MSP430 Compiler Tips & Tricks

George Mock

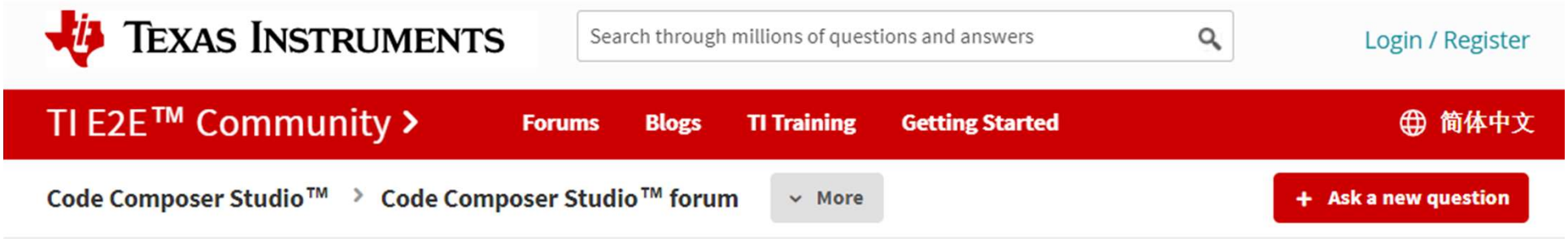
August 2019

Agenda

- Resources
- Compiler Options
- MSP430 Specifics
- Diagnostics
- Other Tips

Online Resource - CCS Forum

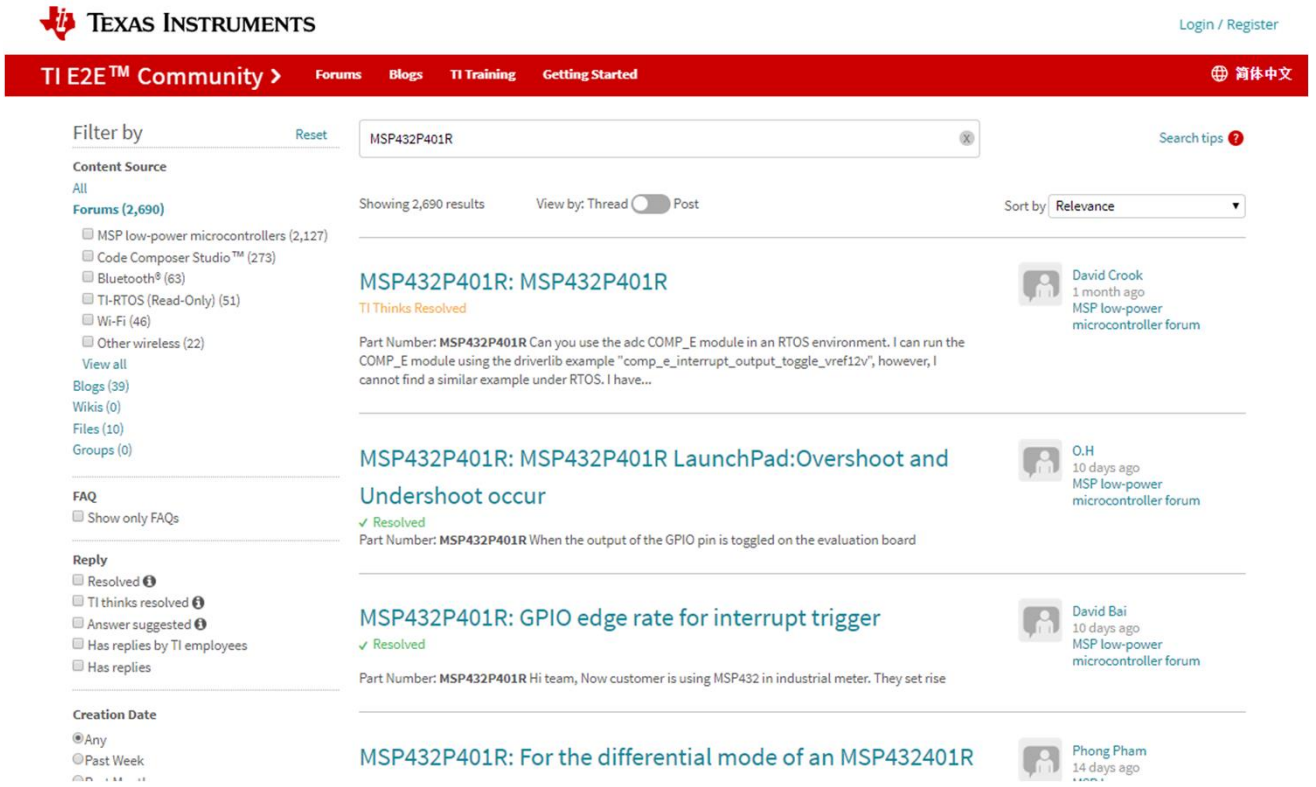
- http://e2e.ti.com/support/tools/ccs/f/81(link)
- Questions about CCS and Compiler
- Many responses are the same business day
- Thousands of posts to search



Online Resource – E2E China

- https://e2echina.ti.com (link)

Forum Filter Search Results



CCS Documentation Online

- http://software-dl.ti.com/ccs/esd/documents/ccs_documentation-overview.html (link)

Feature Overviews, Application Notes and How-to Articles

General and IDE

The resources below relate to the Code Composer Studio Eclipse IDE and other general topics.

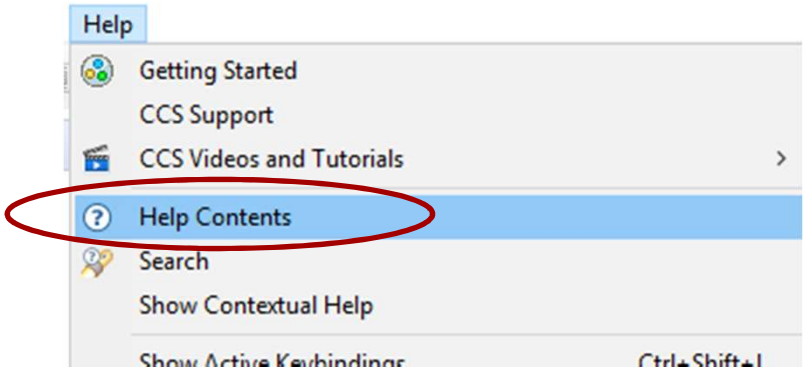
Eclipse Concepts	Information on various concepts that are part of the Eclipse/CCS environment
Getting Started View	The Getting Started View in CCS
Resource Explorer	Resource Explorer helps you find all the latest examples, libraries, demo applications, datasheets, and more for your chosen platform
App Center	The Code Composer Studio App Center provides access to additional tools and utilities to help users get up and running faster on their chosen platform
Tasks View	Tasks view in CCS allows you to create and keep track of 'To-Do' (or Tasks) list
MatLab with CCS	This document describes the level of CCS support for various MatLab releases

TI Compiler Online Home

- http://www.ti.com/tool/ti-cgt (link)
- All TI compilers
- Downloads
- Up-to-date manuals

Compiler Manuals in CCS

- Compiler Manuals available from within CCS



Compiler Manuals in CCS

Use the README!

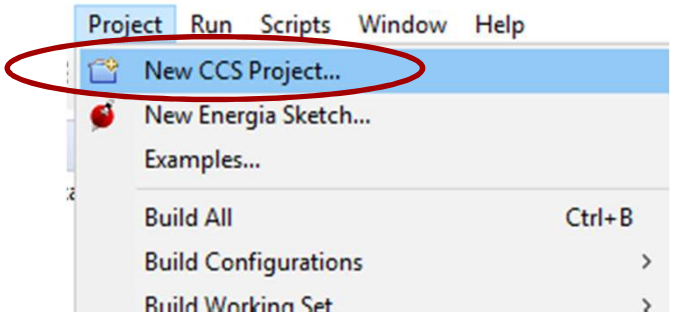
- Latest information on compiler
- Critical details
- Well worth the 30-60 minutes it takes to read it
- In root directory of compiler install
- Typical path
  - C:\ti\ccs901\ccs\tools\compiler\ti-cgt-msp430_18.12.1.LTS\README.txt

Agenda

- Resources
- **Compiler Options**
- MSP430 Specifics
- Diagnostics
- Other Tips

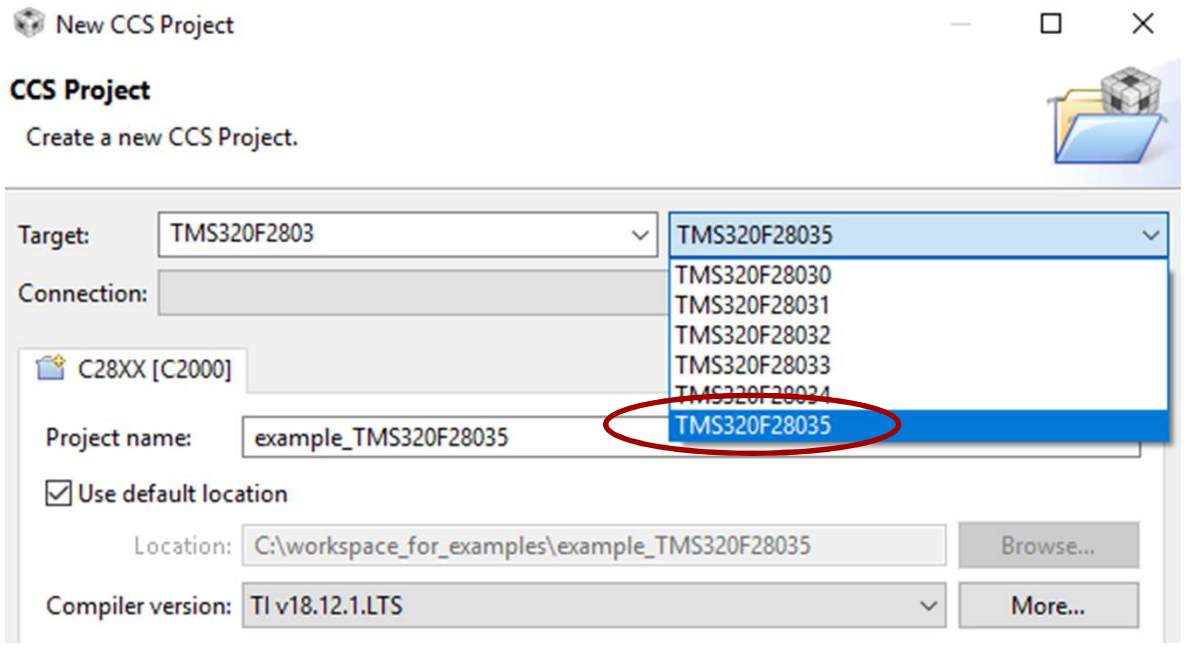
Compiler Options Chosen by CCS

- When you start a new project in CCS



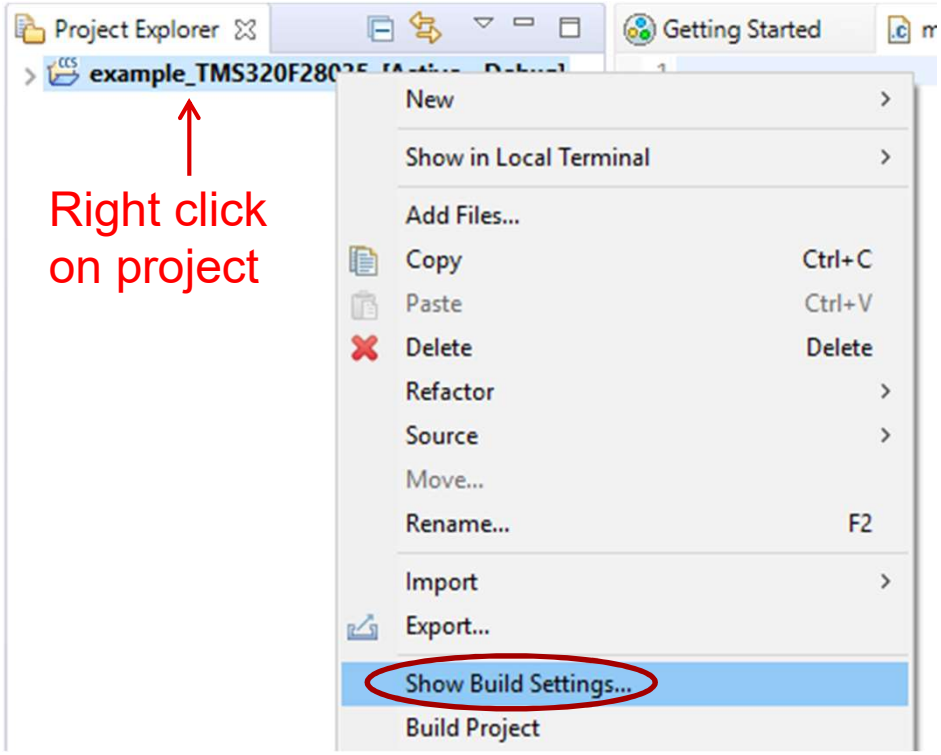
Compiler Options Chosen by CCS

- You specify the processor



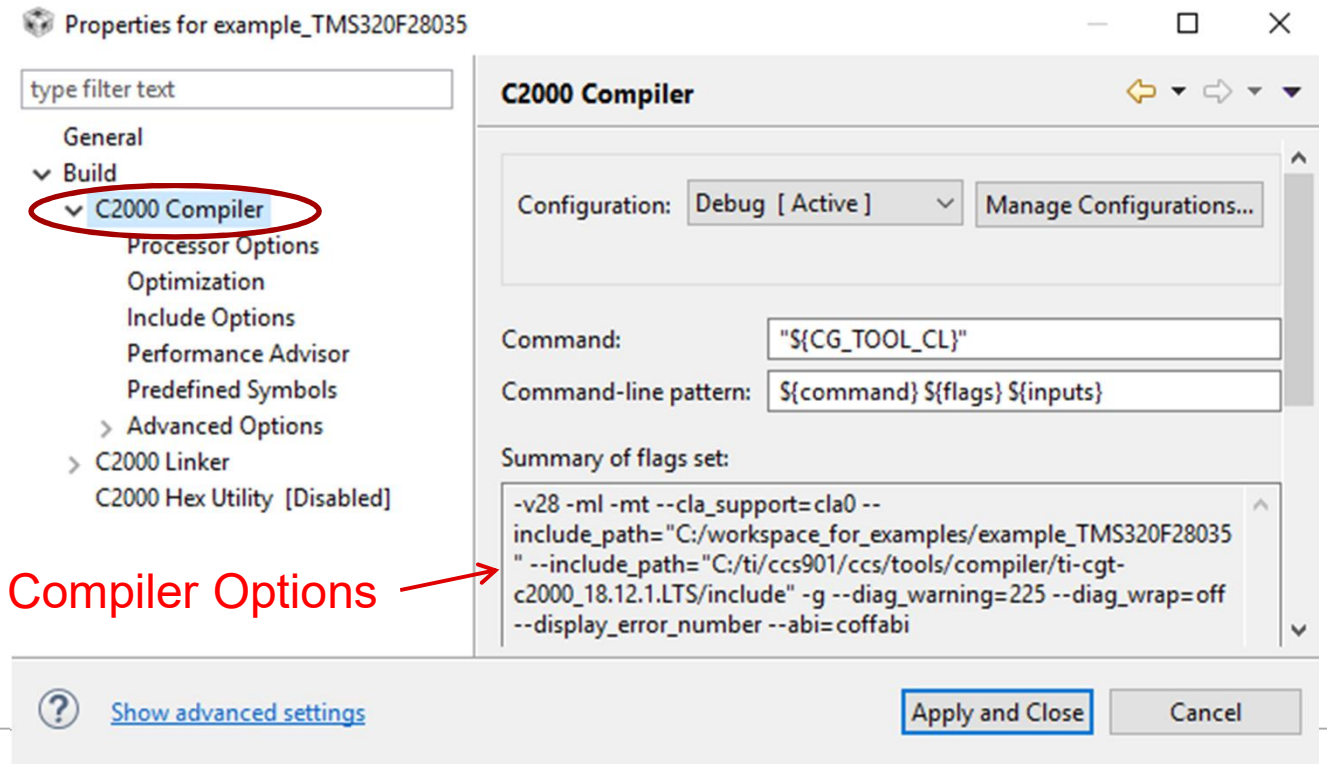
Compiler Options Chosen by CCS

- CCS then chooses many compiler options for you
- Here is how to see them



Compiler Options Chosen by CCS

- These slides explain many of these automatically chosen options



MSP430 Specific Compiler Options

- --silicon_version=*value*
  - Valid values: msp, mspx
  - msp: 16-bit code addressing
  - mspx: 20-bit code addressing
- --use_hw_mpy=*value*
  - Valid values: 16, 32, F5, none
  - Controls which RTS function performs multiply
  - If --opt_for_speed >= 4 and --opt_level >= 0, multiply function is inlined
- --align_for_power
  - Aligns functions and loops to 4-byte boundaries
  - Avoids cache misses
  - Best for small loops and interrupt service routines
  - Increases code size

MSP430 Code Memory Model

--code_model=	Function Ptrs	Range	Penalty
small	16 bits	0-64K	none
large	20 bits	0-1MB	slight

- All files and libraries must use the same model
- Enforced by the linker

MSP430 Data Memory Model

--data_model=	Range	Object Size	Penalty
small	0-64K	64K	none
restricted	0-1MB	64K	slight
large	0-1MB	1MB	modest

- All files and libraries must use the same model
- Enforced by the linker

MSP430 Near Data

- Regardless of --data_model, compiler presumes memory past 64K is read only
- Disable with --near_data=none
  - Small penalty
  - Default is --near_data=global
- Can mix files and libraries with different --near_data settings

Optimization

Option	Range of Optimization
--opt_level=off	None
--opt_level=0	Statements
--opt_level=1	Blocks
--opt_level=2	Functions
--opt_level=3	Files
--opt_level=4	Between files and libraries

- Only a rough summary
- Some level 0 and 1 optimizations range farther

Default Optimization Level - MSP430

- The default level is always --opt_level=0
- Occasionally makes debugging uncomfortable
  - Example: When single stepping, a line appears to be skipped over for no reason
- If that happens, try --opt_level=off

Link Time Optimization --opt_level=4

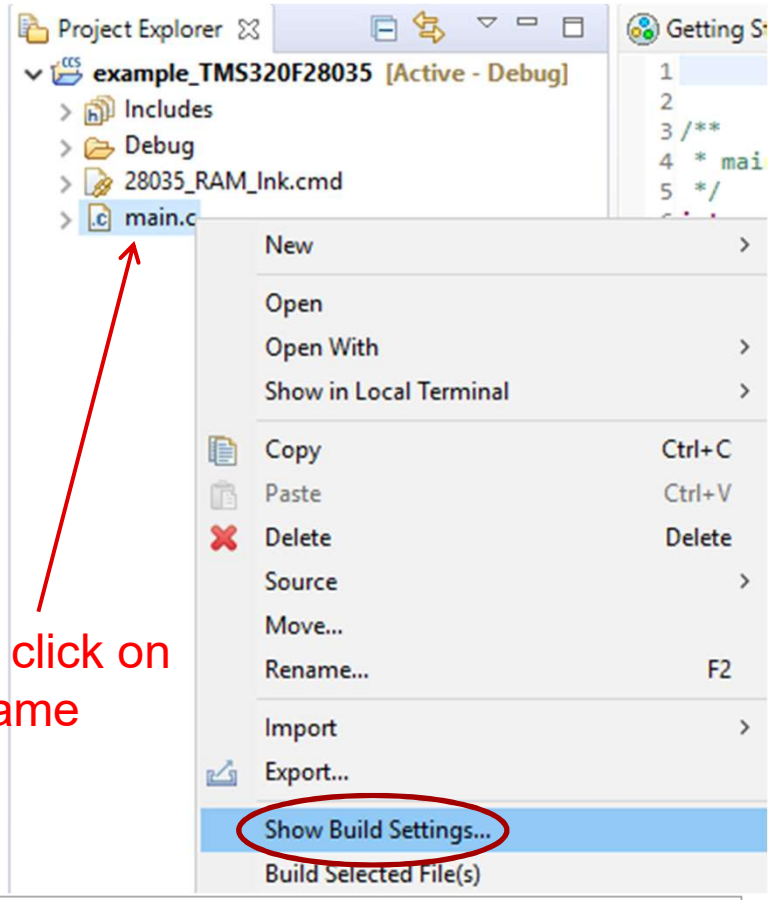
- Optimizes across the entire program
- Linking takes longer
- Presents opportunities rarely seen within files
  - May see all the calls to a function
  - If one argument is always the same, just replace it
- Use --opt_level=4 during compile and link
  - CCS takes care of this for you
- Information encoded in object files during compile step is used by optimization during link step
- Libraries built with --opt_level=4 can participate

Debug vs Optimization Trade-Off

- Compiler emits debug information used by CCS
  - Where are the variables
  - What line of source is executing
- Do not need -g
  - Debug information always emitted
- Optimization still affects ease of debugging
- What is the trade-off point? The lowest --opt_level which meets your system constraints
- http://software-dl.ti.com/ccs/esd/documents/sdto_cgt_debug_versus_optimization_tradeoff.html (link)

File Specific Options

- Use Case: The --opt_level you need is too hard to debug
- Possible Solution: Reduce --opt_level only for the files you are debugging
- http://software-dl.ti.com/ccs/esd/documents/users_guide/ccs_project-management.html#file-specific-options (link)



Agenda

- Resources
- Compiler Options
- **MSP430 Specifics**
- Diagnostics
- Other Tips

MSP430 Watchdog Timer

- Resets the device every N cycles
- Enabled by default
- Avoids a runaway system
- Your code must either manage it, or stop it
  - Timer is stopped by setting the hold bit
- For all the HW details see http://www.ti.com/lit/pdf/slau399 (link)

MSP430 Watchdog Timer

- Typical first attempt …

```
#include <msp430.h>
int main()
{
    WDTCTL = WDTPW | WDTHOLD;      // Stop watchdog timer
    …
}
```

- Does this always work? No.
- Initialization of C global variables occurs before main, and may take too long

MSP430 Watchdog Timer

- Linker option --cinit_hold_wdt=*state*
- Valid states: on, off
- During C global variable initialization
  --cinit_hold_wdt=on means watchdog is stopped
  --cinit_hold_wdt=off means watchdog is unmodified

MSP430 Useful Documents

- MSP430 Software Coding Techniques
  - Explains why MSP430 is programmed with an interrupt-based code flow model
  - http://www.ti.com/lit/pdf/slaa294 (link)
- CCS User's Guide for MSP430
  - Perfect for those new to CCS and MSP430
  - http://www.ti.com/lit/pdf/slau157 (link)

Agenda

- Resources
- Compiler Options
- MSP430 Specifics
- **Diagnostics**
- Other Tips

Compiler Diagnostics

	Remark	Warning	Error
Severity	Low	Medium	High
Build fails?	No	No	Yes
To enable	--issue_remarks	Default	Default

- Advice: Do not ignore remarks
- Indicates a real problem most of the time

Control Diagnostic Levels

- First see diag *id* with --display_error_number

Set level to:	Option	#pragma
Remark	--diag_remark=id	#pragma diag_remark id
Warning	--diag_warning=id	#pragma diag_warning id
Error	--diag_error=id	#pragma diag_error id
Default	none	#pragma diag_default id
Suppress	--diag_suppress=id	#pragma diag_suppress id

- Diagnostics with “-D” appended to id can be suppressed or changed
  - All warnings or remarks
  - A few errors
- #pragma provides line by line control

Diagnostic Control Example

```
C:\dir>type ex.c
int contrived_example(int i)
{
    switch (i)
    {
        case 10 :
           return val();                 /* line 6                  */

           break;                        /* line 8                  */

    }
    return 0;
}

C:\dir>cl430 --display_error_number ex.c
"ex.c", line 6: warning #225-D: function "val" declared implicitly
"ex.c", line 8: warning #112-D: statement is unreachable
```

Diagnostic Control Example

```
C:\dir>type ex.c
int contrived_example(int i)
{
    switch (i)
    {
        case 10 :
           return val();                  /* line 6                    */
           #pragma diag_suppress 112      /* suppress diag on break */
           break;                         /* line 8                    */
           #pragma diag_default 112       /* restore diag level     */
    }
    return 0;
}

C:\dir>cl430 --display_error_number --diag_error=225 ex.c
"ex.c", line 6: error #225-D: function "val" declared implicitly
1 error detected in the compilation of "ex.c".

>> Compilation failure
```

Verbose Diagnostics

- Option --verbose_diagnostics
- Echoes problem source line
  - A caret ^ marks the critical point in the line
- Continuing the previous example …

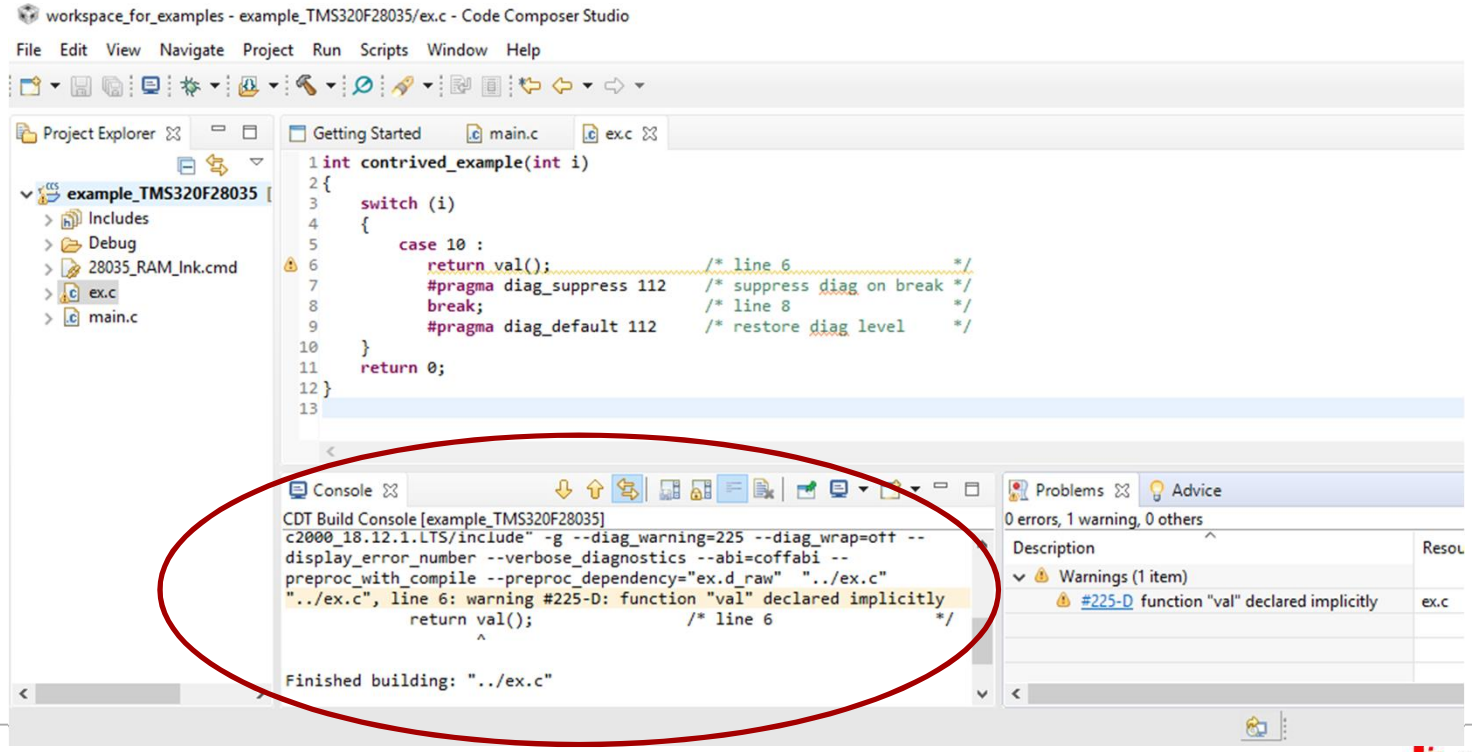
```
C:\dir>cl430 --diag_error=225 --verbose_diagnostics ex.c
"ex.c", line 6: error: function "val" declared implicitly
          return val();                 /* line 6                  */
                 ^

1 error detected in the compilation of "ex.c".

>> Compilation failure
```

Verbose Diagnostics

- Appear in CCS Console view



Agenda

- Resources
- Compiler Options
- MSP430 Specifics
- Diagnostics
- **Other Tips**

Other Tips

- Functions in RAM
- Type Sizes
- Compiler Version Numbers
- Avoid printf
- Linker Command File
- Program Flash Memory

Functions in RAM

- On systems that have FLASH and RAM
- Executing code from RAM is faster
- But all the code does not fit in RAM
- Solution: Run only the most critical functions from RAM
- Two Methods
  - Function attribute ramfunc
  - Build option --ramfunc=on

Functions in RAM: Two Methods

- Function attribute ramfunc

```
__attribute__((ramfunc))
int ramfunc_example(int arg)
{
    /* code here */
}
```

- Build option --ramfunc=on
  - All functions in the source file run from RAM
  - Avoid modifying source
  - Apply only to certain files, and not entire project
  - Not enough RAM for that

Functions in RAM: Details

- Requires special code in linker command file
  - Already provided
- Functions are allocated in both FLASH and RAM
  - Load allocation FLASH
  - Run allocation RAM
- Startup code automatically copies from FLASH to RAM before main starts
  - No special initialization steps
  - Startup code provided in compiler RTS library
- Only the startup code knows about RAM functions being in FLASH too
- All other functions act as if these functions are always in RAM

Other Tips

- Functions in RAM
- **Type Sizes**
- Compiler Version Numbers
- Avoid printf
- Linker Command File
- Program Flash Memory

Type Sizes by CPU

Type	ARM	MSP430	C28x	C28x CLA
char	8	8	16	16
short	16	16	16	16
int	32	16	16	32
long	32	32	32	32
long long	64	64	64	32
float	32	32	32	32
double	64	64	32	32
long double	64	64	64	32

- Shaded sizes are not what programmers usually expect

Standard Type Names

```
#include <stdint.h>
```

- Use standardized type names from <stdint.h>

Type	Means
int32_t	signed, exactly 32-bits
int16_t	signed, exactly 16-bits
int_fast16_t	signed, fastest type that is at least 16-bits
intptr_t	signed, wide enough to hold a pointer

Other Tips

- Functions in RAM
- Type Sizes
- **Compiler Version Numbers**
- Avoid printf
- Linker Command File
- Program Flash Memory

Compiler Version Numbers

- Release numbers are of the form YY.MM.P.STS or YY.MM.P.LTS
  - Example: 18.12.2.LTS
- YY – Year of the first release
- MM – Month of the first release
- P – Patch number
  - Releases which vary only P differ only in bug fixes
- STS – Short term support
  - Supported for 3 months
  - Introduce new features
- LTS – Long term support
  - Supported for 1-2 years
  - Ever more stable over time

Compiler Version Numbers

- http://software-dl.ti.com/ccs/esd/documents/sdto_cgt_lts-and-sts-compiler-releases.html (link)
- Use STS releases to get new features quickly
- Use LTS releases for more stability
  - Prefer the highest P available. Has the most bugs fixed.

Other Tips

- Functions in RAM
- Type Sizes
- Compiler Version Numbers
- **Avoid printf**
- Linker Command File
- Program Flash Memory

Avoid printf

- Classic first program

```
main() { printf("hello, world\n"); }
```

- Fine for hosted systems
- Bad for embedded systems
- Requires lots of memory
- More than you have?
- Only runs under CCS
- Does your system HW include terminal output?
- Any C I/O operation stops CPU execution
- The host OS takes over to perform the low level I/O
- Can your system withstand that breakpoint?

Still like printf?

- Tips on making it work
  http://software-dl.ti.com/ccs/esd/documents/sdto_cgt_tips_for_using_printf.html (link)
- Option to reduce memory needed
  - Use --printf_support=*mode*
  - Valid modes: minimal, nofloat, full
  - Also reduces what can be printed
  - Details in compiler manual
- Alternatives
  - TI-RTOS log_printf
  - UART examples in MSP430ware and C2000Ware

Other Tips

- Functions in RAM
- Type Sizes
- Compiler Version Numbers
- Avoid printf
- **Linker Command File**
- Program Flash Memory

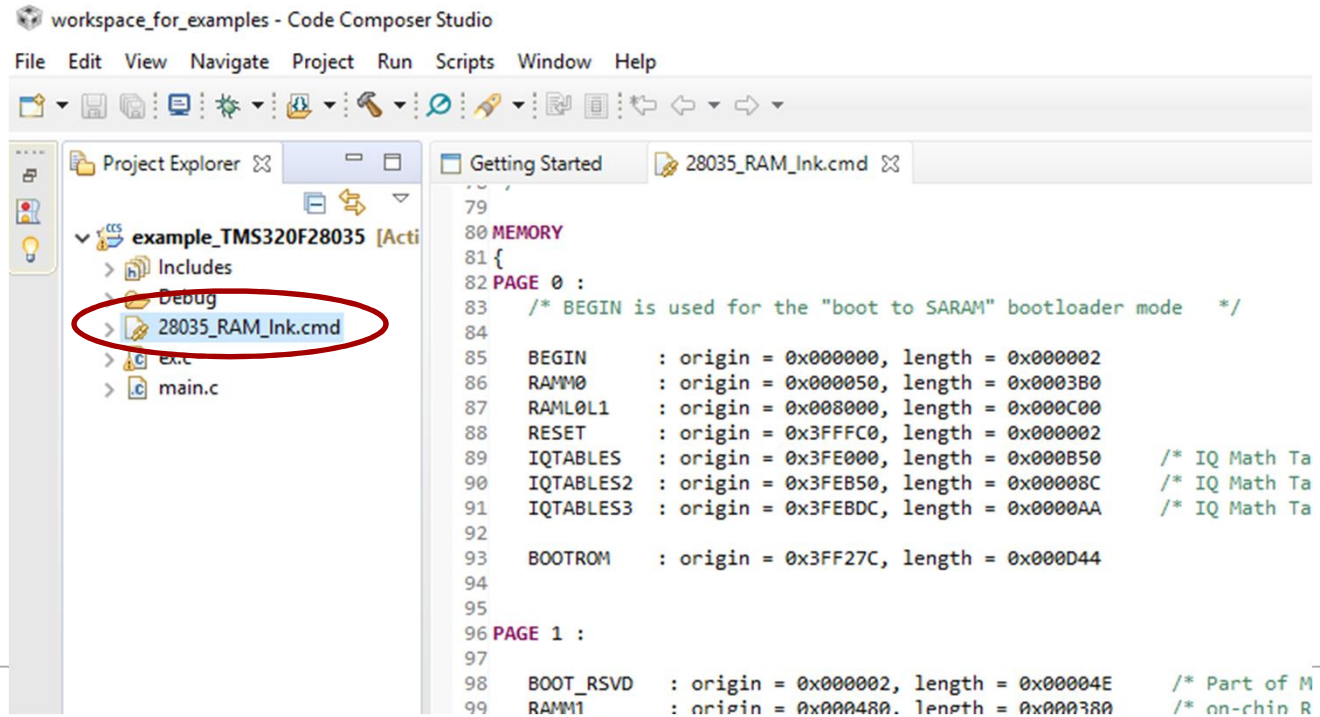
Linker Command File

- Specifies to the linker
  - Memory layout
  - How to form output sections
  - Where to put output sections in memory
- You use them all the time without knowing it

```
% gcc -Wl,--verbose
… snip …
SECTIONS
{
  /* Make the virtual address and file offset synced if the alignment is
     lower than the target page size. */
  . = SIZEOF_HEADERS;
  . = ALIGN(__section_alignment__);
  .text   __image_base__ + ( __section_alignment__ < 0x1000 ? . :
… snip …
```

Linker Command File

- Rare to write your own
- Usually supplied when start a new CCS project



Linker Command File

- Good overview
  http://software-dl.ti.com/ccs/esd/documents/sdto_cgt_Linker-Command-File-Primer.html (link)
- Full documentation in Linker chapter of Assembly Language Tools Reference Guide
  - http://www.ti.com/lit/pdf/slau131 (link)

Other Tips

- Functions in RAM
- Type Sizes
- Compiler Version Numbers
- Avoid printf
- Linker Command File
- **Program Flash Memory**

Program Flash Memory

- When CCS loads code and data, it automatically programs the on-chip flash
  - No special steps are needed
- But how do you program the flash on thousands of devices? Not with CCS!
- Typically done with a gang programmer
  - Programs many devices at once

Program Flash Memory

- Gang programmers cannot read the .out file produced by TI compiler
- Instead, they expect ASCII hex representation
- Several different formats in use. This example is TI-TXT format …

```
@5c00
31 40 00 5C B0 13 92 5C 0C 93 02 24 B0 13 1A 5C
0C 43 B0 13 88 5C B0 13 96 5C 1A 14 3F 40 00 00
3F 90 01 00 22 28 3F 40 00 00 3F 90 01 00 1D 28
3A 40 00 00 3A 80 00 00 3A 50 07 00 5A 09 39 40
...
```

Address

Code and Data

- Create ASCII hex representation with hex utility
- Available from within CCS
- http://software-dl.ti.com/ccs/esd/documents/ccs_hex-utility.html (link)

Questions?