Application Note

AN2292
6/2002

8-Bit Software Development Kit

By Jiri Ryba

Introduction

This application note describes the features and advantages of the 8-bit SDK (software development kit) to help users quickly create their own applications. The intended audience is developers who are starting their first application using the 8-bit SDK.

8-Bit SDK Overview

The Motorola Embedded Software Development Kit (8-bit SDK) has been created to complement the existing development environment for Motorola M68HC08 embedded processors. It provides a software infrastructure that allows fast and efficient development of software applications, which are portable and reusable among different M68HC08 Family MCUs. The M68HC08 SDK development environment contains peripheral drivers, algorithms, examples, and interfaces that allow programmers to create their own C application code, independent of the core architecture. The introductory SDK components are oriented towards motor control applications but the SDK is by no means limited to these.

This SDK has been tested with compilers from Metrowerks and Cosmic. It is expected that the drivers and algorithms within can be readily ported to other environments.

The purpose of the 8-bit SDK is to:

- Speed up the process of application development
- Decrease requirements for knowledge of MCU peripherals
- Simplify development of structured software
- Achieve high software efficiency — higher-speed operation/lower-memory consumption
- Achieve high transparency and easy modification of the finalized code

8-Bit SDK Features

The M68HC08 SDK environment is composed of the following major components (see **Figure 1**):

- Core-system infrastructure
- On-chip drivers (drivers for built-in peripherals)
- Off-chip drivers (drivers for external hardware)
- Example applications
- Libraries with math functions and algorithms

The basic idea of the 8-bit SDK is the separation of the application software from the hardware of the microcontroller. The hardware peripherals are controlled only through the dedicated drivers. Thus, layered and structured software can be developed. Highest attention is given to code efficiency.

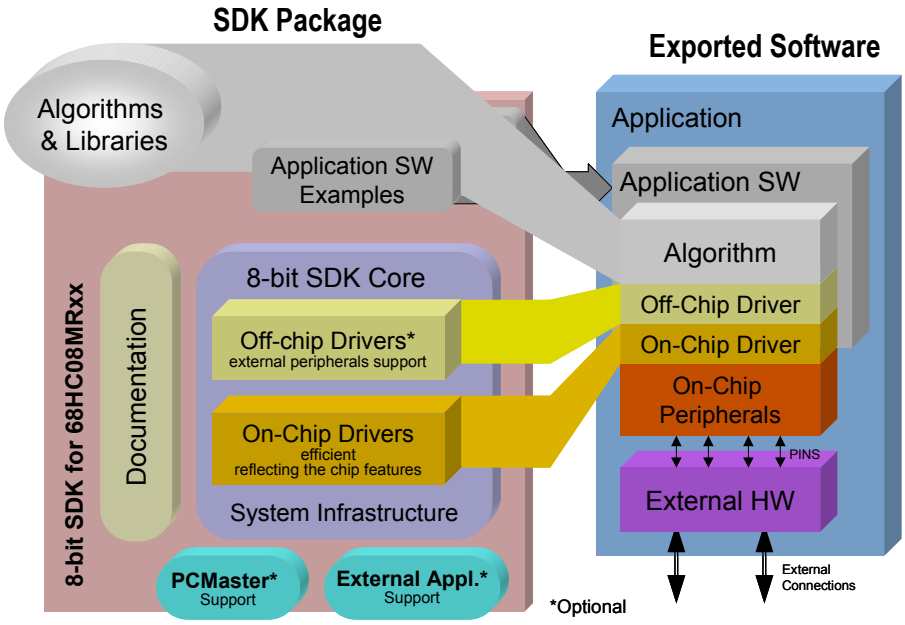

Figure 1. 8-Bit SDK Architecture

The 8-bit SDK core drivers and algorithms are intended to provide a high level of code efficiency, comparable with pure assembly language code.

All the components of the 8-bit SDK use a C compatible interface.

This section provides an overview of the SDK components, while a comprehensive description can be found in the 8-bit SDK M68HC08 target documentation (see **References**).

Core-System Infrastructure

The core-system infrastructure creates the fundamental infrastructure for M68HC08 device operation and enables further integration with the 8-bit SDK components. The provided basic development support includes:

- Commonly used macro definitions
- Portable architecture-dependent register declaration
- Generic types
- A mechanism for static configuration of on-chip peripherals
- Interrupt vectors
- Project templates

Static Initialization

The 8-bit SDK uses static initialization of the peripheral configuration. The default configuration of the peripherals corresponds to the default RESET values of the microcontroller. The default configuration is stored in the header files of the peripheral drivers, such as “pwmdrv.h” for the PWM driver, “spidrv.h” for the SPI driver, etc. You can modify any of the default peripheral registers using the application-specific file “appconfig.h”. Then, the configuration bits of the individual registers are composed according to the required setting in the “appconfig.h”. When the configuration of the register is equal to its default value, no code is generated and the original RESET content of the register is preserved. Finally, the generated configuration is stored in the configuration file “config.c” that is used during the initialization of the microcontroller. The static initialization is depicted in **Figure 2**.

Such an approach allows the easy initialization of the on-chip peripherals using just the application-specific file “appconfig.h”. You do not need to study the placement and syntax of the individual bits of the configuration registers of on-chip peripherals. Any modification of the settings can easily be done in the application-specific file without any accidental side effects.

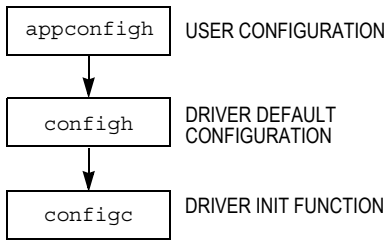

Figure 2. Static Initialization of On-Chip Peripherals

On-Chip Drivers

The on-chip drivers isolate the hardware-specific functionality from user software by a set of driver commands with a defined API (application programming interface). The API standardizes the interface between the software and the hardware, see **Figure 3**. Application software accesses peripherals only through the on-chip drivers. This isolation enables a high degree of portability or architectural and hardware independence for application code. This is mainly valid for devices with similar peripheral modules.

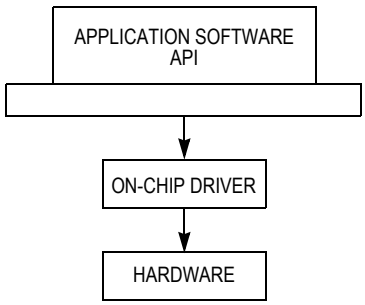

Figure 3. Software Structure

On-chip driver characteristics include:

- Dedicated to control of all on-chip peripherals
- Low-level drivers implemented as efficient macros
- Driver API clearly suggests its intended purpose
- Access to all HW features of the individual on-chip peripherals
- Static configuration
- Emphasis on efficiency — performance and memory
- Tested functionality
- Application examples available for each driver

Off-Chip Drivers

The off-chip drivers have the same functionality as the on-chip drivers. The only difference is that the off-chip drivers perform the interface between the software and a peripheral which is not on the chip, but which can be controlled by the chip. Off-chip drivers can access the hardware by means of the on-chip drivers. This isolation enables a high degree of portability or architectural and hardware independence for both application and off-chip drivers code. This is mainly valid for standard external peripherals, such as a PC, display, keyboard, switch, LED, etc.

Algorithms

The 8-bit SDK provides a set of dedicated algorithms, ready to use in the target application. The algorithm library contains, for example, a Math Library and a Motor Control Library. All algorithms are independent of drivers and can be used for all chips with the same 68HC08 core. For more detail, see the 8-bit SDK Algorithm Libraries.

PC Master Software

PC master software is one of the off-chip drivers, which supports communication between the target microcontroller and PC. This tool was initially created for developers of motor control applications, but it may be extended to any other application development. This tool allows the programmer to remotely control an application using a user-friendly graphical environment that is running on a PC. It also provides the ability to view some real-time application variables in both text and graphic form.

Main features:

- Graphic environment
- Visual Basic Script or Java Script can be used to control the target board
- Easy to understand navigation
- Connection to target board is possible over a network, including the Internet
- Demo mode with support for password protection
- Visualization of real-time data in the scope window
- Acquisition of fast data changes using the integrated recorder
- Value interpretation using custom defined text messages
- Built-in support for standard variable types (integer, floating point, bit fields)
- Several built-in transformations for real type variables
- Automatic variable extraction from Metrowerks' CodeWarrior® linker[1] output files (MAP, ELF)
- Remote control of application execution

PC master software is a versatile tool to be used for multipurpose algorithms and applications. It provides a lot of excellent features, including:

- Real-time debugging
- Diagnostic tool
- Demonstration tool
- Education tool

A full description can be found in the PC master software User's Manual and in the dedicated application notes (see **References**).

1. CodeWarrior is a registered trademark of Metrowerks, a Motorola company.

Interrupts

The 8-bit SDK supports two types of interrupt callbacks.

- Type 1 first calls the user function and then the SDK interrupt service routine.
- Type 2 first calls the SDK interrupt service routine and then the user function.

The SDK interrupt service routine provides service of the interrupt according to its definition. It clears the interrupt flags, so you do not have to worry about the interrupt flags in your code. The SDK drivers can call the user's interrupt routine automatically. For more detail, check the appropriate driver documentation. An example of an SDK driver is the analog-to-digital converter (ADC) driver. The driver starts the ADC and when the conversion is completed, it buffers the converted input and starts the conversion again for the following ADC channel. In this example, the service of the interrupt "Conversion Completed" is served automatically by the SDK.

On the other hand, most of the low-level interrupt drivers only clear the interrupt flags. You can disable clearing of the flags.

Quick Start with the 8-Bit SDK

This section describes the installation and use of the individual components of the 8-bit SDK from a practical point of view. This example is for Metrowerks CodeWarrior.

Install the 68HC08 SDK

Before the 8-bit SDK can be used, it needs to be installed on your PC. For installation of the 8-bit SDK you need to:

1. Execute Setup.exe.
2. Follow the 68HC08 SDK software installation instructions on your screen.
3. Copy the 8-bit SDK stationery to the Metrowerks stationery folder.
4. Set the path to the 68HC08 SDK source within IDE.
   a. Launch CodeWarrior IDE from the Start->Programs->Metrowerks CodeWarrior menu.
   b. Open the IDE Preferences dialog window using Edit->Preferences.
   c. Select the Source Trees panel from the IDE Preferences Panels->General.
   d. Type 68HC08 SDK src into the Name box.
   e. Choose Absolute Path as the path type.
   f. Click Choose and locate the 68HC08 SDK installation directory, e.g. C:\Program Files\Motorola\68HC08 SDK\src.
   g. Click Add.
   h. Click OK to finish.

Create Project

Follow these steps to create a new project:

1. In the menu File-New, select 68HC08 SDK Stationery and set your project name and path, where the application will be placed.
2. Select the required device and required template. This creates a new project (see **Figure 4**) including:
   - Start-up code for the target device
   - Appropriate linker command file “default.prm”
   - Template of main routine `main()` in “main.c”
   - Architecture dependent definitions “arch.h
   - On-chip driver commands
   - Application specific configuration file “appconfig.h”

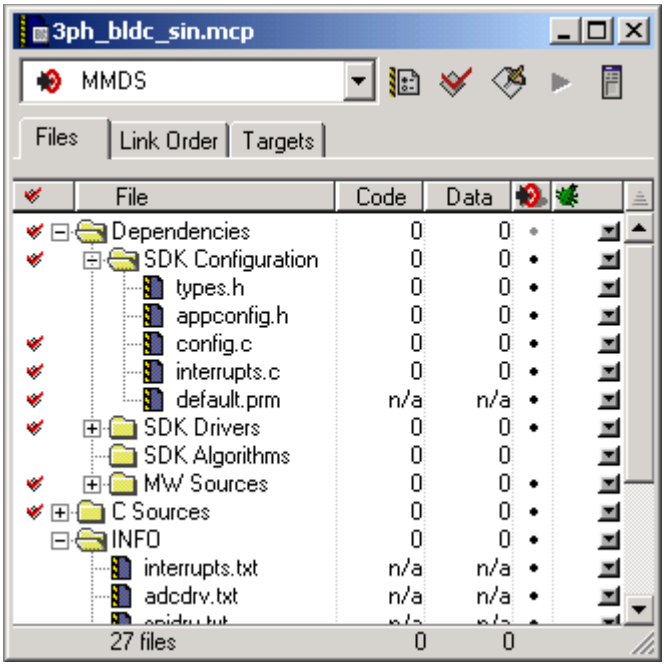

Figure 4. New Project Created from the 68HC08 SDK Stationery

Static Initialization

The static initialization of the on-chip drivers is set in the “appconfig.h” file. Follow these steps to configure the on-chip peripherals:

1. Copy the configuration items from the *driver “name*.txt” e.g. “pwmdrv.txt” to the “appconfig.h” file. The files can be found in the project window (see **Figure 4**).
2. In the “appconfig.h” file select the appropriate value for the driver constants. The possible value are in comments. The individual options are described in the 8-bit SDK User’s Manual. E.g., to define PWM reload frequency use:

```
#define PWM_RELOAD_FREQUENCY PWM_EVERY_4_CYCLE
                                              /*PWM_EVERY_1_CYCLE*/
                                              /*PWM_EVERY_2_CYCLE*/
                                              /*PWM_EVERY_4_CYCLE*/
                                              /*PWM_EVERY_8_CYCLE*/
```

3. Include the desired driver by defining INCLUDE_*driver* in “appconfig.h”; e.g., to include PWM driver use:

```
#define INCLUDE_PWM
```

4. Call Initialization function in the `main()`:

```
void main (void)
{
     /* Static Initialization of periphery */
     (void) peripheralInit();
}
```

NOTE: *Unchanged constants in the “appconfig.h” do not bring any code overhead. Only non-default values are written to registers. If a constant is not defined in the “appconfig.h”, then the default value, defined in the driver header file, is used.*

On-Chip Drivers

On-chip peripherals are controlled by the IOCTL driver commands. The general form of the driver command is the following:

`IOCTL(`*`peripheral_module_identifier`*`,`*`command`*`,`*`command_specif_parameter`*`);`

Where:

The *peripheral_module_identifier* parameter specifies the peripheral module with predefined symbolic constants, like **PWM** for PWM module, **TIMA** for Timer A, **TIMB** for Timer B etc.

The *command* parameter specifies the action, which will be performed on the peripheral module. It represents the command name as it is implemented for each on-chip driver.

The *command_specific_parameter* parameter specifies other data required to execute the command. Generally speaking, it can be a pointer to the structure, the “NULL” value or a variable value dependent on the specific command. If the required parameter is a variable value, you should use a constant value if possible, because it influences the efficiency of the resulting code. The efficiency is illustrated by the following examples:

Using of the compilation constant as a command specific parameter (written in “C”):

```
/* Write 1 to CH1IE in TBSC1
(Enable Interrupt for channel 1 of Timer B) */
IOCTL(TIMB,TIM_SET_CH1_INT,TIM_ENABLE);
```

assembly code as compilation result:

```
                BSET      6,89
```

Using a variable as a command specific parameter (written in “C”):

```
/* Write varU8 to CH1IE in TBSC1
(If varU8, Enable Interrupt for channel 1 of Timer B) */
IOCTL(TIMB,TIM_SET_CH1_INT,varU8);
```

assembly code as compilation result:

```
                LDA       varU8
                BIT       #1
                BNE       L1673 ;abs = 1673
                BCLR      6,89
                SKIP2     L1675 ;abs = 1675
L1673:          BSET      6,89
L1675:
```

If the parameter in this example is constant then the command is compiled to one assembly instruction. If the parameter is variable the parameter value is tested at execution time resulting in longer code.

Algorithms

To use an algorithm from the SDK library, add the source file of the algorithm to your project and include the appropriate header file in your source code.

For example, if you are going to use a controller, add the “mccontrollers.c” to your project file (*.mcp) and put the

```
#include mccontrollers.h
```

in your source file.

Available algorithms with detailed descriptions can be found in the “Algorithm Library” documentation.

Interrupts

Interrupt Callbacks

You can assign any application function to an interrupt by callback definition in "appconfig.h":

```
#define INT_PWM_RELOAD_CALLBACK_1 IsrPWM_Reload
```

In this example the "`IsrPWM_Reload`" is the name of the user function. This name can be an arbitrary string. The "`INT_PWM_RELOAD_CALLBACK_1`" determines in which interrupts the user function will be called and the type of the callback. In this case the PWM interrupt callback type 1 is defined by the suffix _1 at the end. It means that the `IsrPWM_Reload()` user function will be called first, then the PWM Interrupt flag is cleared.

You can also define their own callback after the 68HC08 SDK ISR

```
#define INT_PWM_RELOAD_CALLBACK_2 IsrPWM_Reload
```

In this example the PWM Interrupt flag will be cleared and then the `IsrPWM_Reload()` user function will be called.

Interrupt Flag Service

Most interrupts use flags, which signal the request of the interrupt. New interrupt cannot be requested until the flag is cleared. The constant `INT_FlagName_FLAG` defined in the "appconfig.h", determines if the flag is cleared by the 68HC08 SDK or if you will take care of this flag by yourself:

```
#define INT_PWM_RELOAD_CALLBACK_1 IsrPWM_Reload
#define INT_PWM_RELOAD_FLAG CLEAR_USER
```

This definition leaves the flag service to you. In this case, you are responsible for servicing the interrupt flag.

NOTE: *The callback definition can be copied from the "callback.txt" file.*

Interrupt Debug Strobes

The IDS (interrupt debug strobes) can be used during the development of the application. It indicates the interrupt duration to the developer. The selected I/O pin is set at the beginning of the interrupt and cleared at the end of the interrupt. The required I/O port and the pin have to be defined in the "appconfig.h" by the definition:

```
/* definition of PORT for debug signal */
#define INT_InterruptName_STROBE_PORT PORTx
```

and

```
/* definition of PIN for debug signal on specified port */
#define INT_InterruptName_STROBE_PIN n
```

where "x" is the port identifier (A,B,C,E,F) and the "n" is the number of the pin (0,1,2,3,4,5,6,7).

For example, your definition of the debug strobe for the PWM interrupt can be:

```
/* definition of PORTB for debug signal */
#define INT_PWM_RELOAD_STROBE_PORT  PORTB
/* definition of PIN 4 for debug signal on specified port */
#define INT_PWM_RELOAD_STROBE_PIN   4
```

Interrupt Debug Mode

The IDM (interrupt debug mode) provides the service for all unhandled interrupts. If you define `INT_DEBUG_MODE TRUE` in “appconfig.h”, then jumps to a never-ending loop will be added to all of the unhandled interrupts, such as:

```
#define INT_DEBUG_MODE  TRUE /* Allow never-end loop in unhandled INT */
```

By default IDM is disabled

```
#define INT_DEBUG_MODE  FALSE
```

PC Master Software

PC master software runs on a PC, connected with the target processor via an RS232 serial interface. Make sure that the SCI baud rate and application bus speed constants in the “pcmastersw.c” and “appconfig.h” files are configured to match the PC communication rate.The default baud rate is 9600 baud for an 8 MHz bus clock. A small program, resident in the target processor, communicates with the PC master software. It enables access to any memory location of the target processor in real time. This section helps you install the PC master software and include the PC master software in your application.

PC Requirements

PC master software can run on any computer with a Microsoft® Windows® operating system and Internet Explorer® 4.0 or higher installed.[1]

The following requirements result from those for the Internet Explorer 4.0 application:

Computer: 486DX/66 MHz or higher processor (Intel® Pentium® recommended)[2]

Operating system: Microsoft Windows NT®, Windows 98 or Windows 95 + DCOM pack

Memory: Windows 95 or 98: 16 MB RAM minimum (32 Mb recommended); for Windows NT: 32 MB RAM minimum (64 MB recommended)

Required software: Internet Explorer 4.0 or higher installed, Metrowerks CodeWarrior (or similar)

Hard drive space: 6 MB

1. Microsoft, Windows, NT, and Internet Explorer are registered trademarks of Microsoft Corporation in the United States and/or other countries.
2. Intel and Pentium are registered trademarks of Intel Corporation.

Other hardware requirements: Mouse, serial RS-232 port for local control, network access for remote control

PC master software is an optional part of the 8-bit SDK environment and must be installed separately running the PC master software “Setup.exe”.

Enabling PC Master Software on the Target Application

To enable the PC master software operation on the target board application, add the “pcmastersw.c” to your project and define

```
#define INCLUDE_PCMASTERSW /* allow PC master software installation */
```

in your “appconfig.h” file.

NOTE: *If a RAM overflow occurs during compilation or if your application is time critical, you can also use the assembly version of PC master software. In this case, add “pclink.c and pclink.asm and pclinkvar.asm to your project and define*

```
/*allow PC master software assembly version installation */
#define INCLUDE_PCLINK
```

in your “appconfig.h” file.

Using PC Master Software

The following steps help you use PC master software for your own application:

1. Connect your application to the PC via the RS232 port.
2. Switch on your application.
3. Start the PC master software from the Start Menu.
4. Set your COM port number in the menu Project / Options / MCB Comm.
5. Check the communication speed. The default SCI baud rate is 9600 baud. If your application uses other baud rate, set the Speed property in the menu Project / Options / MCB Comm.
6. Set your MAP file in the menu Project / Options / MAP Files (If the application is compiled in Hiware object file format, use the “*.map” file. If the application is compiled in ELF/DWARF format, use the “*.abs” file. The compilation setting can be found in the menu Edit / Target Setting... / Compiler / Options / Output / Object File Format).

References

- 8-bit SDK 68HC908MR32 Targeting Documentation, Motorola, 2002
- 8-bit SDK Algorithm Libraries, Motorola, 2002
- 8-bit SDK Application Targeting Documentation, Motorola 2002
- Embedded PC Master Application, Motorola, 2002
- Motorola SPS web page: http://motorola.com/semiconductors

This page intentionally left blank.

This page intentionally left blank.

This page intentionally left blank.

HOW TO REACH US:

USA/EUROPE/LOCATIONS NOT LISTED:

Motorola Literature Distribution;
P.O. Box 5405, Denver, Colorado 80217
1-303-675-2140 or 1-800-441-2447

JAPAN:

Motorola Japan Ltd.; SPS, Technical Information Center,
3-20-1, Minami-Azabu Minato-ku, Tokyo 106-8573 Japan
81-3-3440-3569

ASIA/PACIFIC:

Motorola Semiconductors H.K. Ltd.;
Silicon Harbour Centre, 2 Dai King Street,
Tai Po Industrial Estate, Tai Po, N.T., Hong Kong
852-26668334

TECHNICAL INFORMATION CENTER:

1-800-521-6274

HOME PAGE:

http://www.motorola.com/semiconductors