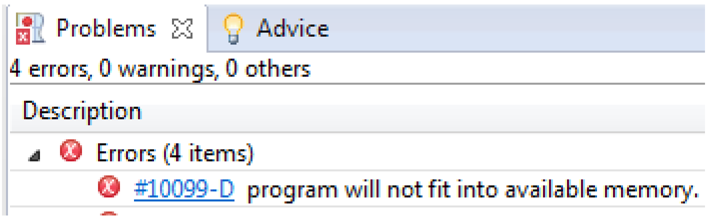
Technical Article

Optimizing Code to Fit Your MCU: Tips and Tricks

Katie Pier

Have you ever been close to finishing your microcontroller (MCU) code, only to get the dreaded error message shown in Figure 1?



4 errors, 0 warnings, 0 others

Description

Errors (4 items)

#10099-D program will not fit into available memory.

Figure 1. Error Message

Your program is too large to fit in your device! This may seem like a hopeless situation, but fortunately, there are a lot of strategies you can take to get your code to build smaller and hopefully squeeze a few more functions into your device. By using the compiler settings to their fullest and carefully considering how you write your code, you can get the best code-size efficiency possible for your MSP430™ MCU project.

Tip No. 1: Use Optimizer Assistant in Code Composer Studio™ IDE

The Code Composer Studio integrated development environment (IDE) offers different optimization levels as well as different size-versus-speed settings for compiler optimizations. That's a lot of combinations to try. However, there is a little-known gem built right into the software called Optimizer Assistant; see Figure 2.

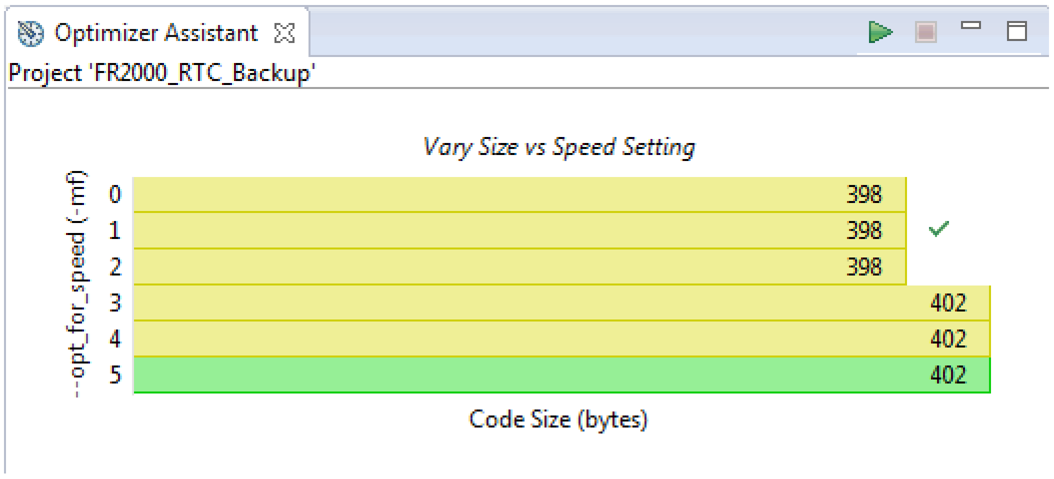
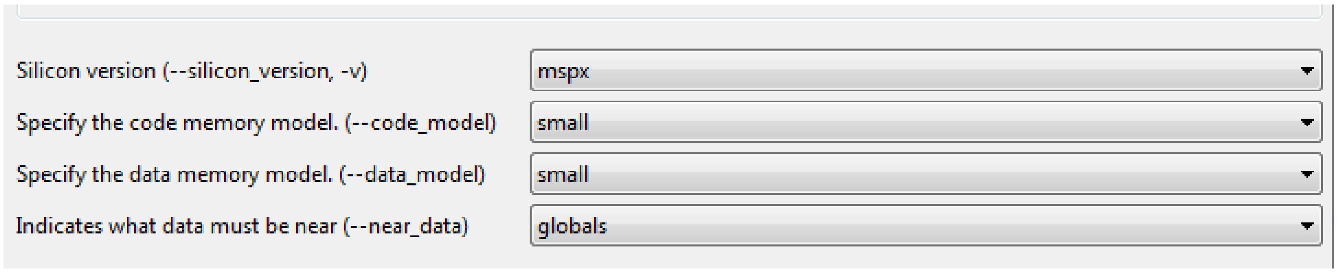


Figure 2. Optimizer Assistant

Optimizer Assistant automatically builds with all of the different variants for size versus speed or optimization level and reports the code size back for each, along with whether the project will fit into the device. You can use it to help pick the right settings for your project: for example, to get the best speed performance for code that will still fit into the device's nonvolatile memory.

Tip No. 2: Pick the Right Code and Data-model Settings

The MSP430X central processing unit (CPU) architecture on most MSP430 MCUs supports up to 20-bit addresses, because some MSP430 devices have memory sizes such that they need to have locations above 10000 h, requiring 20-bit addressing. However, the extended instruction set used for operations on 20-bit addresses requires increased program space. The code and data-model settings as shown in Figure 3 tell the compiler whether to support 20-bit addressing and extended instructions or not.



Silicon version (--silicon_version, -v)	mspx
Specify the code memory model. (--code_model)	small
Specify the data memory model. (--data_model)	small
Indicates what data must be near (--near_data)	globals

Figure 3. Code and Data Model Settings

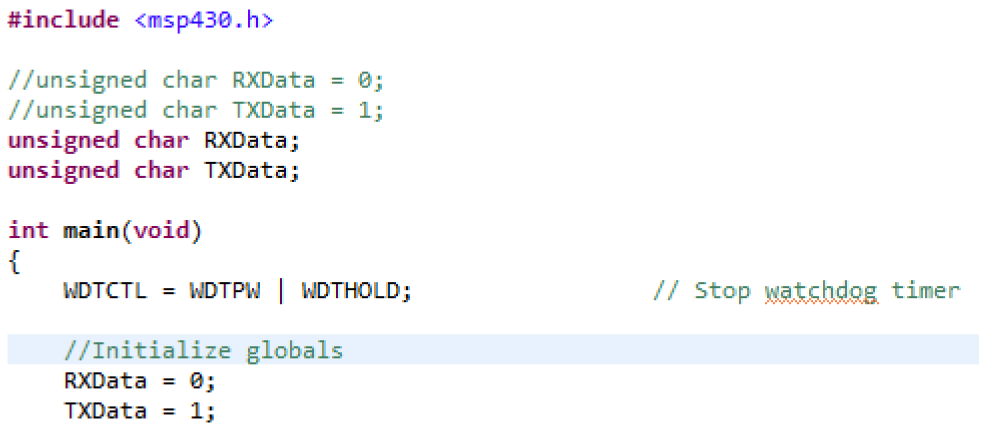
So if you are using an MSP430 device that doesn’t have any nonvolatile memory above 10000h, make sure to choose small code model so that the compiler can optimize and use only 16-bit addressing, thus taking less code space.

Selecting small data model tells the compiler that there will be no variables above 10000h (for example, no random access memory [RAM] nor ferroelectric RAM [FRAM] variables above 10000h). Making just this simple setting change on the msp430fr211x_euscia0_uart_03.c code example caused the code to build 38% smaller.

Tip No. 3: Eliminate Automatic Global Variable Initialization

The C compiler inserts initialization code from the runtime support (RTS) library that runs every time the device starts up to prepare the C environment. The C compiler’s default initialization routines use a compressed table of initialization data for global variables and therefore run decompression routines during startup to get the data and initialize the variables. For large projects with large numbers of variables, this saves code space. But for small projects with just a few global variables, the code is no longer space-efficient.

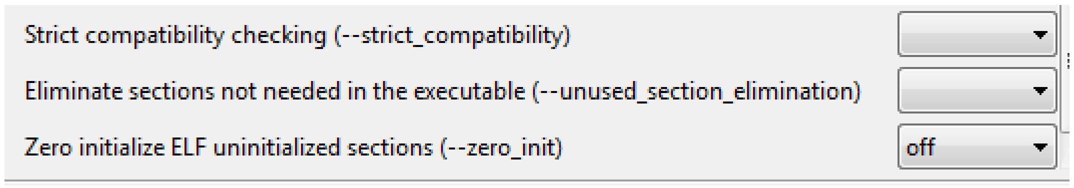
When you are using a small device and are concerned about code size, I recommend either eliminating global variables entirely (if possible), or using a limited number of global variables and setting them up such that they are initialized at the beginning of main() by user code instead of by the RTS library. To do this, simply move the initial value of the global variable into main, as shown in Figure 4.



```
#include <msp430.h>

//unsigned char RXData = 0;
//unsigned char TXData = 1;
unsigned char RXData;
unsigned char TXData;

int main(void)
{
    WDTCTL = WDTPW | WDTHOLD;                 // Stop watchdog timer

    //Initialize globals
    RXData = 0;
    TXData = 1;
```

Figure 4. Moving Initialization into Main()

Then, set Project > Properties > MSP430 Linker > Advanced Options > Miscellaneous > “Zero initialize ELF uninitialized sections (--zero_init)” to off, as shown in Figure 5.



Strict compatibility checking (--strict_compatibility)

Eliminate sections not needed in the executable (--unused_section_elimination)

Zero initialize ELF uninitialized sections (--zero_init) off

Figure 5. Disable Automatic Zero-initialization of Uninitialized Variables

For the msp430fr211x_euscia0_uart_03.c example, making this change resulted in a 62% reduction in code size, saving more than half the memory size of the 512-byte MSP430FR2000 device.

Tip No. 4: Write Your Code for Efficiency

There are also a number of coding techniques, like using lookup tables instead of complex calculations, using small types for variables and constants, using the __even_in_range() intrinsic, and more that can help your code build in the smallest amount of space.

The application note, "Optimizing C Code for Size with MSP430 MCUs: Tips and Tricks," outlines these and other techniques in more detail, including tips for the IAR compiler. Check it out today and get optimizing!