IAR Application Note
Migrating from the previous product

SUMMARY

This application note describes how to modify source code written for the AT90S IAR Compiler in order to use it with the AVR IAR Compiler.

KEYWORDS

ATMEL AT90Sxxx, C, AVR, migrate.

Background

Your application was originally written for the AT90S IAR Compiler, but now you want to take advantage of the brand new features of the AVR IAR Compiler, which is based on the latest IAR compiler technology. This technology allows you to enhance your application code in a way that was previously not possible. The highlights are that there is:

- Availability of embedded C++.
- A new global optimizer, which improves the efficiency of the generated code.
- Stronger adherence to the ISO/ANSI standard; for example, it is possible to use #pragma directives instead of extended keywords for defining special function registers (SFRs). In addition, the checking of data types adheres more strictly to the ISO/ANSI standard in the AVR IAR Compiler than in products using a previous generation of compiler technology.

  The AVR IAR Compiler generates more efficient object code than previously possible. It is, however, required that you make some modifications to your existing source code before compiling it with the new AVR IAR Compiler, which may otherwise generate warnings or error messages.

The Modifications Required

The following aspects of the source code are affected by the new compiler technology and therefore, need modification:

- Extended keywords
- #pragma directives

- Intrinsic functions
- Startup sequence Mainly, it is the notation of the language extensions that has changed. In some cases, the behavior and scope is also different. For example, some extended keywords and `#pragma` directives have been removed, while new ones have been added.

  The new compiler technology also introduces new compiler options. This means that the project configuration in the IAR Embedded Workbench differs between the AT90S IAR Compiler and the AVR IAR Compiler. It is therefore very important that you install the AVR IAR Compiler toolkit in a separate directory and that you keep all source files and project files apart. You will then be able to continue using the AT90S IAR Compiler if you so wish.

  Note: The mechanisms that control the initialization of segments differ completely between the two products. This means that you cannot use a `CSTARTUP` file from the AT90S IAR Compiler in an AVR IAR Compiler project.

The Solution

IAR Systems offers the following solutions:

- The file `comp_a90.h`, which is included with the product, helps you get your AVR project up and running quickly. It contains translation macros that facilitate the migration by taking care of changes in the notation such as new names on intrinsic functions and double underscores on extended keywords; for example, `__near` instead of `near`.
- A comprehensive and detailed chapter, *Migrating to the AVR IAR Compiler* in the *AVR IAR Compiler Reference Guide,* is available for reference.
- A step-by-step process is clearly outlined in order for you to verify that you have covered the required steps.

The Migration Process

1. Install the IAR Embedded Workbench toolkit with the AVR IAR Compiler toolkit in a directory separate from the AT90S IAR Compiler. Make sure that all your source code files and project files are kept apart.
2. Create a new project and copy your existing source code files to it.
3. Examine the use of doubles in the existing source code.

   The AT90S IAR Compiler supports only 4-byte doubles, while the AVR IAR Compiler supports both 4-byte and 8-byte doubles. Use the compiler option `--64bit_doubles` to control the size of the double type.

   If you do not use 8-byte doubles, it is not necessary not modify doubles in the source code.

4. Replace the AT90S IAR Compiler extended keywords in the source code with AVR IAR Compiler keywords.

   In the AVR IAR Compiler, all extended keywords except asm start with two underscores, for example `__near`. This is managed by the migration macros in the file `comp_a90.h`.

   Notice that the behavior of the following extended keywords has changed: memory specification keywords (for example, `__near` and `__flash`), `__no_init`, `__interrupt`, and `__monitor`.

   In the AVR IAR Compiler, the `sfrb` and `sfrw` keywords are not available. Replace `sfrb` with `volatile __io unsigned char` and replace `sfrw` with `volatile __io unsigned int`.

   Refer to the *AVR IAR Compiler Reference Guide*for details.

5. Replace the AT90S IAR Compiler `#pragma` directives with AVR IAR Compiler directives. Notice that the behavior differs between the two products. For example:

   - In the AT90S IAR Compiler, the directives `#pragma memory` and `#pragma function` change the default attribute to use for declared objects; they do not have any effect on pointer types.
   - In the AVR IAR Compiler the directives `#pragma type_attribute` and `#pragma object_attribute` change the next declared object or `typedef`.

   The set of `#pragma` directives in the AVR IAR Compiler is quite different from that in the AT90S IAR Compiler. Refer to the *AVR IAR Compiler Reference Guide* for details.

6. Replace the AT90S IAR Compiler intrinsic functions with AVR IAR Compiler intrinsic functions.

   In the AVR IAR Compiler, the intrinsic functions start with two underscores, for example `__enable_interrupt`. This is handled by the migration macros in the file `comp_a90.h`.

   The AT90S IAR Compiler intrinsic functions `_args$` and `_argt$` are not available in the AVR IAR Compiler. Other AT90S IAR Compiler intrinsic functions are available, but have new names in the AVR IAR Compiler. The translation of names is handled by the migration macros in the file `comp_a90.h`.

   New intrinsic functions have been added in the AVR IAR Compiler. Refer to the *AVR IAR Compiler Reference Guide* for details.

7. Rewrite interrupt functions from:

   ```
   interrupt void [vector] func()
   ```

   to

   ```
   #pragma vector=vector
   __interrupt void func()
   ```

8. Modify the predefined symbols.

   In the AVR IAR Compiler, all predefined symbols start and end with double underscores, for example `__IAR_SYSTEMS_ICC__`.

   New predefined symbols have been added in the AVR IAR Compiler, for example, __ICCAVR__ which allows you to distinguish between the compilers. See the *AVR IAR Compiler Reference Guide* for details.

9. Compile the code using appropriate AVR IAR Compiler compiler options; these are described in the chapter *Compiler options* in the *AVR IAR Compiler Reference Guide.* If you use the IAR Embedded Workbench, refer to the *AVR IAR Embedded Workbench User Guide*.

10. Link the code by using one of the ready-made linker command files provided with the product. Alternatively, use the appropriate linker command file template and modify in accordance with the requirements of your application. This is described in the *Configuration* chapter in the *AVR IAR Compiler Reference Guide*.

11. Run the project in the IAR C-SPY Debugger in order to sort out any remaining problems.

Note: When your project compiles without problem, you should substitute the intrinsic functions properly in the source code. The `comp_a90.h` migration file is intended as an interim solution.

The User Benefits

By following the migration process described above, you will be able to compile your existing application with the brand new AVR IAR Compiler compiler with minimal effort.

Conclusion

In spite of the obvious differences between the AT90S IAR Compiler and the AVR IAR Compiler, existing AT90S IAR Compiler applications can easily be adapted to benefit from the advantages of the new AVR IAR Compiler.