CCS/IAR Code Size Optimization for Small Devices

MSP430 Apps

Dec 2017

Agenda

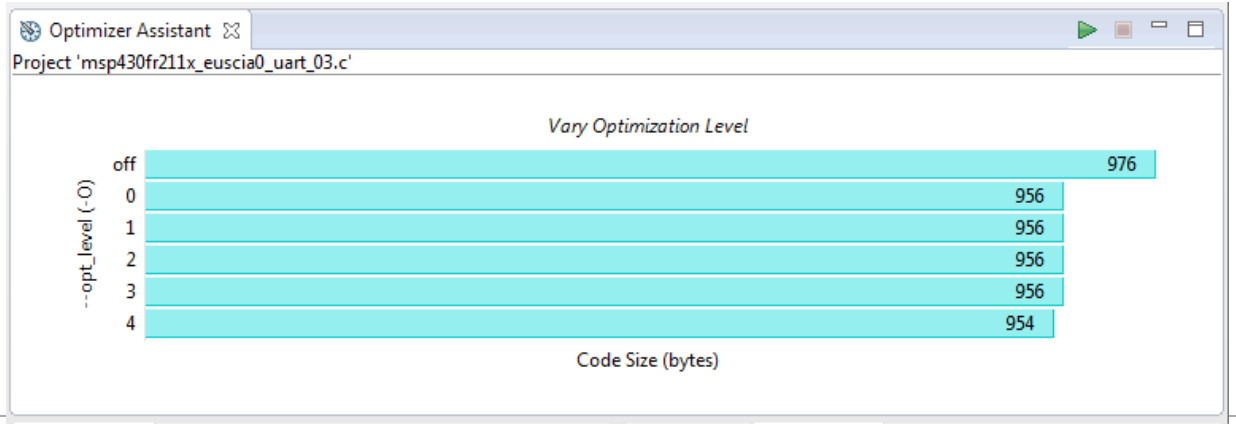
- Optimization Assistant
- Code & Data Model
- Global Variables & Initialization Settings
  - Eliminate Global variables
  - Move initialization into main

Purpose

- All topics will be represented in terms of code savings on an example project:
  - Msp430fr211x_euscia0_uart_03.c
- CCSv7.2
- http://dev.ti.com/tirex/#/?link=Software%2FMSP430Ware%2FDevices%2FMSP430FR2XX_4XX%2FMSP430FR2111%2FPeripheral%20Examples%2FRegister%20Level%2Fmsp430fr211x_euscia0_uart_03.c
- CCS default settings are not fully optimized for code size. When using a very small part like FR2000 with only 0.5kB of FRAM, every byte counts. The next topics will give clues on how to better optimize for size.
  - The .map file found under the Debug folder will be used to reference size of code

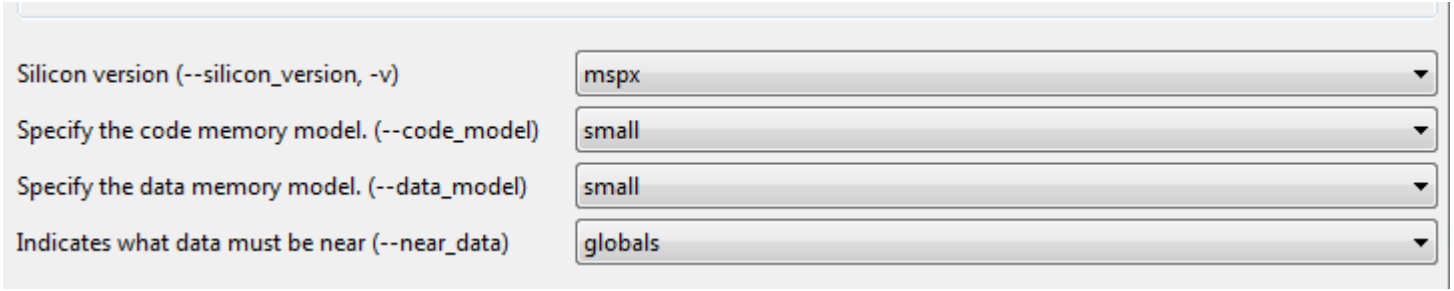
Optimization Assistant

- In CCS, there is a tool called Optimization Assistant that can be used to find the optimization level needed for code to fit into the device.
  - View > Optimizer Assistant
- Click Start Analysis
  - Can vary Speed vs Size or can vary Optimization level
- Running the tool allows you to determine the optimization settings you want to choose



Code & Data Model

- CCSv7.2 appears to default to Large code model even on small devices like FR2000
  - This is inefficient because it will use the larger instructions for 20-bit addresses, even though there are no 20-bit addresses in the part
- Set both Code & Data Model to small memory model for best optimization (note this is only for parts without FRAM or RAM above 0x10000 boundary)



- CCS default settings: 928 bytes
- Small Code & Data Model: 572 bytes

Global Variables & Initialization Settings

- With global variables in the project (in this case 2 variables, RXData and TXData), there are initialization routines included by the compiler that take up space.
  - Look in the .map file, you'll see a few sections of .text that aren't part of application code and may not be needed for your simple application:

```
.text      0    0000f10e    00000226
                0000f10e    0000007e     rts430x_sc_sd_eabi.lib : autoinit_wdt.obj (.text:_auto_init_hold_wdt)
                0000f18c    0000006e                            : copy_decompress_lzss.obj (.text:decompress:lzss:__TI_decompress_lzss)
                0000f1fa    00000058                            : cpy_tbl.obj (.text:copy_in)
                0000f252    00000054     msp430fr211x_euscia0_uart_03.obj (.text:main)
                0000f2a6    00000032     msp430fr211x_euscia0_uart_03.obj (.text:USCI_A0_ISR)
                0000f2d8    0000001a     rts430x_sc_sd_eabi.lib : boot_special.obj (.text:_c_int00_noargs_noexit)
                0000f2f2    00000014                            : mult16.obj (.text)
                0000f306    00000012                            : copy_decompress_none.obj (.text:decompress:none:__TI_decompress_none)
                0000f318    00000010                            : memcpy.obj (.text:memcpy)
                0000f328    00000006                            : exit.obj (.text:abort)
                0000f32e    00000004                            : pre_init.obj (.text:_system_pre_init)
                0000f332    00000002                            : startup.obj (.text:_system_post_cinit)
```

  - These are for initializing global variables at startup.
  - The next slides will show how to eliminate these sections.

Global Variables Options

- Option 1: Don’t use global variables
  - If possible, don’t use global variables. That will eliminate most of this overhead.
- Option 2: Use a very small number of global variables, and don’t have the pre-initialized. Initialize them inside of main.
  - Next slides focus on how to implement option 2

Move initialization into main

- Our example includes two necessary global variables: RXData and TXData
- The c-startup code to initialize these is inefficient for extremely small numbers of variables when you are code-size sensitive
  - Compression routine gets included. Makes sense on large parts, but not when there are so few variables and so little code space
- **Fix: Move initialization of the globals into main**

```
#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;
```

Before: 572 bytes
After: 468 bytes

Disable WDT hold during pre-init

- **If** for some reason you need your global variables to be pre-initialized (instead of init at beginning of main) you can at least get rid of the routine that halts the watchdog during initialization.
  - In a project this small, you won't have enough variables for the watchdog to time out before main and this to be a concern.

```
.text      0    0000f10a    000001c2
                0000f10a    0000007e     rts430x_sc_sd_eabi.lib : autoinit_wdt.obj (.text:_auto_init_hold_wdt)
                0000f188    0000005c     msp430fr211x_euscia0_uart_03.obj (.text:main)
                0000f1e4    00000058     rts430x_sc_sd_eabi.lib : cpy_tbl.obj (.text:copy_in)
                0000f23c    00000032     msp430fr211x_euscia0_uart_03.obj (.text:USCI_A0_ISR)
                0000f26e    0000001a     rts430x_sc_sd_eabi.lib : boot_special.obj (.text:_c_int00_noargs_noexit)
                0000f288    00000014                            : copy_zero_init.obj (.text:decompress:ZI:__TI_zero_init)
```

- **FIX:** Project > Properties > MSP430 Linker > Basic Options, set "Hold watchdog timer during cinit auto-initialization" to "off"

Before: 468 bytes
After: 448 bytes

Disable zero-initialization

- Even after moving initialization of global variables into main, there is some wasted code space for the global variables
  - By default, CCS will initialize any un-initialized Global variables to 0

```
.text      0    0000f10a    000001c2
                 0000f10a      0000007e     rts430x_sc_sd_eabi.lib : autoinit_wdt.obj (.text:_auto_init_hold_wdt)
                 0000f188      0000005c     msp430fr211x_eusci_a0_uart_03.obj (.text:main)
                 0000f1e4      00000058     rts430x_sc_sd_eabi.lib : cpy_tbl.obj (.text:copy_in)
                 0000f23c      00000032     msp430fr211x_eusci_a0_uart_03.obj (.text:USCI_A0_ISR)
                 0000f26e      0000001a     rts430x_sc_sd_eabi.lib : boot_special.obj (.text:_c_int00_noargs_noexit)
                 0000f288      00000014                            : copy_zero_init.obj (.text:decompress:ZI:__TI_zero_init)
```

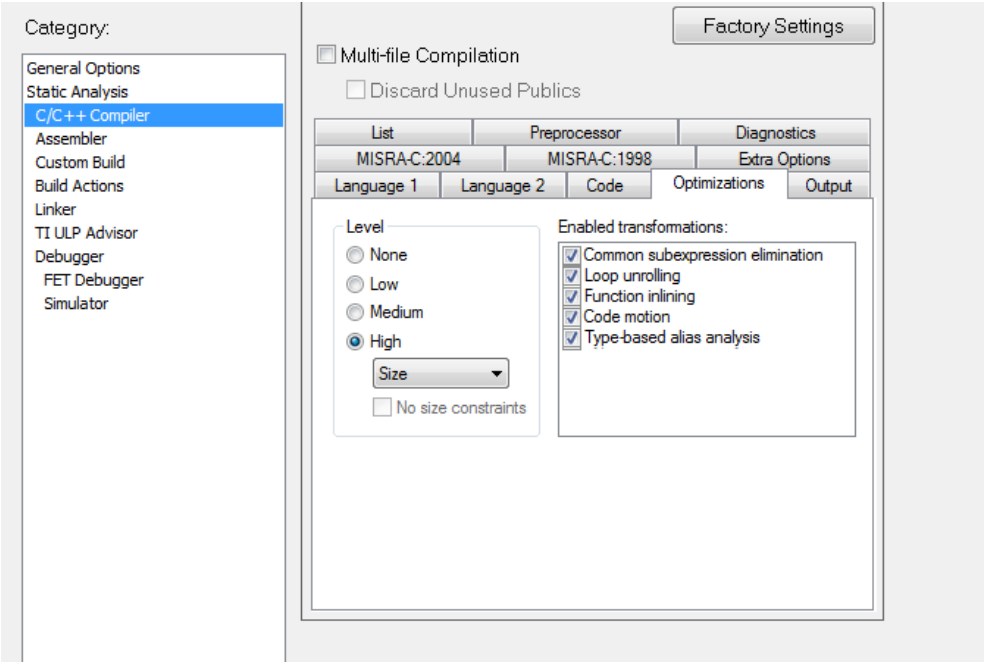
- **FIX:** Either set ALL globals as no-init using #pragma NOINIT, or use the CCS setting to disable zero-initialization (Project > Properties > MSP430 Linker > Advanced Options > Miscellaneous, set “Zero initialize ELF uninitialized sections” to “Off”)

Before: 468 bytes
After: 178 bytes

Optimization Settings - IAR

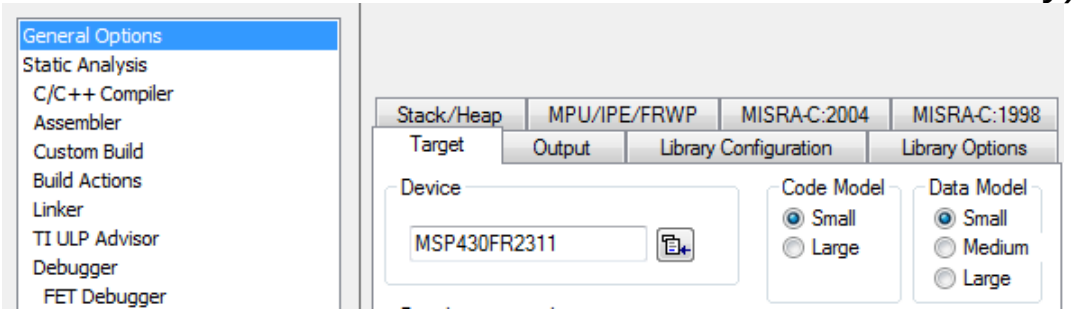
- Set Optimization level to highest level to get more optimized code size
- Project > Options > C/C++ Compiler > Optimizations tab
  - Select Level to High
  - Set dropdown to Size

Before: 250 bytes
After: 238 bytes



Code & Data Model - IAR

- IAR EW430 v7.10.4 appears to default to Large code model even on small devices like FR2000
  - This can be inefficient because it may use the larger instructions for 20-bit addresses, even though there are no 20-bit addresses in the part
- Set both Code & Data Model to small memory model for best optimization (note this is only for parts without FRAM or RAM above 0x10000 boundary)



- IAR default settings: 238 bytes
- Small Code & Data Model: 238 bytes

Global Variables & Initialization Settings - IAR

- With global variables in the project (in this case 2 variables, RXData and TXData), there are initialization routines included by the compiler that take up space.
  - Look in the .map file, you'll see a few sections of .text that aren't part of application code and may not be needed for your simple application:
  - These are for initializing global variables at startup.
  - The next slides will show how to eliminate these sections.

Module	CODE	DATA		CONST
	(Rel)	(Rel)	(Abs)	(Rel)
?__dbg_break	2			
?__exit	20			
?_exit	4			
?cstart	40			
?exit	4			
?memcpy	18			
?memzero	20			
?reset_vector	2			
msp430fr211x_euscia0_uart_03	130	2	26	1
+ common	102			
N/A (command line)		160		
Total:	240	162	26	1
+ common	102			

Global Variables Options - IAR

- Option 1: Don’t use global variables
  - If possible, don’t use global variables. That will eliminate most of this overhead.
- Option 2: Use a very small number of global variables, and don’t have the pre-initialized. Initialize them inside of main.
  - Next slides focus on how to implement option 2

Move initialization into main - IAR

- Our example includes two necessary global variables: RXData and TXData
- The c-startup code to initialize these is inefficient for extremely small numbers of variables when you are code-size sensitive
  - Compression routine gets included. Makes sense on large parts, but not when there are so few variables and so little code space
- **Fix: Move initialization of the globals into main**

```
#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;
```

Before: 238 bytes
After: 208 bytes

Disable zero-initialization - IAR

- Even after moving initialization of global variables into main, there is some wasted code space for the global variables
  - By default, CCS will initialize any un-initialized Global variables to 0
- **FIX:** Precede all global variables with the __no_init keyword.
  - E.g. __no_init unsigned char RXData;

Module	CODE	DATA	
	(Rel)	(Rel)	(Abs)
?__dbg_break	2		
?__exit	20		
?_exit	4		
?cstart	24		
?exit	4		
?memzero	16		
?reset_vector	2		
msp430fr211x_euscia0_uart_03	138	2	26
+ common	102		
N/A (command line)		160	
---	---	---	---
Total:	210	162	26
+ common	102		

Before: 208 bytes
After: 180 bytes