

Simplifying Microcontroller Development through Compiler Flexibility



Developing code that is compiler independent for microcontrollers adds a great degree of flexibility to any project and helps companies mitigate the risks associated with compiler dependence.

Compiler-independent Projects

By creating a compiler-independent project, the developer is given much more flexibility to shift from one tool chain to another with ease. Having the flexibility to try different compilers with a project can also clearly show the performance of various tool chains operating under the same conditions. Compilers each have their own individual characteristics that distinguish one from another. One compiler may optimize the assembly that is generated to be very small, another may optimize for speed, and another may not optimize at all. Optimizations for size can be an important consideration if the application code size is the final deciding factor in the microcontroller selection process. If the code size is not a factor, the cost of a license may be an issue. Today, 8051 compilers range in cost from several thousand dollars per license to nothing at all.

Decreased Debugging Time

Aside from the added flexibility and ease of switching between compilers, compiler-independent code is much more readable than compiler-specific code. This readability leads to decreased debugging time, which in turn reduces the overall cost of a project. In an application where various engineers will be using the same source code in their end project, there is a need for the source code to work with more than one tool chain. In this case, the developer would specify certain tools that have been successfully tested with the project and include instructions on how to integrate different tool chains into the build. This flexibility increases the ability for code reuse among engineers.

Using macro definitions in source code instead of compiler-specific syntax is the basis for making a project compiler independent. The only extra file needed in the project is a header file with the macro definitions for supported tool chains. This file contains a list of conditional “if” statements where the argument to the “if” statement is a compiler name that, if selected, will cause the macro definitions for a specific compiler to be used throughout the project. As a result, only one line of code needs to be changed in the macro definition file in order to modify many lines in the source code.

The idea of using a macro definition header file is no different than using a microcontroller-specific header file for special function registers names, addresses, and bits. The macro definition header file is an included header file that is designed to make the code development process easier.

Compilers each have their own specific syntax for the following:

- Interrupts
- Interrupt prototypes
- Register banking
- Memory segment definitions
- Locating variables in memory segments
- Pointers to memory segments
- No-operations (NOPs)
- Special function register declaration
- Special function register bit declaration

Any of these can prevent a project from becoming compiler independent if compiler-specific syntax is used. Other potential issues when switching between tool chains include endian differences, generic pointers, and locating and initializing variables. Big endian compilers store the most significant byte of a multibyte number at the lowest memory location and little endian compilers store the least significant byte of a multibyte number at the lowest memory location. This key difference is important to note if there is any sort of manipulation of a multibyte number in the source code.

Example of Big Endian vs. Little Endian Variable Storage

For a 4-byte variable called 'temp-variable' stored at address 'x':

temp-variable = 0 x 01 02 03 04

Big Endian		Little Endian	
Memory Location	Contents	Memory Location	Contents
X	01	X	04
X + 1	02	X + 1	03
X + 2	03	X + 2	02
X + 3	04	X + 3	01

The format of generic pointers can also vary. In a three-byte generic pointer, some compilers may use the least significant byte to store the target memory segment and the remaining two bytes for the address, and some will use the most significant byte to store the target memory segment and the remaining two bytes for the address. Initializing a variable that is located at a specified memory address can potentially cause problems as well because not all compilers allow a located variable to be initialized at the same time.

Real Life Examples of Compiler Independence

An example of a compiler-independent macro definition header file and microcontroller example code is available for download from the Silicon Laboratories website:

<https://www.silabs.com/products/mcu/Pages/SoftwareDownloads.aspx>

The example code and compiler-independent macro definition header file are both installed with the Silicon Laboratories Integrated Development Environment (IDE).

Compiler_defs.h contains macro definitions for the following 8051 tool chains:

- SDCC
- Raisonance
- Keil
- Tasking
- IAR

The following items also have macros defined in the `compiler_defs.h` file:

- memory segments
- interrupts
- interrupt prototypes
- register banking
- locating variables
- memory-specific pointers

This header file can be used on any Silicon Laboratories microcontroller and is also used by device-specific header files that define special function registers, interrupt numbers, and bit addressable special function registers.

The macro definitions in `compiler_defs.h` can be used as a starting point to integrate any tool chain into a project.

Summary

Making a compiler-independent project gives the developer a significant amount of flexibility. The risk associated with compiler dependence is diminished and the performance of compilers can be examined and compared side-by-side. Changes to one line of code in a header file will propagate throughout the project, which can decrease debugging time. The overall effort to port an existing project or to create a compiler-independent project is minimal. The benefits and ease of generating compiler-independent code for microcontrollers makes this approach something that any developer should consider.

For more information, visit www.silabs.com/mcu.