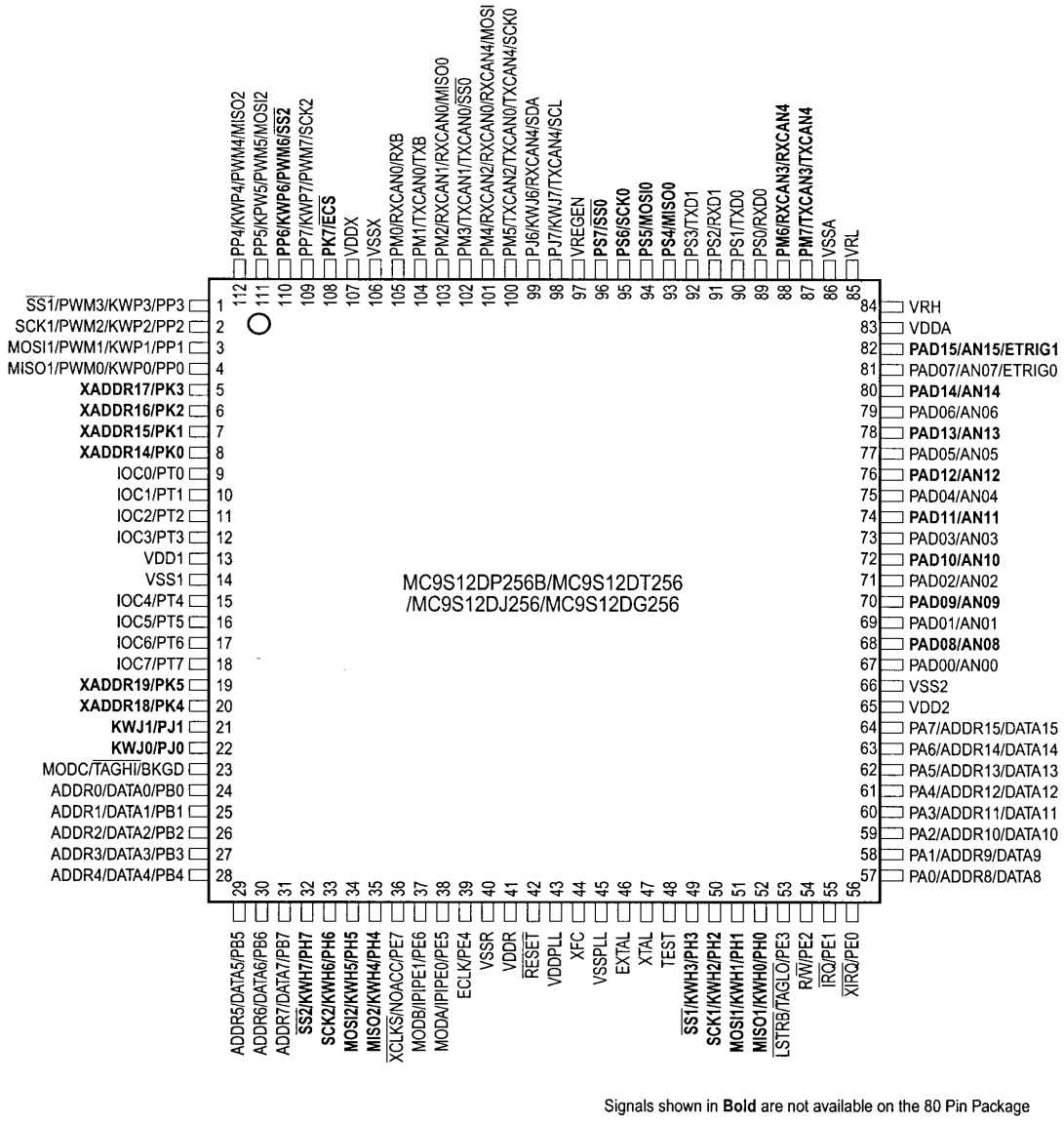



**Fig 1-3: MC9S12DG256 MCU pin assignments**

## 1.3 **On-board hardware features:**

The Dragon12_light_d board includes the following features:

1. On-board USB interface for SCI0
2. RGB color LED
3. DS1307 RTC interface header
4. CAN interface header
5. Dual 10-bit DAC for testing SPI interface and generating analog waveforms
6. Four digit 7-segment LED display for learning multiplexing technique
7. Eight LEDs
8. Eight-position DIP switch
9. Four push button switches
10. Speaker to be driven by timer, or DAC or PWM signal for alarm or music applications.
11. Dual H-Bridge motor driver controls two DC motor or ine stepper motor
12. 5V Power-On LED indicators
13. BDM-in connector to be connected with a BDM from multiple vendors for debugging
14. BDM POD mode for programming other HCS12 boards. No extra hardware needed
15. Abort switch (for D-Bug12 only) for stopping program when program is hung in a dead loop
16. Mode switch for selecting 4 operating modes: EVB, Jumo-to-EEPROM, BDM POD and Bootloader for D-Bug12 only
17. External 4 X 4 membrane keypad
18. X-Y-Z accelerometer interface or GP2-D12 distance measuring sensor interface for distance measurement
19. Potentiometer trimmer pot for analog input
20. Temperature sensor
21. Light sensor
22. Female headers provide shortest distance (great for high speed applications!) from bread board to every I/O pin of the MC9S12DG256

## 1.4 **I/O Pin Usage**

Many I/O pins of the MC9S12DG256 on the Dragon12_light_d board are used by on-board peripherals and it seems that there are only a few of unused pins left for your circuits on the breadboard. Fortunately, it's unlikely that all on-board peripherals will be used by one application program. So the I/O pins on unused peripheral devices can still be used by your circuits on the breadboard. For instance, if you don't touch the 4x4 on-board keypad, the entire port A will be available to your circuits. If you don't use the LCD or just unplug the LCD, the port K will be available as well. Port B drives LEDs, but if you ignore the status of the LED, the port B can drive any other I/O devices on the breadboard. Each pin in port H reads a switch, but it still can be used as an input for reading a TTL or CMOS output from your circuits.

| Pin Name | Pin # | I/O Usage |
|---|---|---|
| PA0 (output) | Pin 57 | Col_0 of keypad |
| PA1 (output) | Pin 58 | Col_1 of keypad |
| PA2 (output) | Pin 59 | Col_2 of keypad |
| PA3 (output) | Pin 60 | Col_3 of keypad |
| PA4 (input) | Pin 61 | Row_0 of keypad |
| PA5 (input) | Pin 62 | Row_1 of keypad |
| PA6 (input) | Pin 63 | Row_2 of keypad |
| PA7 (input) | Pin 64 | Row_3 of keypad |
| | | |
| PB0 (output) | Pin 24 | LED0 or H-bridge |
| PB1 (output) | Pin 25 | LED1 or H-bridge |
| PB2 (output) | Pin 26 | LED2 or H-bridge |
| PB3 (output) | Pin 27 | LED3 or H-bridge |
| PB4 (output) | Pin 28 | LED4 |
| PB5 (output) | Pin 29 | LED5 |
| PB6 (output) | Pin 30 | LED6 |
| PB7 (output) | Pin 31 | LED7 |
| | | |
| PE0 (input) | Pin 56 | Abort switch SW8 |
| PE1 | Pin 55 | not used |
| PE2 | Pin 54 | not used |
| PE3 | Pin 53 | not used |
| PE4 | Pin 39 | not used |
| PE5 | Pin 38 | not used |
| PE6 | Pin 37 | not used |
| PE7 | Pin 36 | not used |
| | | |
| PH0 (input) | Pin 52 | DIP switch 1 or pushbutton switch SW5 |
| PH1 (input) | Pin 51 | DIP switch 2 or pushbutton switch SW4 (input) |
| PH2 (input) | Pin 50 | DIP switch 3 or pushbutton switch SW3 (input) |
| PH3 (input) | Pin 49 | DIP switch 4 or pushbutton switch SW2 (input) |
| PH4 (input) | Pin 35 | DIP switch 5 (input) |
| PH5 (input) | Pin 34 | DIP switch 6 (input) |
| PH6 (input) | Pin 33 | DIP switch 7 (input) |
| PH7 (input) | Pin 32 | DIP switch 8 (input) |
| | | |
| PJ0 | Pin 22 | not used |
| PJ1 | Pin 21 | not used |
| PJ6 | Pin 99 | SDA for DS1307(U11) or external I2C (J2) |
| PJ7 | Pin 98 | SCL for DS1307(U11) or external I2C (J2) |
| | | |
| PK0 (output) | Pin 8 | RS of LCD module |
| PK1 (output) | Pin 7 | EN of LCD module |
| PK2 | Pin 6 | DB4 of LCD module (bi-directional) |
| PK3 | Pin 5 | DB5 of LCD module (bi-directional) |
| PK4 | Pin 20 | DB6 of LCD module (bi-directional) |
| PK5 | Pin 19 | DB7 of LCD module (bi-directional) |
| PK7 | Pin 108 | not used |

**Table 1-1: I/O pin usage list 1**

| Pin Name | Pin # | I/O Usage |
|---|---|---|
| PM0 | Pin 105 | CAN0 |
| PM1 | Pin 104 | CAN0 |
| PM2 | Pin 103 | not used |
| PM3 | Pin 102 | not used |
| PM4 | Pin 101 | not used |
| PM5 | Pin 100 | not used |
| PM6 | Pin 88 | CS of LTC1661 (DAC) |
| PM7 | Pin 87 | Not used |
| PP0 (output) | Pin 4 | Digit 3 of 7-segment display or EN12 of H-bridge |
| PP1 (output) | Pin 3 | Digit 2 of 7-segment display or EN34 of H-bridge |
| PP2 (output) | Pin 2 | Digit 1 of 7-segment display |
| PP3 (output) | Pin 1 | Digit 0 of 7-segment display |
| PP4 (output) | Pin 112 | Servo motor 1 or RGB LED |
| PP5 (output) | Pin 111 | Servo motor 2 or RGB LED |
| PP6 (output) | Pin 110 | Servo motor 3 or RGB LED |
| PP7 (output) | Pin 109 | Servo motor 4 |
| | | |
| PS0 | Pin 89 | SCI0 for PC communication, RECV |
| PS1 | Pin 90 | SCI0 for PC communication, XMIT |
| PS2 | Pin 91 | SCI1 for user applications, RECV |
| PS3 | Pin 92 | SCI1 for user applications, XMIT |
| PS4 | Pin 93 | MISO for LTC1661, and external SPI (J10) |
| PS5 | Pin 94 | MOSI for LTC1661, and external SPI (J10) |
| PS6 | Pin 95 | SCLK for LTC1661, and external SPI (J10) |
| PS7 | Pin 96 | /SS for external SPI (J10) |
| PT0 | Pin 9 | not used |
| PT1 | Pin 10 | not used |
| PT2 | Pin 11 | not used |
| PT3 (input) | Pin 12 | not used |
| PT4 (output) | Pin 15 | not used |
| PT5 (output) | Pin 16 | Speaker (output) |
| PT6 (output) | Pin 17 | BDMout reset (used in POD mode only) |
| PT7 | Pin 18 | BDMout data line (bi-directional, used in POD mode only) |
| | | |
| PAD0 | Pin 67 | D-bug12 mode select, SW7 |
| PAD1 | Pin 69 | D-bug12 mode select, SW7 |
| PAD2 | Pin 71 | not used |
| PAD3 | Pin 73 | not used |
| PAD4 | Pin 75 | Light sensor (phototransistor Q1) |
| PAD5 | Pin 77 | Temperature sensor (U14, LM45) |
| PAD6 | Pin 79 | not used |
| PAD7 | Pin 81 | Trimmer pot VR2 |
| PAD8 | Pin 68 | X axis input for Wytec accelerometer or ADC input for GP12D2 |
| PAD9 | Pin 70 | Y axis input for Wytec accelerometer or ADC input for GP12D2 |
| PAD10 | Pin 72 | Z axis input for Wytec accelerometer or ADC input for GP12D2 |
| PAD11 | Pin 74 | not used |
| PAD12 | Pin 76 | not used |
| PAD13 | Pin 78 | not used |
| PAD14 | Pin 80 | not used |
| PAD15 | Pin 82 | not used |

**Table 1-2: I/O pin usage list 2**

# Chapter 2. Quick Start

By default the Dragon12_light_d board is pre-installed with the bootloader (Freescale AN2153.pdf) and the D-Bug12 monitor (Freescale DB12RG4.pdf). In chapters 2 and 3 the AsmIDE is used as the main software tool to develop and debug assembly programs. If you prefer to use CodeWarrior IDE for program development and your board is pre-installed, per your request, with the serial monitor (Freescale AN2548.pdf), **skip the chapters 2 and 3 after installing software.**

People often use different terminologies. In our product manuals, **Download** means to transfer a file from PC to a development board, while **Upload** means to transfer a file from a development board to PC. Throughout the manual, **left click** means that you click the left button of the mouse and **right click** means that you click the right button of the mouse.

## 2.1 Install software:

After downloading software from our web site, unzip it to your drive C. It will create a folder, c:\dragon12_light_d. All sample programs are located at c:\dragon12_light_dt\examples

After the software is successfully installed, you can make a shortcut to AsmIDE.exe on the desktop. First, right click the Start button, then left click "Explorer", left click on c:\dragon12_light_d, right click on AsmIDE.exe (an application program), left click "Send to" and finally left click "Desktop" (do not click "COPY"). It will create an icon named "shortcut to AsmIDE" on the desktop. You can double check the target location by right clicking on the icon, then left click on "properties". You should see that the target location is c:\dragon12_light_d. If you want to make a shortcut for AsmIDE on the Desktop, this is the correct way to do it. If you don't follow this method, your may have a problem running your program. Never drag the AsmIDE.exe to the desktop folder.

## 2.2 Getting Started (for the AsmIDE and D-Bug12 monitor firmware only)

To operate the dragon12_light_d board, follow steps1 through 5 below:

**1.** Make sure that the both DIP switches of SW7 must be set in the "LOW" positions for EVB mode, then plug the USB cable to the USB jack P1 on the **upper left** corner of the Dragon12_light_d board. Plug the other end of the USB cable into a USB port of your PC. After power up, the PB7-PB0 LEDs should flash sequentially from left to right, the speaker should chirp once (If the chirp is too soft you can remove the sticker on the speaker to increase the volume) and the LCD should display the following message:

**" DRAGON12-Light " ; you can display your name on LCD and see details**
**"D-Bug12 EVB MODE" ; at CDROM\examples\name_display\readme.txt**

If it does not occur, make sure that the Power-On LED indicator is on. The PWR LED is on when VCC (5V) is present.

**2.** To invoke the AsmIDE, you can right click the Start button, then left click "Explorer", left click on C:\Dragon12_light_d and finally, double left click on AsmIDE.exe. If you have created a shortcut icon on the desktop, just double click the AsmIDE icon on the desktop.

**Warning note: In order to establish a reliable USB communication, always connect the Light board to your PC's USB port first before invoking the IDE (CodeWarrior or AsmIDE), otherwise the IDE may not be able to communicate with the Light. During a debugging session, if you accidentally unplug the USB cable from the Light, you need to re-establish the USB communication. The IDE will not recognize the Light again if you just simply plug the USB cable back in.**

**To re-establish the USB communication you need to exit the IDE, then power cycle the dragon12_light board, then re-invoke the AsmIDE to re-establish its communication with the dragon12-light board. If this does not work, you need to restart your PC. If the Restart does not solve your problem, the board may be defective or the D-Bug12 monitor may be corrupted..**

The AsmIDE is simple and very easy to use. You only need to use three commands from the AsmIDE for your HCS12 development work. Use the File command to edit your source code, the Build->Assemble command to assemble your source code, and the Build->Download command to download an s19 file to the Dragon12_light_d board.

In the View->Option->Terminal Window Options menu, set the COM port as 1 or 2 to match the COM port number that is assigned to the USB port by Device Manager in control panel. Also, set the COM port options at 9600, N,8,1, and check the "enable the terminal window".

3. After reset, the D-Bug12 monitor defaults baud rate at 9600 and and Hyperbaud function is disabled. If Hyperbaud function is enabled, the Hyperbaud toolbar button sends the BAUD 57600 command to the D-Bug12 monitor, and then it also changes the serial port to the 57600 baud rate. **IMPORTANT:** When you reset your board it will go back to 9600 baud and you will see characters 'aaaaaaaaaa' on the screen. You will need to press the Hyperbaud button once to return AsmIDE to 9600 baud, and press it again to get 57600 baud. To stay at the 57600 baud all the time, you need to press the Hyperbaud button twice after every reset. The Hyperbaud function is disabled by default and it should only be used by an experienced user, not a beginner.

4. You can program text values for function keys to be sent from the terminal window. Some function keys are pre-programmed, but you can change it any time in configuration options (View->Options->Terminal Func Keys).

5. In the View->Option->Assembler menu, make sure that the chip family is **68HC12**, not 68HC11. If you would like to use your own assembler, you can replace the as12.exe with the name of your own assembler.

6. The screen is divided into two windows. The top window is for editing your source code and the bottom window is shared by the **message window** and the **terminal window**.

If the terminal options are set correctly, you should see the following prompt every time the reset button on the Dragon12_light_d board is pressed. If you do not see this, the bottom window may be set for message window. Sometime it's a little confusing when terminal window is disabled and the message window does not display what you have typed. In order to enable terminal window you have to click the terminal button in the bottom window to enable the terminal window display, then move the cursor to any location in the terminal window and click the left button on the mouse. After seeing a solid block cursor flashes, press the <Enter> key and it will enable the terminal window.

D-Bug12 v4.0.0b32
Copyright 1996 - 2005 Freescale Semiconductor
For Commands type "Help"
>

## 2.3 **Test Hardware:**

To help users get up and running, the Dragon12_light_d board comes with many fully debugged and ready-to-run sample programs including source code. The hardware test program, test.asm, simultaneously scans the keypad, plays a song, multiplexes the 4 LED seven segment display, changes display brightness by adjusting the trimmer pot.

All sample programs must be run from RAM in EVB mode. In order to run the test program in EVB mode, the both DIP switches of SW7 must be set in the “low” positions (the picture at the left of the SW7 shows the switch settings for 4 different modes).

The steps to run your first sample program are as follows:

1. Click the File button to open the test.asm from c:\Dragon12_light_d\examples. After the test.asm is loaded into the AsmIDE window, you can view instructions of how to test all hardware on the Dragon12_light_d board.

2. Click the Build button to assemble code and generate the test.s19 file. This is how you normally generate an s19 file. You can omit this step, because the test.s19 is already on your hard disk.

3. Press the reset button on the board, you will see:

D-Bug12 v4.0.0b32
Copyright 1996 - 2005 Freescale Semiconductor
For Commands type "Help"
>

4. Type “LOAD”, then hit <Enter> key.

5. Click the Build button. Select Download option and locate the file ‘test.s19’ for downloading. If it prompts you with the “save changes?” message, you can ignore that message and click the “No” answer.

6. After download is done, type “G 2000” and hit <Enter> key to run the test program.

All sample programs are developed in RAM. You can try to run a different example program later after you have finished reading this manual. You should always press the reset button before downloading a new program, because the new program may not work if an interrupt was enabled by a previous program.

All example programs are fully debugged, so the assembler won’t generate an error. If you have an error, even a warning error, in your program, you must correct it before it can generate an s19 file.

# Chapter 3. Software descriptions

## 3.1 **Bootloader and D-Bug12 Monitor**

If the MC9S12DG256 on the dragon12_light_d board is pre-loaded with bootloader and D-Bug12 monitor firmware and it will operate in 4 different modes depending on the setting of the 2-position DIPswitch, SW7. After power up or reset, the MC9S12DG256 will read the PAD0 and PAD1 to decide which mode to boot up.

The bootloader (**AN2153.PDF)**, the D-Bug12 reference guide (**DB12RG4.PDF**) and the MC9S12DG256 data book **(MC9SDG256.PDF**) are the most important documentations. They can be found online. The HCS12 instruction set, register map and memory map can be found on page 26, 65 and 120 of the data book, respectively.

The new D-Bug12 V4.x is much different and much larger (about 60K) than old D-Bug12 V2.x. The $C000-$EFFF are just a part of the monitor, In 16-bit S1 record they are $C000-$EFFF. In 24-bit S2 record, they are $FC000-FEFFF (ppage=$3F). Since the ppage register deals with the 16K window $8000-$BFFF the addresses $C000-$FFFF are not affected by the ppage. The other part of the monitor is at C0000-C87FF (16K window $8000-$BFFF when ppage=$30,$31 and $32). See details on page 20 of the app note AN2153 or page 71 of the D-Bug12 v4 reference guide on the CD.

### 3.1.1 **EVB mode:** PAD1=0, PAD0=0.

This is the standard debug environment running on the MC9S12DG256 for on-chip RAM or EEPROM based code development. Using an IDE program to view and modify registers and memory locations, you may set breakpoints, single step through programs, and assemble and disassemble code as you would in a BUFFALO monitor based Freescale 68HC11 EVB. It gives you 12K RAM and 3K EEPROM to develop and debug your code. You must place your interrupt vectors at $3E00-$3E7F, because real interrupt vector addresses are taken by bootloader, bootloader and D-Bug12 monitor will redirect interrupts to the RAM interrupt vector table at $3E00-$3E7F.

After booting up in this mode, the LCD should display the following message:

**“ DRAGON12-Light ”**
**“D-Bug12 EVB MODE”**

and you should see the following message on PC screen:

```
D-Bug12 v4.0.0b32
Copyright 1996 - 2005 Freescale Semiconductor
For Commands type "Help"
>
```

Typing “help” then <Enter> will display a list of available commands.

In this mode, you **cannot** erase or program on-chip flash memory.

If the D-Bug12 monitor is erased, the LCD will display the following message after reset:

**“ DRAGON12-Light ”**
**“ D-Bug12 ERASED ”**

You can use bootloader to re-program D-Bug12 monitor into flash memory.

**Note:** Some user may accidentally erase D-Bug12 monitor in bootloader mode, so it is important to know how to re-program D-Bug12 monitor in bootloader mode.

### 3.1.2 **Jump-to-EEPROM mode:** PAD1=0, PAD0=1

This mode enables the MC9S12DG256 to jump directly to the internal EEPROM at location $0400 upon reset.

This mode makes the MC9S12DG256 a replacement for the old 68HC811E2 microcontroller, but it also gives you 3K EEPROM instead of 2K EEPROM with the 68HC811E2. The bus speed is 4MHz, one half of the crystal frequency by default, the PLL function must be initialized by user's code for a higher bus speed, because the D-Bug12 monitor firmware that boosts bus speed to 24 MHz is bypassed. If you need to auto start your code upon reset, the procedure is available in the folder named eeprom_programming.

After booting up in this mode, the LCD should display the following message:

**" DRAGON12-Light "**
**" JUMP TO EEPROM "**

### 3.1.3 **BDM POD** mode: PAD1=1, PAD0=0

In this BDM POD mode, the D-Bug12 firmware acts as a master to access all target MCU resources on the target board (another dragon12_light_d board) via the BDM port in a non-intrusive manner. It becomes a BDM that will have all the features that a standard BDM has in debugging the target MCU. Also, it gains all the features a programmer has for programming the flash memory of the MCU on the target board (another dragon12_light_d board).

To use the master board as a programmer, you need a 6-pin ribbon cable to connect from the BDM OUT of the master board to the BDM IN of the target board (make sure that the orientation of the cable is correct). You don't have to provide the power to both boards, but only to one board. The master board communicates to a PC COM port while the target board does not need to be connected to a PC COM port.

After booting up in this mode, the LCD should display one of the following two messages:

If the D-Bug12 monitor is erased, the LCD will display the following message after reset:

**" DRAGON12-Light "**
**"POD-Bug12 ERASED"**

Otherwise it will display:

**" DRAGON12-Light "**
**" BDM POD MODE "**

and you should see the following message on PC screen:

Can't Communicate With Target CPU

1.) Set Target Speed (48000 KHz)
2.) Reset Target
3.) Reattempt Communication
4.) Erase & Unsecure
?

You first must set the target speed with the choice 1). After entering the first choice, you will be prompted to enter the target speed. It's the crystal frequency, not the bus speed that is boosted up by the on-chip PLL. After a reset, before the PLL is enabled, the target MC9S12DG256 is running from the crystal frequency, not the PLL frequency. Enter 8000 for the target speed. After the correct speed is entered, the master will try to communicate with the target board. If it's not successful, enter choice. 2) to reset the target board.

**Note:** The newer D-Bug12 monitor in POD mode may auto-detect the crystal frequency of a target board, so most likely the step 1 may not be needed.

Can't Communicate With Target CPU

1.) Set Target Speed (8000 KHz)
2.) Reset Target
3.) Reattempt Communication
4.) Erase & Unsecure
? 1

Enter Target Crystal Frequency (kHz): 8000

Can't Communicate With Target CPU

1.) Set Target Speed (8000 KHz)
2.) Reset Target
3.) Reattempt Communication
4.) Erase & Unsecure
? 2

When the communication is established, you will see the following:

D-Bug12 v4.0.0b32
Copyright 1996 - 2005 Freescale Semiconductor
For Commands type "Help"

S>

You will notice that the debug prompt is "S>" in the POD mode, not just a ">" in the EVB mode. The S> tells that this is the POD mode and the MC9S12DG256 on target (slave board) is stopped. Sometimes the prompt could be an "R>" that means the target MCU is running. If you see the "R>", just type "reset" then <Enter> to reset the target and it will come back to the "S>" prompt.

R>Reset <Enter>
S>

**Note:** The initial communication in POD mode does not always work smoothly and sometimes the PC screen would only display an incomplete sign-on message. You need to re-start it all over again by pressing reset buttons on the master board, then press the Enter key on PC keyboard. You cannot go to the next step until PC screen shows the prompt 's>'.

In order to program the flash memory, you have to erase it by using the FBULK command.

S>fbulk <Enter>
S>

When the prompt "s>" returns, the FBULK command has already erased all of the flash memory contents of the target MC9S12DG256 including the bootloader. If it returns with a message "Flash or EEPROM Failed To Erase" the MC9S12DG256 is defective.

Now we are going to program the bootloader and the D-Bug12 into the flash memory of the target MC9S12DG256.
Before we actually program the flash memory, we must understand there are two different types of s-record file that can be generated by compilers and assemblers.

An s1-record uses a 16-bit starting address field while an s2-record uses a 24-bit starting address field.
An s1-record file looks like this:

```
S123FFA0F64CF650F654F658F65CF660F664F668F66CF670F674F678F67CF680F684F6883D
S123FFC0F68CF690F694F698F69CF6A0F6A4F6A8F6ACF6B0F6B4F6B8F6BCF6C0F6C4F6C81D
S123FFE0F6CCF6D0F6D4F6D8F6DCF6E0F6E4F6E8F6ECF6F0F6F4F6F8F6FCF700F704F00009
S9030000FC
```

An s2-record file looks like this:

```
S2240FEFA0DB70DB66DB5CDB52DB48DB3EDB34DB2ADB20DB16DB0CDB02DAF8DAEEDAE4DADA41
S2240FEFC0DAD0DAC6DABCDAB2DAA8DA9EDA94DA8ADA80DA76DA6CDDD0DA62DA58DA4EDA4494
S2240FEFE0DA02DA0ADA12DA1ADA22DA2ADA32DA3AD9FAD9F2D9AFD98AD9D5EF00EF00EF0039
S9030000FC
```

We are not going to explain the s-record format here. If you would like to know more on the subject, you can review the D-Bug12 reference guide on the CDROM (**BD12RG4.PDF**). It explains the subject in great details. Right now, all you need to know is that an s1-record file must be converted to an s2-record file before using the FLOAD command. The "FLOAD" command in the D-Bug12 is for downloading an s2-record file.

Our dragon12_light_d bootloader is modified from the Motorola's BootDP256.asm. We added our modification to the original source code and the s record file is generated by the AsmIDE. It's an s1-record file and we converted it into an s2-record file by using the following commands:

Sreccvt –m c0000 fffff 32 –of f0000 -o Boot_DR12_8MHz.s29 Boot_DR12_8MHz.s19

Now we type "FLOAD" <Enter> at the prompt. Click the Build button, select the Download option, and select the file named **Boot_DR12P_8MHz.s29** located in the folder named "D-Bug12_Monitor ". You should see the following on the terminal window when programming is done (when the prompt "s>" appears):

S>fload <Enter>

**********

S>

Now we are going to program the D-Bug12 monitor into the flash memory. We need to type “FLOAD” <Enter> at the prompt. Click the Build button, select the Download option, and select the file named **DBug12v32_DR12_light_8MHz** located in the folder named “D-Bug12_Monitor”. You should see the following on the terminal window when programming is done (when the prompt “s>” appears):

S>fload <Enter>

```
**************************************************************************
**************************************************************************
**************************************************************************
**************************************************************************
**************************************************************************
**************************************************************************
**************************************************************************
**************************************************************************
**************************************************************************
************************************************************
```

S>

> With the bootloader and the D-Bug12 programmed in the flash memory, the target board now becomes a true development board. That’s how we program the board before we ship it. Your dragon12_light_d board actually becomes a programmer. You can then repeat above steps as many times as you want. Just unplug the 6-pin BDM cable from the target board, and then plug it into a new target board to program its flash memory with these two files. You even don’t have to turn off the power while doing this.

For your convenience, we combined both the bootloader and the D-Bug12 monitor into a single s2 file named **Boot_ DBug12v32_DR12_light_8MHz .s29**. In case you need to update both of them, you can download this combined file.

> The D-Bug12 monitor is an application program runs from the bootloader. If you program the D-Bug12 portion of flash memory with your application program, your program will run automatically in EVB mode after power up or reset. When running your code instead of the D-Bug12 monitor, the bus speed is 4MHz, one half of the crystal frequency by default. The PLL function must be initialized by your code for a higher bus speed, because the D-Bug12 monitor firmware was not in flash memory anymore. For your convenience, we include a PLL code template in chapter 7.

If you need to auto start your code upon reset, the procedure is available in the folder named flash_programming.

#### 3.1.4 **BOOTLOADER** mode: PAD1=1, PAD0=1

This bootloader allows you to erase/program flash memory and erase EEPROM. It is mainly used to program the D-Bug12 monitor into flash memory or download a user’s fully debugged code into the D-Bug12 portion of flash memory. The latter allows the board to be operated in EVB mode and start your code every time the board is turned on or reset.

When you program your code into the D-Bug12 portion of flash memory, it wipes out the D-Bug12 monitor. You can restore it any time, just as if you were downloading another application program since the bootloader is not erased. You can erase and program the D-Bug12 monitor portion of the flash memory of the MC9S12DG256 on its own board in bootloader mode, but you cannot erase and program bootloader by itself. **The bootloader can only be erased by an external BDM via BDM-in port.**

After booting up in this mode, the LCD should display the following message:

**“ DRAGON12-Light ”**
**“ BOOT LOADER ”**

and you should see the following bootloader menu on PC screen:

MC9S12DG256 bootloader menu:

a) Erase Flash
b) Program Flash
c) Set Baud Rate
d) Erase EEPROM
?

The option a) will erase the D-Bug12 portion of flash memory, not the bootloader itself.
The option b) will program the D-Bug12 portion of flash memory, not the bootloader itself.

The file to be programmed into flash memory must be an s2-record file. If your assembler and compiler generate s1-record files only, you must convert an s1-record file to an s2- record file before programming flash memory with the bootloader.

The option c) will set a new baud rate.
The option d) will erase all on-chip EEPROM.

**Note:** Some users may accidentally erase the D-Bug12 monitor when entering this mode, so it is important to know how to re-program the D-Bug12 monitor.

To program flash memory with the D-Bug12 monitor:

1. Enter the option a) to erase D-Bug12 portion of flash memory. Wait until the bootloader menu re-appears after flash memory is erased.
2. Enter the option b), the bootloader will wait for your file. **Do not type** any thing on keyboard.
3. Click the Build button, select the Download option, and select the file named **DBug12v32_DR12_light_8MHz .s29** located in the folder named “D-Bug12_Monitor” for downloading. You should see the following on the screen:

```
**************************************************************************
                **************************************************************************
**************************************************************************
**************************************************************************
****************************************
```

**4.** It will take 3 minutes to program the D-Bug12 at 9600 baud rate and the bootloader menu will reappear after the D-Bug12 monitor is successfully programmed into flash memory.

## 3.2 **Making a simple assembly program in RAM:**

We are using AsmIDE as a terminal program and the following instructions to create your first assembly program. If you are using a different terminal program, the instructions may vary.

The steps to create your first program are as follows:

1. Click the **File** button to open a new file.

In assembly language, you specify the starting address of your CODE by an ORG statement.

You can start the data RAM at address $1000 with the statement org $1000 followed by RAM variables, as shown by:

```
            org     $1000

count:      rmb     1           ; reserve one byte of RAM for temp storage
temp:       rmb     2           ; reserve two bytes of RAM for temp storage
```

If your program is small, say less than 4K, you can start your program at address $2000 with the statement org $2000 followed by your program, as shown by:

```
            org     $2000
```

It will assemble your source program and generate hex code within 4K locations from $2000 to $2FFF.

Here is a very simple program, but it's complete. It will flash the PB0 LED at 2Hz when it's running. The RAM byte named 'counter' is added for demonstrating how a RAM data byte is used in a user program. In this simple program it's not really necessary, because the accumulator A can be used as the RAM byte 'counter'.

For a good programming practice, you should always place the lds instruction in the first line of your code.

```
#include    reg9s12.h
REGBLK:     equ     $0000
STACK:      equ     $2000       ; do not use $4000
;
            org     $1000
counter:    rmb     1

            org     $2000       ; program code
start:      lds     #STACK
            ldx     #REGBLK

            ldaa    #$ff

            staa    ddrb,x      ; make port B an output port
            staa    ddrp,x      ; make port P an output port
            staa    ptp,x       ; turn off 7-segment LED display
```

```
back:   clr     portb,x         ; turn off PB0
        jsr     d250ms          ; delay 250ms
        inc     portb,x         ; turn on PB0
        jsr     d250ms          ; delay 250ms
        jmp     back
*
d250ms: ldaa    #250            ; delay 250 ms
        staa    counter

delay1: ldy     #6000           ; 6000 x 4 = 24,000 cycles = 1ms
delay:  dey                     ; this instruction  takes 1 cycle
        bne     delay           ; this instruction  takes 3 cycles
        dec     counter
        bne     delay1          ; not 250ms yet, delay again
        rts

        end
```

2. Click File button, select Save option to save your assembly source file. Save your file frequently while editing. If you are creating a new file and giving the file a name to save, enter the file name including file extension, such as “Flash_PB0.asm”, not just “Flash_PB0”.

3. Click Build button, select Assemble option, or click the assembler button on the toolbar to assemble your code and generate an s19 file. If the assembler detects an error, the error message will show the line numbers of your source code that caused the error. You have to correct all errors in your program.

4. Go to the line and correct the errors and go back to step 3 until there are no errors.

5. Press the reset button on the board, you will see:

```
D-Bug12 v4.0.0b32
Copyright 1996 - 2005 Freescale Semiconductor
For Commands type "Help"
>
```

6. Type “LOAD” and then hit <Enter> key

7. Click Build button, select Download option and locate the file named ‘Flash_PB0.s19”’ for downloading. After download is done, type “G 2000” and hit <Enter> key to run the program.

For your convenience, we have included this sample program in the folder named “example”.

## 3.3 Software development

### 3.3.1 **Use on-chip 12K RAM for software development in EVB mode.**

You can download your s19 file into the RAM and debug it with the D-Bug12 monitor in EVB mode. You must place your interrupt vectors at $3E00-$3E7F, because real interrupt vector addresses are taken by the bootloader. The bootloader and the D-Bug12 monitor will redirect interrupts to the RAM interrupt vector addresses at $3E00-$3E7F

Because RAM will lose its contents after power off, you have to load your program every time after power-up. In the beginning of your program, you must initialize the interrupt vectors at $3E00-$3E7F.

In all sample programs, the user program code locations are at $2000-$3FFF. The user data RAM locations are at $1000-$1FFF. The 64 RAM interrupt vector addresses are at $3E00-$3E7F.

The 64 RAM interrupt vector addresses (128 bytes of RAM) are assigned by the D-Bug12 monitor to different interrupt sources. The listing of interrupt sources is show on chapter 7.

### 3.3.2 **Use on-chip 3K EEPROM for testing your code in EVB mode.**

If your program is small enough to fit into a 3K range, then you can download your code into the EEPROM. In this way, your program can be auto started from $0400 upon reset. You cannot set software breakpoints and single step in the EEPROM in EVB mode, so it makes sense to do development work in the RAM first. When your code is completely debugged, then re-assemble or re-compile it at $0400 and download the final s19 file into the EEPROM for the auto start feature. With the early versions of D-Bug12 monitor, an s19 file must be converted to an s29 file to program the EEPROM, but it's not required in the current version.

Like the RAM-based development, your interrupt vectors are at $3E00-$3E7F. In the beginning of your program, you must initialize the interrupt vectors at $3E00-$3E7F.

### 3.3.3 **Program on-chip flash memory in BOOTLOADER mode.**

In this mode, you download your program code directly into on-chip flash memory. You first erase the D-Bug12 monitor portion of flash memory, and then program that portion of the flash memory by downloading your application program code in an s29 file. Your program code will replace the D-Bug12 monitor in the flash memory. The bootloader portion of the flash memory remains intact. To run your code, set the mode switch SW 7 to EVB mode, then press the reset button. It usually runs the D-Bug12 monitor, but now it runs your program. The flash memory is non-volatile like the EEPROM. Your code will run every time the board is turned on or reset.

The bootloader redirects interrupts to the secondary interrupt vectors at $EF80-$EFFF. The D-BUG12 is not present and the interrupt vectors of your program are at $EF80-$EFFF. The addresses $EFFE and $EFFF contains the starting address of your program.

In order to program the MC9S12DG256 flash memory, you must program an even number of bytes and begin on an even address boundary for each s-record. If any one s-record in the file contains an odd number of bytes or begins with an odd address, the flash memory cannot be programmed. If your assembler or compiler cannot generate the even format, you must use the Freescale s-record conversion utility **sreccvt.exe** to convert your odd format to the even format by using the following command line:

Sreccvt –m c0000 fffff 32 –of f0000 –o test.s29 test.s19

It will create a new file named test.s29 that has the even format and can be programmed into flash memory. For your convenience, the sreccvt.exe is included in the folder named CDROM\document\Sreccvt-GUI.

# Chapter 4: Hardware Descriptions

The crystal frequency is 8 MHz and usually it will result in a 4 MHz bus speed, but on this board the MC9S12DG256's internal PLL boosts the bus speed up to 24 MHz.

The circuit is designed in such way that the value of all resistors and capacitors are not critical.

## 4.1 **LEDs:**

Each port B pin is monitored by a LED. In order to turn on a port B LED, the pin must be programmed as an output pin first, it can be done by setting the corresponding bit to 1 in the port B direction register, then set the corresponding bit to 1 in the port B data register. On the Dragon12-Plus2, the PJ1 (pin 21 of the MC9S12DG256) must be programmed as output and set for logic zero. This is not needed on this Dragon12_light_d board.

## 4.2 **DIP switch and pushbuttons:**

Port H is connected to an 8-position DIP switch. The DIP switch is connected to VCC via the RN4 (four 2.7K resistors) and RN5 (four 2.7K resistors). To GND Via RN8 (four 47K resistors) and RN9 (four 47K resistors), so it's not dead short to VCC or GND. When port H is programmed as an output port, the DIP switch setting is ignored, but for the best result all 8 DIP switches should be open (at the low positions).

## 4.3 **7-Segment LED multiplexing**

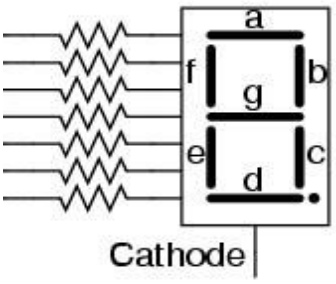
**Note: On the rev D board, the 7-segment is changed from Common Cathodes to Common Anode, but we added inverters for the display, so it behaves like Common Cathode, so your software written in the past will run without any modifications.**

There are 4 digits of 7-segment LEDs on the Dragon12_light_d board. The type of the 7-segment LED on board is called common cathode. In an individual digit, all anodes are driven individually by an output pin and all cathodes are internally connected together.

Before sending a number to a 7-segment LED, the number must be converted to its corresponding 7-segment code depending on how the 7-segment display is connected to an output port.

The Dragon12_light_d board uses port B to drive 7-segment anodes and uses PP0-PP3 to drive common cathodes. We will explain how to multiplex 7-segment by displaying the number 1234 on the display.

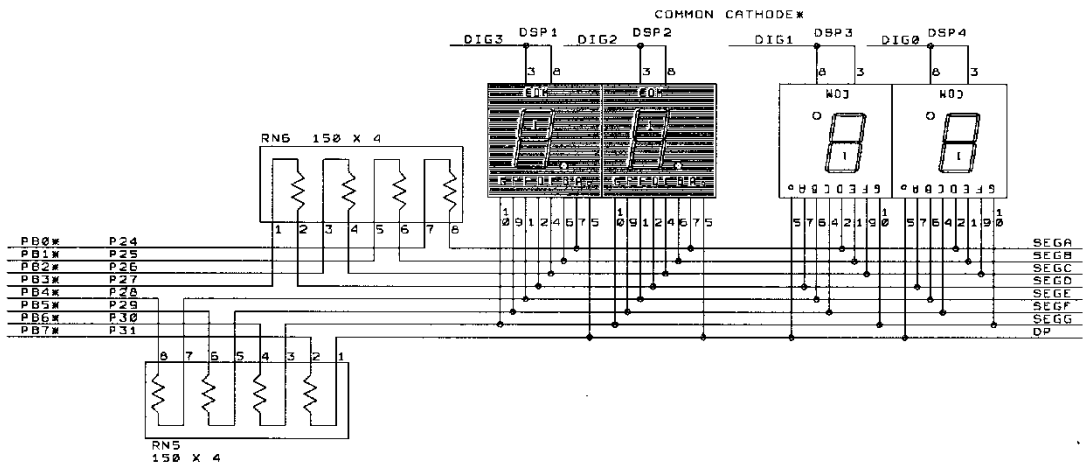
By convention, the 7segments are called segment A, B, C, D, E, F and G. Their locations in the display are shown below:




The segment A, B, C, D, E, F, G and Decimal Point are driven by PB0, PB1, PB2, PB3, PB4, PB5, PB5 and PB7, respectively. The hex value of the segment code is shown in the following table:

| Number | DP | G | F | E | D | C | B | A | Hex Value |
|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | $06 |
| 2 | 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | $5B |
| 3 | 0 | 1 | 0 | 0 | 1 | 1 | 1 | 1 | $4F |
| 4 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | $66 |

The schematic for multiplexing 4 digits is shown below. The two of the digits at the right are deliberately placed upside down and the hardware connections compensate for this configuration. The reason for the upside down digits is to place two decimal pointers on the middle as a colon for a clock display.



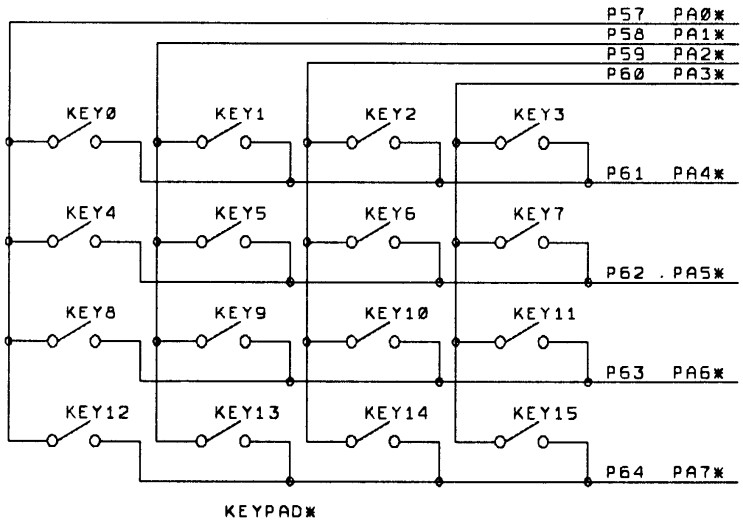

The digit 3, 2, 1, and 0 are driven by PP0, PP1, PP2 and PP3, respectively. The 7-segment LED is turned on one at a time at 250 Hz refresh rate. It's so fast that our eyes will perceive that all 4 digits are turned on at the same time. To display the number 1234 on the 7-segment display, the following steps should be taken:

1. Output $06 to port B, set PP0 low and PP1, PP2, and PP3 high. The number 1 is shown on the digit 3 (the leftmost digit), but other 3 digits are turned off.
2. Delay 1ms.
3. Output $5B to port B, set PP1 low and PP0, PP2, and PP3 high. The number 2 is shown on the digit 2, but other 3 digits are turned off.
4. Delay 1ms.
5. Output $4F to port B, set PP2 low and PP0, PP1, and PP3 high. The number 3 is shown on the digit 1, but other 3 digits are turned off.
6. Delay 1ms.
7. Output $66 to port B, set PP3 low and PP0, PP1, and PP2 high. The number 4 is shown on the digit 0 (the rightmost digit), but other 3 digits are turned off.
8. Delay 1ms.
9. Go back to step 1.

## 4.4 **Keypad:**

Port A is an 8-bit bi-directional port. Its primary usage is for a 4X4 keypad. If the port is not used for the keypad, it can be used as a general-purpose I/O port.

The schematic for the keypad connections is shown below:




Keypad connections:

PA0 connects COL0 of the keypad
PA1 connects COL1 of the keypad
PA2 connects COL2 of the keypad
PA3 connects COL3 of the keypad
PA4 connects ROW0 of the keypad
PA5 connects ROW1 of the keypad
PA6 connects ROW2 of the keypad
PA7 connects ROW3 of the keypad

Keypad scan routine sets PA3 low and PA0, PA1,PA2 high, then tests PA4-PA7.
If no key is down, PA4-PA7 remain high.
If PA7 = low, the key 15 is down.
If PA6 = low, the key 14 is down.
If PA5 = low, the key 13 is down.
If PA4 = low, the key 12 is down.

Keypad scan routine sets PA2 low and PA0, PA1, PA3 high, then tests PA4-PA7.
If no key is down, PA4-PA7 remain high.
If PA7 = low, the key 11 is down.
If PA6 = low, the key 10 is down.
If PA5 = low, the key 9 is down.
If PA4 = low, the key 8 is down.

Keypad scan routine sets PA1 low and PA0, PA2, PA3 high, then tests PA4-PA7.
If no key is down, PA4-PA7 remain high.
If PA7 = low, the key 7 is down.
If PA6 = low, the key 6 is down.
If PA5 = low, the key 5 is down.
If PA4 = low, the key 4 is down.

Keypad scan routine sets PA0 low and PA1, PA2, PA3 high, then tests PA4-PA7.
If no key is down, PA4-PA7 remain high.
If PA7 = low, the key 3 is down.
If PA6 = low, the key 2 is down.
If PA5 = low, the key 1 is down.
If PA4 = low, the key 0 is down.

### 4.5 **LCD display**

Port K is an 8-bit bi-directional port. It's used for the LCD display module. If the port is not used for the LCD display, it can be used as a general-purpose I/O port.

The pinouts of J11 are as follows:

| | | |
|---|---|---|
| Pin 1 | GND | |
| Pin 2 | VCC (5V) | |
| Pin 3 | Connect to GND via the VR1 for contrast adjustment | |
| Pin 4 | PK0 | RS pin for LCD module |
| Pin 5 | GND | Write only for LCD module |
| Pin 6 | PK1 | EN pin for LCD module |
| Pin 7 | Not used | |
| Pin 8 | Not used | |
| Pin 9 | Not used | |
| Pin 10 | Mot used | |
| Pin 11 | PK2 | DB4 pin for LCD module |
| Pin 12 | PK3 | DB5 pin for LCD module |
| Pin 13 | PK4 | DB6 pin for LCD module |
| Pin 14 | PK5 | DB7 pin for LCD module |
| Pin 15 | Via a 22 Ohm resistor to VCC | LED backlight for LCD module |
| Pin 16 | GND | |

Please notice that PK2-PK5 (not PK4-PK7) are used to drive DB4-DB7 of the LCD module.

The LCD module is hardwired for write-only operation.

### 4.6 **Trimmer pot**

The trimmer pot VR2 is connected to the AN07 input of the ADC port.

### 4.7 **Dual Digital-to-Analog Converters (DACs)**

The on-board 2-ch, 10-bit DAC is installed for learning SPI communication. It convers a digital binary code to an analog signal so a program can generate different waveforms from the DAC.

The DAC installed on the board is LTC1661. Its analog output, OUTA, is provided on the pin between the headers H7 and H8. The other analog output, OUTB, is provided on the pin between the headers H1 and H2. A good application is to connect a DAC output to an ADC input, so a user can send a binary code to the DAC and read the code back from the ADC.

## 4.8 **Speaker**

The speaker is a 5V audio transducer and it can be driven by PT5, Output Comparator 3, or PP5 (PWM 5), or the output B of the DAC LTC1661. The jumper on J26 is preset for the PT5 at factory and all sample programs on the CD will drive the speaker via PT5.

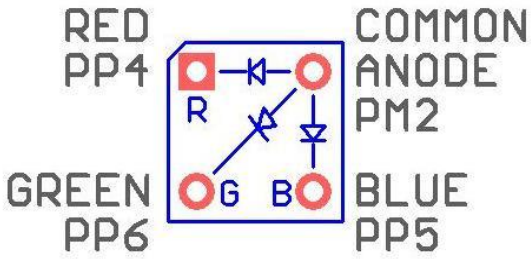
During reset, the bootloader or the serial monitor will generate a chirp via the speaker. If the jumper is not installed for the PT5, the chirp won't happen.

## 4.9 **Dual SCI communication ports**

The SCI0 is used by D-Bug12 or serial monitor for developing and debugging user programs.

The SCI1 can be used by user's application programs, but it requires a FTDI cable to be plugged on J71. For loopback testing you can connect PS2 and PS3 on the J71. The J71 is located next to the keypad header (J76).

## 4.10 **RGB LED ( Common Anode)**




The anode is enabled by PM2.

The PP4, PP5 and PP6 control Red, Blue and Green LEDs, respectively.

## 4.11 **All jumper settings**

All on-board jumpers:

J2 DS1307 RTC header and $I^2C$ interface
J6 PP4 PWM output for a servo, the servo connector must be installed horizontally
J7 PP5 PWM output for a servo, the servo connector must be installed horizontally
J8 PP6 PWM output for a servo, the servo connector must be installed horizontally
J9 PP7 PWM output for a servo, the servo connector must be installed horizontally
J11 On-board LCD connector for a 16x2 LCD

J20 BDM input
J21 BDM output, when the board is booted in POD mode
J25 DC motor power select. The jumper is installed on the two left pins if motors are powered by the on-board unregulated 9V (VIN). The jumper is installed on the two right pins if motors are powered by external voltage source that is less than 15V at the terminal block T3.

J26 Selects speaker driving source. The speaker can be driven by PT5 (OC3), PP5 (PWM) and DAC B

J35 Servo motor power select. The jumper is installed on the two left pins if servos are powered by the on-board VCC (5V). The jumper is installed on the two right pins if servos are powered by an external 5V power supply at the terminal block T7.
J36 X-Y-X Accelerometer module interface

J39 Connects PM0 to RXD of CAN interface U2, It's hard-wired. (Located on solder side)

J40 PJ1 bypass, when a jumper is not installed, it will require the PJ1=0 to sink port B LEDs. For the Dragon12_light_d Rev D board, this jumper should be installed.

J57 Analog sensor input 1 and can be used for an IR distance sensor, such as GP2D12 or other analog or digital sensors.

J58 Analog sensor input 2 and can be used for an IR distance sensor, such as GP2D12 or other analog or digital sensors
J62 Provides VCC to the H-Bridge motor control IC, U12 (TB6612FNG). It's hard-wired (Located on solder side). In case the U12 is damaged, VCC and VSS may be internally shorted, by cutting this jumper the board may be able to work again without the H-Bridge.

J71 FTDI header

J72 Force the HCS12 into reset. Don't place a jumper on this header when developing HCS12 programs.

J76 4x4 keypad header for an external membrane keypad.

# Chapter 5: CodeWarrior and Serial monitor

CodeWarrior is a very powerful and professional IDE. The main feature of CodeWarrior IDE is the source level debugger in assembler and C. CodeWarrior Special Edition is a wonderful gift from Freescale to all of us and it's free for educational use. What's more, by CodeWarrior supporting serial monitor, they have made it very affordable to support CodeWarrior for OEMs.

Freescale has invested millions of dollar into CodeWarrior and the current versions work very well. Freescale knows they will never sell enough copies of CodeWarrior to make back what they have invested. They did it to drive chip sales.

As a software developer, the first thing you look at is available tools and what it will cost. There are many companies making MCU chips nowadays and for the most part they all have about the same features at a similar price. Special Edition CodeWarrior sets Freescale apart from others.

CodeWarrior IDE does not work with D-Bug12, but it works with serial monitor. Before Freescale created the serial monitor a BDM is needed as an interface between the PC and HCS12. Freescale created the serial monitor for working with CodeWarrior to eliminate the cost of a BDM.

Now a student can use the serial monitor with CodeWarrior to debug his program and in fact, many universities have been using the serial monitor with CodeWarrior without a BDM in their classrooms.

Without spending money on a BDM, a student will be able to spend his savings on purchasing a more advanced trainer, like the Dragon12_light_d board with many on-board peripherals. Purchasing an EVB board that comes with a BDM at a reasonable price, most likely leaves the student with an EVB of only limited functionality.

Some universities use D-Bug12 monitor first, then replace the D-Bug12 monitor with serial monitor to be used with CodeWarrior IDE later. In this case, a school laboratory only needs to have one BDM or use one dragon12_light_d board as a BDM POD, to program all students' boards with serial monitor.

To replace bootloader and D-Bug12 monitor with serial monitor, you need a BDM or a BDM POD to perform the task. The procedure to program the on-chip flash memory is shown at:

http://www.trainer4edu.com/freescale_usbdm_osbdm/usbdm_osbdm_bdm_multilink.html

Some universities use CodeWarrior IDE only. In this case, we pre-load the on-chip flash memory with serial monitor.

For more information on CodeWarrior please visit:

http://www.trainer4edu.com/dragon12/codewarrior_hcs12.html

# Chapter 6: PLL code

```
;     The crystal frequency on the Dragon12_light_d board is 8 MHz so the default bus speed is
;     4 MHz.  In order to set the bus speed high than 4 MHz the PLL must be initialized.
;
;     You can cut and paste the following code to the beginning of your program.
;
;           The math used to set the PLL frequency is:
;
;           PLLCLK = CrystalFreq * 2 * (initSYNR+1) / (initREFDV+1)
;
;           CrystalFreq =     8 MHz on Dragon12_light_d board
;           initSYNR    =     5,  PLL multiplier will be 6
;           initREFDV   =     1,  PLL divisor will be 2
;           PLLCLK = 8*2*6/2 = 48MHz
;           The bus speed = PLLCLK / 2 = 24 MHz
;
;Assembly code
;
start:                                  ; PLL code for 24MHz bus speed from a 4/8/16 crystal
            sei
            ldx     #0
            bclr    clksel,x,%10000000      ; clear bit 7, clock derived from osccclk
            bset    pllctl,x,  %01000000    ; Turn PLL on, bit 6 =1 PLL on, bit 6=0 PLL off
            ldaa    #$05                    ; 5+1=6 multiplier
            staa    synr,x
  ;         ldaa    #$03    ; divisor=3+1=4, 16*2*6 /4 =  48MHz PLL freq, for 16 MHz crystal
            ldaa    #$01    ; divisor=1+1=2,   8*2*6 /2 =  48MHz PLL freq, for 8 MHz crystal
  ;         ldaa    #$00    ; divisor=0+1=1,   4*2*6 /1 =  48MHz PLL freq, for 4 MHz crystal

            staa    refdv,x
wait_b3:    brclr   crgflg,x, %00001000 wait_b3     ; Wait until bit 3 = 1
            bset    clksel,x, %10000000

        /* C code */

void set_clock_24mhz(void)
{
   CLKSEL &= 0x7F;
   PLLCTL |= 0x40;
   SYNR  = 0x05;
   REFDV = 0x01;

/* REFDV=0x00;   for 4 MHz */
/* REFDV=0x01;   for 8 MHz  */
/* REFDV=0x03;   for 16 MHz */

   while(!(0x08 & CRGFLG));
   CLKSEL |= 0x80;
}
```

# Chapter 7: Appendix

## 7.1 D-Bug12 utility routines

The AN1280 was written for OLD 68HC12 family. If you happen to use printf routine with your old 68HC12 board you should be aware that I/O utility routines are moved to different addresses in D-Bug12 V4.x.

The address for the printf is $EE88 and addresses of other I/O routines are listed below:

| Function | Description | Pointer Address |
|---|---|---|
| `far main()` | Start of D-Bug12 | $EE80 |
| `getchar()` | Get a character from SCI0 or SCI1 | $EE84 |
| `putchar()` | Send a character out SCI0 or SCI1 | $EE86 |
| `printf()` | Formatted Output - Translates binary values to characters | $EE88 |
| `far GetCmdLine()` | Obtain a line of input from the user | $EE8A |
| `far sscanhex()` | Convert an ASCII hexadecimal string to a binary integer | $EE8E |
| `isxdigit()` | Checks for membership in the set [0..9, a..f, A..F] | $EE92 |
| `toupper()` | Converts lower case characters to upper case | $EE94 |
| `isalpha()` | Checks for membership in the set [a..z, A..Z] | $EE96 |
| `strlen()` | Returns the length of a null terminated string | $EE98 |
| `strcpy()` | Copies a null terminated string | $EE9A |
| `far out2hex()` | Displays 8-bit number as 2 ASCII hex characters | $EE9C |
| `far out4hex()` | Displays 16-bit number as 4 ASCII hex characters | $EEA0 |
| `SetUserVector()` | Setup user interrupt service routine | $EEA4 |
| `far WriteEEByte()` | Write a data byte to on-chip EEPROM | $EEA6 |
| `far EraseEE()` | Bulk erase on-chip EEPROM | $EEAA |
| `far ReadMem()` | Read data from the M68HC12 memory map | $EEAE |
| `far WriteMem()` | Write data to the M68HC12 memory map | $EEB2 |

**Fig 8-1: D-Bug12 utility routines**

## 7.2 Interrupt vector table

**Table 5-1 Interrupt Vector Locations**

| Vector Address | Interrupt Source | CCR Mask | Local Enable | HPRIO Value to Elevate |
|---|---|---|---|---|
| $FFFE, $FFFF | Reset | None | None | – |
| $FFFC, $FFFD | Clock Monitor fail reset | None | PLLCTL (CME, SCME) | – |
| $FFFA, $FFFB | COP failure reset | None | COP rate select | – |
| $FFF8, $FFF9 | Unimplemented instruction trap | None | None | – |
| $FFF6, $FFF7 | SWI | None | None | – |
| $FFF4, $FFF5 | XIRQ | X-Bit | None | – |
| $FFF2, $FFF3 | IRQ | I-Bit | IRQCR (IRQEN) | $F2 |
| $FFF0, $FFF1 | Real Time Interrupt | I-Bit | CRGINT (RTIE) | $F0 |
| $FFEE, $FFEF | Enhanced Capture Timer channel 0 | I-Bit | TIE (C0I) | $EE |
| $FFEC, $FFED | Enhanced Capture Timer channel 1 | I-Bit | TIE (C1I) | $EC |
| $FFEA, $FFEB | Enhanced Capture Timer channel 2 | I-Bit | TIE (C2I) | $EA |
| $FFE8, $FFE9 | Enhanced Capture Timer channel 3 | I-Bit | TIE (C3I) | $E8 |
| $FFE6, $FFE7 | Enhanced Capture Timer channel 4 | I-Bit | TIE (C4I) | $E6 |
| $FFE4, $FFE5 | Enhanced Capture Timer channel 5 | I-Bit | TIE (C5I) | $E4 |
| $FFE2, $FFE3 | Enhanced Capture Timer channel 6 | I-Bit | TIE (C6I) | $E2 |
| $FFE0, $FFE1 | Enhanced Capture Timer channel 7 | I-Bit | TIE (C7I) | $E0 |
| $FFDE, $FFDF | Enhanced Capture Timer overflow | I-Bit | TSRC2 (TOF) | $DE |
| $FFDC, $FFDD | Pulse accumulator A overflow | I-Bit | PACTL (PAOVI) | $DC |
| $FFDA, $FFDB | Pulse accumulator input edge | I-Bit | PACTL (PAI) | $DA |
| $FFD8, $FFD9 | SPI0 | I-Bit | SP0CR1 (SPIE, SPTIE) | $D8 |
| $FFD6, $FFD7 | SCI0 | I-Bit | SC0CR2 (TIE, TCIE, RIE, ILIE) | $D6 |
| $FFD4, $FFD5 | SCI1 | I-Bit | SC1CR2 (TIE, TCIE, RIE, ILIE) | $D4 |
| $FFD2, $FFD3 | ATD0 | I-Bit | ATD0CTL2 (ASCIE) | $D2 |
| $FFD0, $FFD1 | ATD1 | I-Bit | ATD1CTL2 (ASCIE) | $D0 |
| $FFCE, $FFCF | Port J | I-Bit | PTJIF (PTJIE) | $CE |
| $FFCC, $FFCD | Port H | I-Bit | PTHIF(PTHIE) | $CC |
| $FFCA, $FFCB | Modulus Down Counter underflow | I-Bit | MCCTL(MCZI) | $CA |

**Fig 8-2: MC9S12DG256 Interrupt vector table 1**

| | | | | |
|---|---|---|---|---|
| $FFC8, $FFC9 | Pulse Accumulator B Overflow | I-Bit | PBCTL(PBOVI) | $C8 |
| $FFC6, $FFC7 | CRG PLL lock | I-Bit | CRGINT(LOCKIE) | $C6 |
| $FFC4, $FFC5 | CRG Self Clock Mode | I-Bit | CRGINT (SCMIE) | $C4 |
| $FFC2, $FFC3 | BDLC | I-Bit | DLCBCR1(IE) | $C2 |
| $FFC0, $FFC1 | IIC Bus | I-Bit | IBCR (IBIE) | $C0 |
| $FFBE, $FFBF | SPI1 | I-Bit | SP1CR1 (SPIE, SPTIE) | $BE |
| $FFBC, $FFBD | SPI2 | I-Bit | SP2CR1 (SPIE, SPTIE) | $BC |
| $FFBA, $FFBB | EEPROM | I-Bit | EECTL(CCIE, CBEIE) | $BA |
| $FFB8, $FFB9 | FLASH | I-Bit | FCTL(CCIE, CBEIE) | $B8 |
| $FFB6, $FFB7 | CAN0 wake-up | I-Bit | CAN0RIER (WUPIE) | $B6 |
| $FFB4, $FFB5 | CAN0 errors | I-Bit | CAN0RIER (CSCIE, OVRIE) | $B4 |
| $FFB2, $FFB3 | CAN0 receive | I-Bit | CAN0RIER (RXFIE) | $B2 |
| $FFB0, $FFB1 | CAN0 transmit | I-Bit | CAN0TIER (TXEIE2-TXEIE0) | $B0 |
| $FFAE, $FFAF | CAN1 wake-up | I-Bit | CAN1RIER (WUPIE) | $AE |
| $FFAC, $FFAD | CAN1 errors | I-Bit | CAN1RIER (CSCIE, OVRIE) | $AC |
| $FFAA, $FFAB | CAN1 receive | I-Bit | CAN1RIER (RXFIE) | $AA |
| $FFA8, $FFA9 | CAN1 transmit | I-Bit | CAN1TIER (TXEIE2-TXEIE0) | $A8 |
| $FFA6, $FFA7 | CAN2 wake-up | I-Bit | CAN2RIER (WUPIE) | $A6 |
| $FFA4, $FFA5 | CAN2 errors | I-Bit | CAN2RIER (CSCIE, OVRIE) | $A4 |
| $FFA2, $FFA3 | CAN2 receive | I-Bit | CAN2RIER (RXFIE) | $A2 |
| $FFA0, $FFA1 | CAN2 transmit | I-Bit | CAN2TIER (TXEIE2-TXEIE0) | $A0 |
| $FF9E, $FF9F | CAN3 wake-up | I-Bit | CAN3RIER (WUPIE) | $9E |
| $FF9C, $FF9D | CAN3 errors | I-Bit | CAN3RIER (TXEIE2-TXEIE0) | $9C |
| $FF9A, $FF9B | CAN3 receive | I-Bit | CAN3RIER (RXFIE) | $9A |
| $FF98, $FF99 | CAN3 transmit | I-Bit | CAN3TIER (TXEIE2-TXEIE0) | $98 |
| $FF96, $FF97 | CAN4 wake-up | I-Bit | CAN4RIER (WUPIE) | $96 |
| $FF94, $FF95 | CAN4 errors | I-Bit | CAN4RIER (CSCIE, OVRIE) | $94 |
| $FF92, $FF93 | CAN4 receive | I-Bit | CAN4RIER (RXFIE) | $92 |
| $FF90, $FF91 | CAN4 transmit | I-Bit | CAN4TIER (TXEIE2-TXEIE0) | $90 |
| $FF8E, $FF8F | Port P Interrupt | I-Bit | PTPIF (PTPIE) | $8E |
| $FF8C, $FF8D | PWM Emergency Shutdown | I-Bit | PWMSDN (PWMIE) | $8C |
| $FF80 to $FF8B | Reserved | | | |

**Fig 8-3: MC9S12DG256 Interrupt vector table 2**

| Interrupt Source | Secondary Vector Address | Interrupt Source | Secondary Vector Address |
|---|---|---|---|
| Reserved $FF80 | $EF80 | $I^2C$ bus | $EFC0 |
| Reserved $FF82 | $EF82 | DLC | $EFC2 |
| Reserved $FF84 | $EF84 | SCME | $EFC4 |
| Reserved $FF86 | $EF86 | CRG lock | $EFC6 |
| Reserved $FF88 | $EF88 | Pulse accumulator B overflow | $EFC8 |
| Reserved $FF8A | $EF8A | Modulus down counter underflow | $EFCA |
| PWM emergency shutdown | $EF8C | Port H interrupt | $EFCC |
| Port P interrupt | $EF8E | Port J interrupt | $EFCE |
| MSCAN 4 transmit | $EF90 | ATD1 | $EFD0 |
| MSCAN 4 receive | $EF92 | ATD0 | $EFD2 |
| MSCAN 4 errors | $EF94 | SCI1 | $EFD4 |
| MSCAN 4 wakeup | $EF96 | SCI0 | $EFD6 |
| MSCAN 3 transmit | $EF98 | SPI0 | $EFD8 |
| MSCAN 3 receive | $EF9A | Pulse accumulator A input edge | $EFDA |
| MSCAN 3 errors | $EF9C | Pulse accumulator A overflow | $EFDC |
| MSCAN 3 wakeup | $EF9E | Timer overflow | $EFDE |
| MSCAN 2 transmit | $EFA0 | Timer channel 7 | $EFE0 |
| MSCAN 2 receive | $EFA2 | Timer channel 6 | $EFE2 |
| MSCAN 2 errors | $EFA4 | Timer channel 5 | $EFE4 |
| MSCAN 2 wakeup | $EFA6 | Timer channel 4 | $EFE6 |
| MSCAN 1 transmit | $EFA8 | Timer channel 3 | $EFE8 |
| MSCAN 1 receive | $EFAA | Timer channel 2 | $EFEA |
| MSCAN 1 errors | $EFAC | Timer channel 1 | $EFEC |
| MSCAN 1 wakeup | $EFAE | Timer channel 0 | $EFEE |
| MSCAN 0 transmit | $EFB0 | Real-time interrupt | $EFF0 |
| MSCAN 0 receive | $EFB2 | IRQ | $EFF2 |
| MSCAN 0 errors | $EFB4 | XIRQ | $EFF4 |
| MSCAN 0 wakeup | $EFB6 | SWI | $EFF6 |
| FLASH | $EFB8 | Unimplemented instruction trap | $EFF8 |
| EEPROM | $EFBA | COP failure reset | $EFFA |
| SPI2 | $EFBC | Clock monitor fail reset | $EFFC |
| SPI1 | $EFBE | Reset | $EFFE |

**Fig 8-4: MC9S12DG256 secondary interrupt vector table**