C28x Compiler Tips & Tricks

George Mock

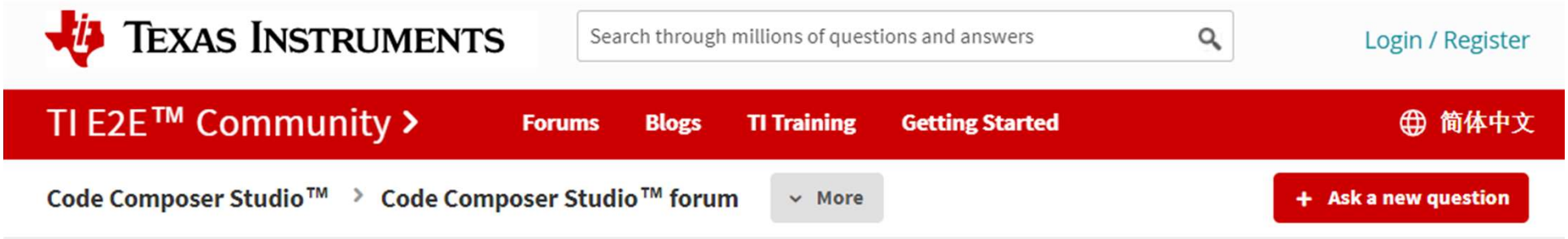
August 2019

Agenda

- Resources
- Compiler Options
- Compiler Qualification Kit
- Diagnostics
- Other Tips

Online Resource - CCS Forum

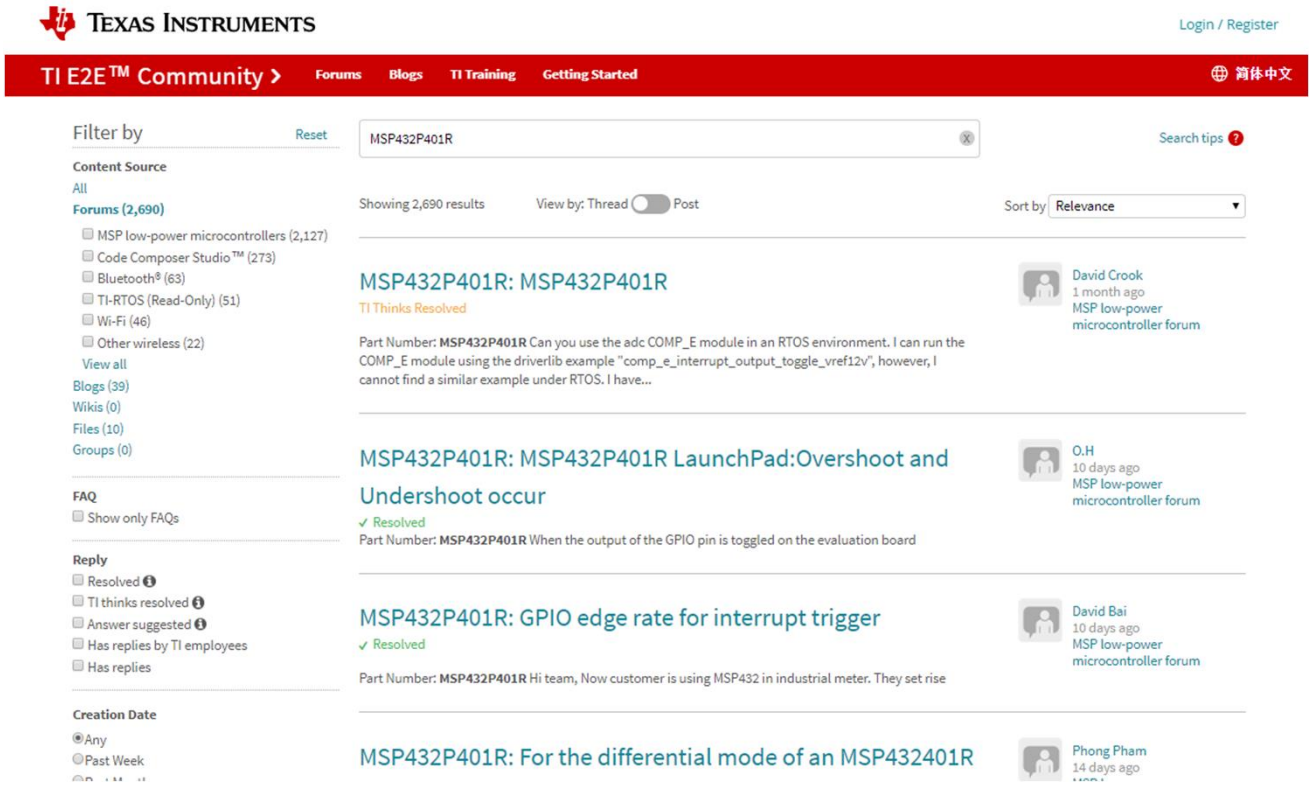
- http://e2e.ti.com/support/tools/ccs/f/81(link)
- Questions about CCS and Compiler
- Many responses are the same business day
- Thousands of posts to search



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.

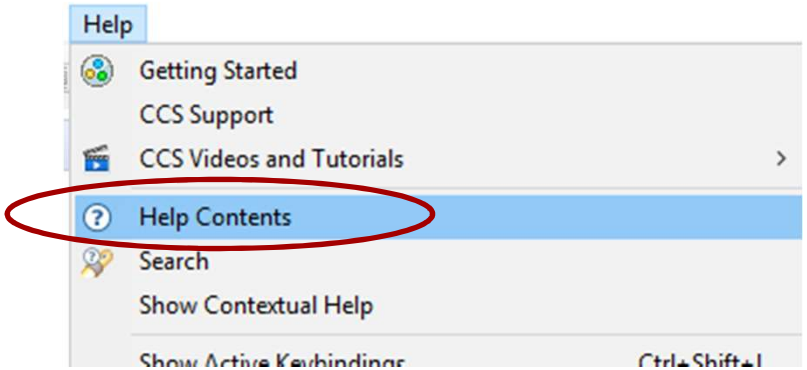
Eclipse Concepts	Information on various concepts that are part of the Eclipse/CCS environment
Getting Started View	The Getting Started View in CCS
Resource Explorer	Resource Explorer helps you find all the latest examples, libraries, demo applications, datasheets, and more for your chosen platform
App 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!

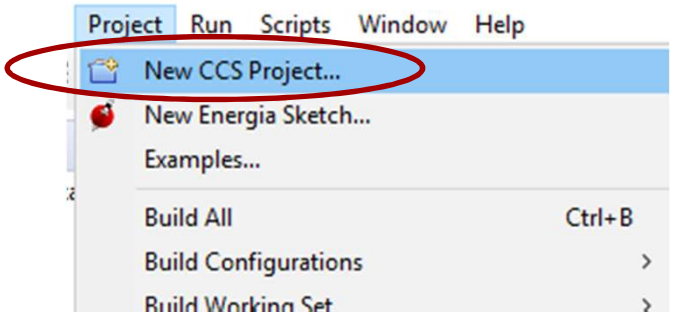
- Latest information on compiler
- Critical details
- Well worth the 30-60 minutes it takes to read it
- In root directory of compiler install
- Typical path
  - C:\ti\ccs901\ccs\tools\compiler\ti-cgt-c2000_18.12.1.LTS\README.txt

Agenda

- Resources
- **Compiler Options**
- Compiler Qualification Kit
- Diagnostics
- Other Tips

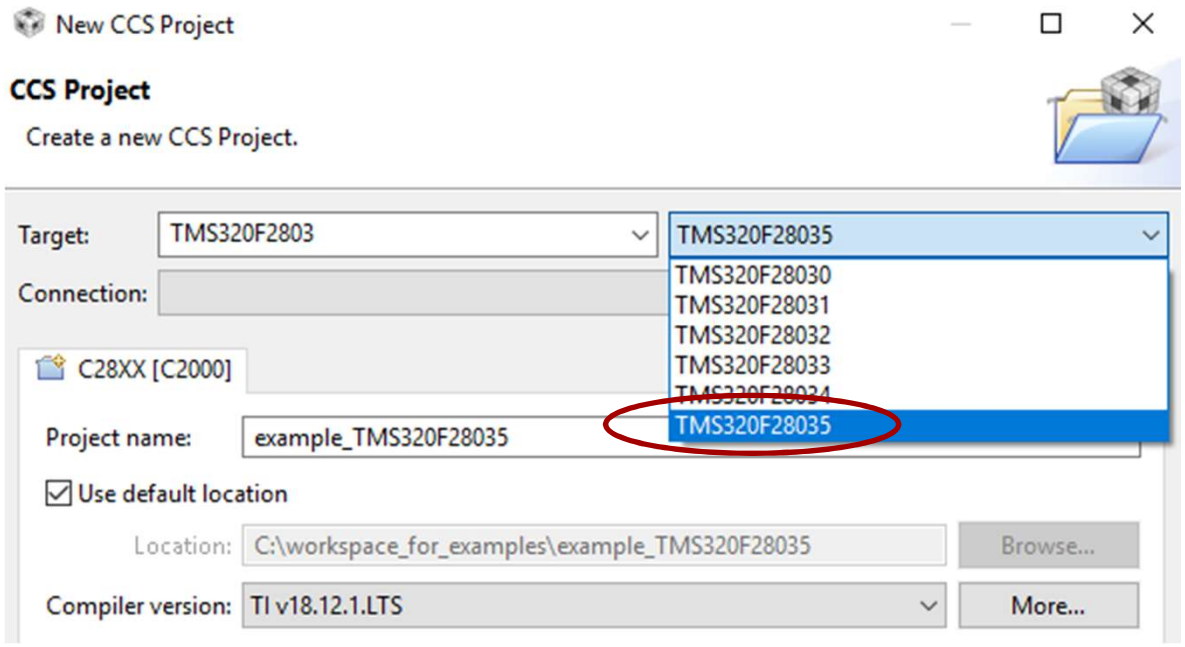
Compiler Options Chosen by CCS

- When you start a new project in CCS



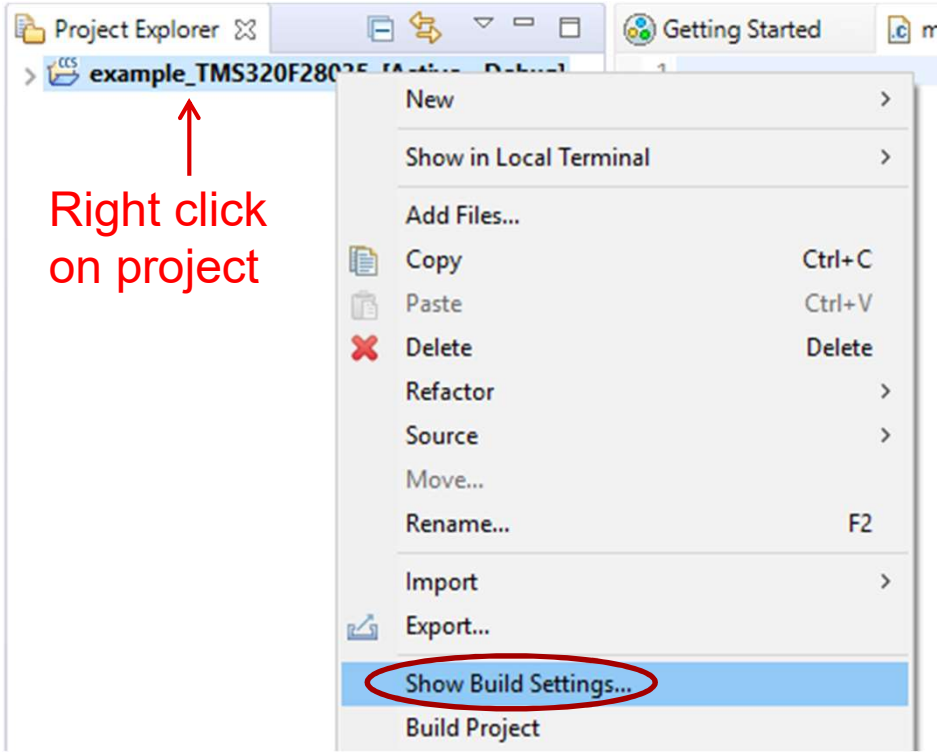
Compiler Options Chosen by CCS

- You specify the processor



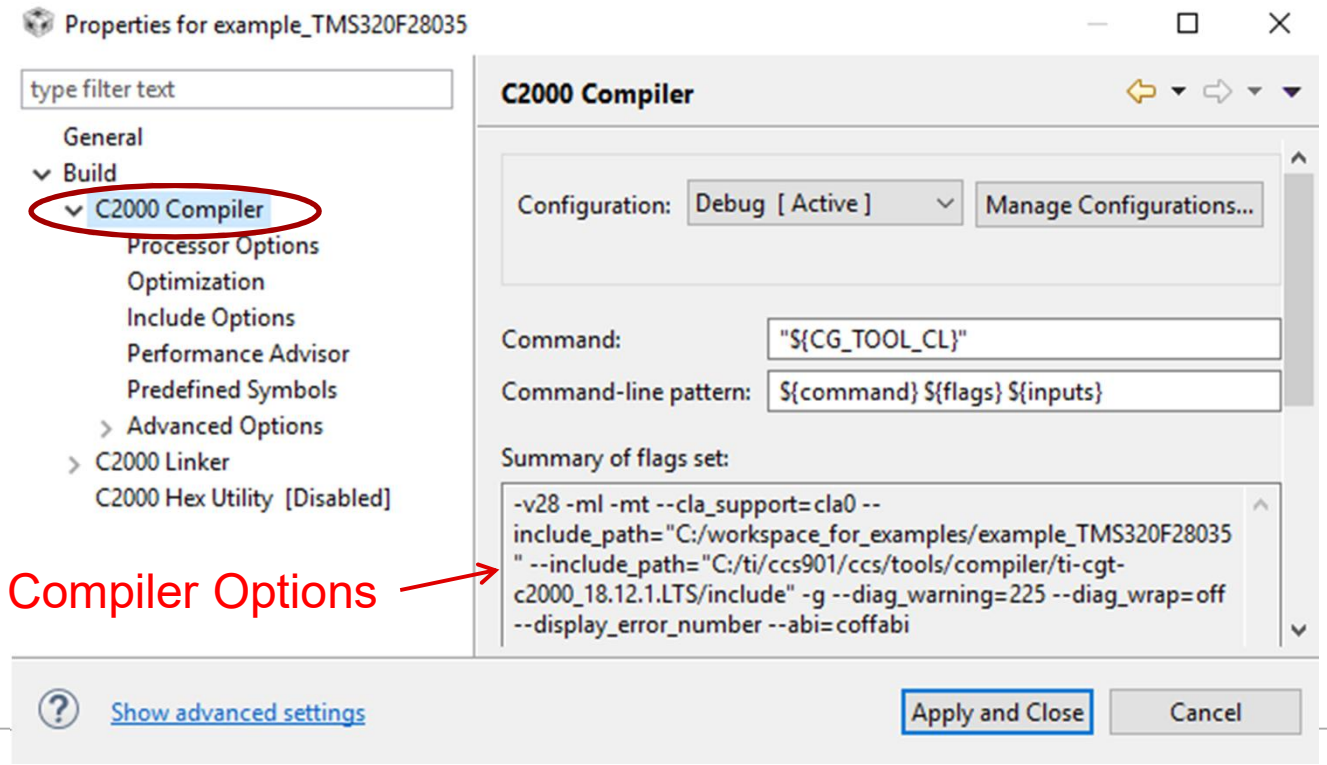
Compiler Options Chosen by CCS

- 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



Notable Compiler Options - C28x

- --unified_memory
  - All memory is available via program and data buses
  - True of nearly all C28x systems
    - If there is the slightest doubt, check the data sheet
  - Enables MAC instructions and more
- --fp_reassoc
  - On by default
  - Only affects floating point operations
  - Means it is OK to change `(a+b)+c` to `a+(b+c)`
  - Can affect precision
  - Required for compiler to effectively use MACF32 and related instructions
  - To disable use --fp_reassoc=off

C28x Hardware Features

- --float_support=fpu32
  - Hardware floating-point instructions
  - Enables minimal overhead loops
    - RPTB: repeat block instruction
    - No overhead per loop. 1-4 cycles loop setup overhead.
- --tmu_support=*type*
  - TMU: Trigonometric Math Unit
  - Valid types: **`tmu0, tmu1`**
  - Supports these floating point operations in HW
    - Division, sqrt, sin, cos, atan, atan2
  - Requires --fp_mode=relaxed
    - Because HW results are slightly different from standard RTS calls

C28x Hardware Features

- --vcu_support=*type*
  - VCU: Viterbi, Complex math, and CRC Unit
  - Valid types: **`vcu0, vcu2, vcrc`**
  - Only for assembly source

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 - C28x

- Default: --opt_level=off

Link Time Optimization --opt_level=4

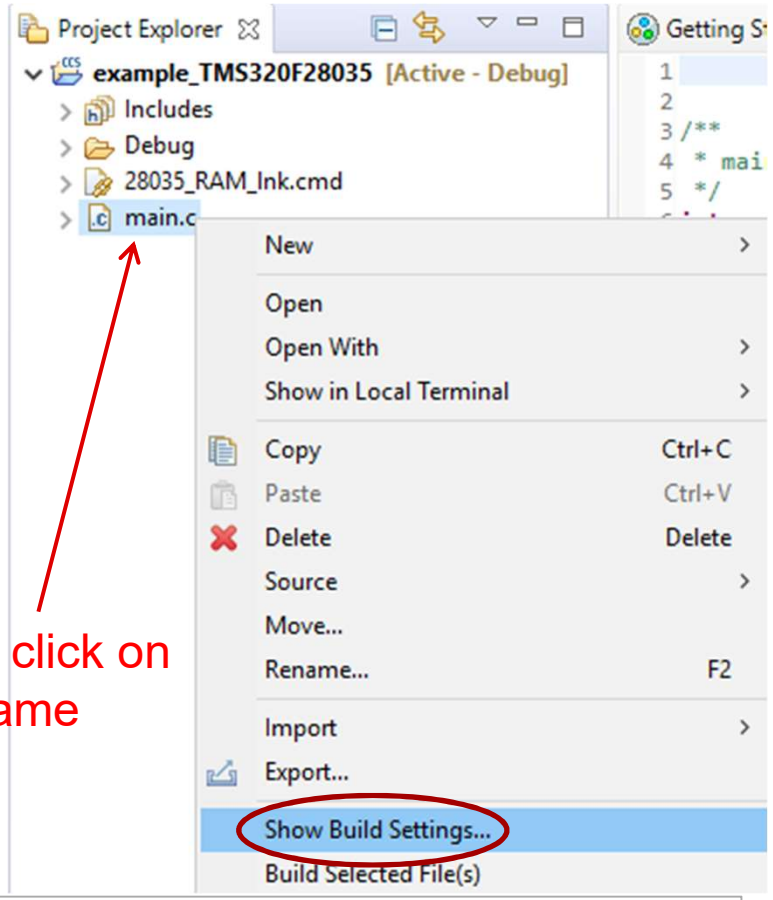
- Optimizes across the entire program
- Linking takes longer
- Presents opportunities rarely seen within files
  - May see all the calls to a function
  - If one argument is always the same, just replace it
- Use --opt_level=4 during compile and link
  - CCS takes care of this for you
- Information encoded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

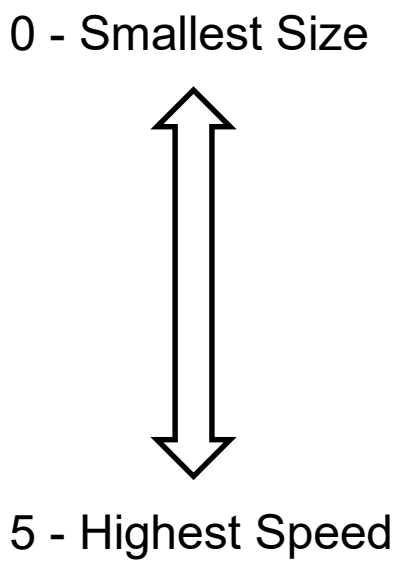
File Specific Options

- 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)



Optimize for Speed or Space

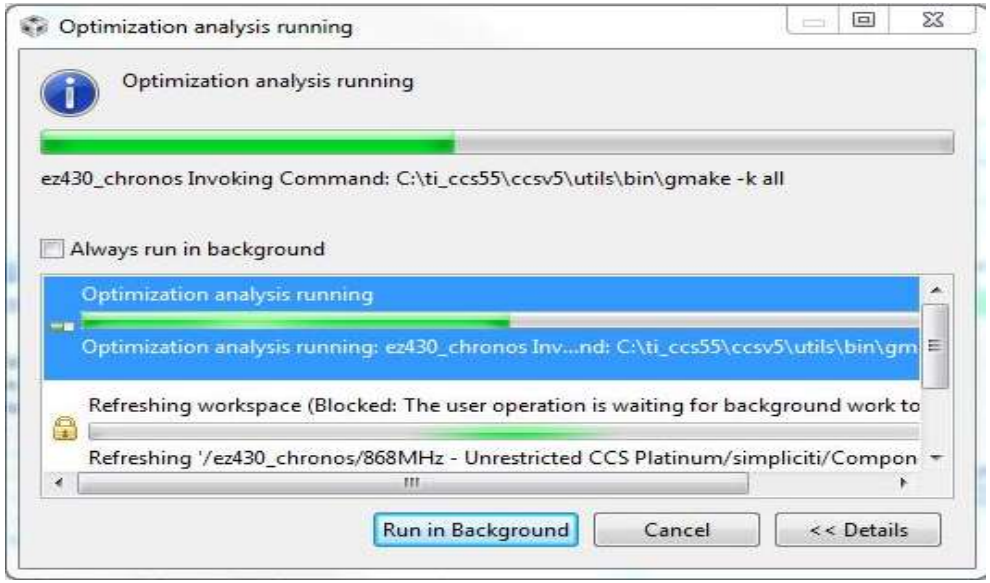
- Most optimizations improve both speed and space
- But some optimizations improve one while degrading the other
  - Loop unrolling
  - Function inlining
- Control with the build option --opt_for_speed=*value*
- Valid values 0-5
- Best choice is often the largest size that still fits



Optimizer Assistant

- What is the best setting of --opt_for_speed? --opt_level?
- It depends on the requirements regarding the following
  - Speed
  - Size of memory used
  - Power consumed
- Finding the best setting requires experimenting with those options
- It is easy to do the experiment with Optimizer Assistant
  - A feature built into CCS

Optimizer Assistant in Action



- Project is built multiple times with different --opt_for_speed settings
  - Or different --opt_level settings
- Remembers how much flash memory is needed for each setting

Optimizer Assistant

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

Agenda

- Resources
- Compiler Options
- **Compiler Qualification Kit**
- Diagnostics
- Other Tips

SafeTI™ Compiler Qualification Kit

- Assists in qualifying the TI C/C++ Compiler to functional safety standards
- Validations are based on compiler release validations (instead of the user executing validations)
- Assessed by TÜV Nord to comply with IEC 61508, and ISO 26262
- Applicable to TI C/C++ Compiler for ARM/C2000/C6000
- Includes:
  - User Guide
  - Qualification documentation templates:
    - Tool Qualification Plan
    - Tool Qualification Report
    - Tool Safety Manual
  - Tool Definition Report
  - Tool Validation Report
  - Tool Process Compliance Report
  - Instrumented Version of Tool
  - TÜV Nord CQKIT assessment report
  - Free
  - For Validas support http://www.validas.de/en/contact/

Approved by TÜV NORD

http://www.ti.com/tool/safeti_cqkit

Texas Instruments

Agenda

- Resources
- Compiler Options
- Compiler Qualification Kit
- **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 …

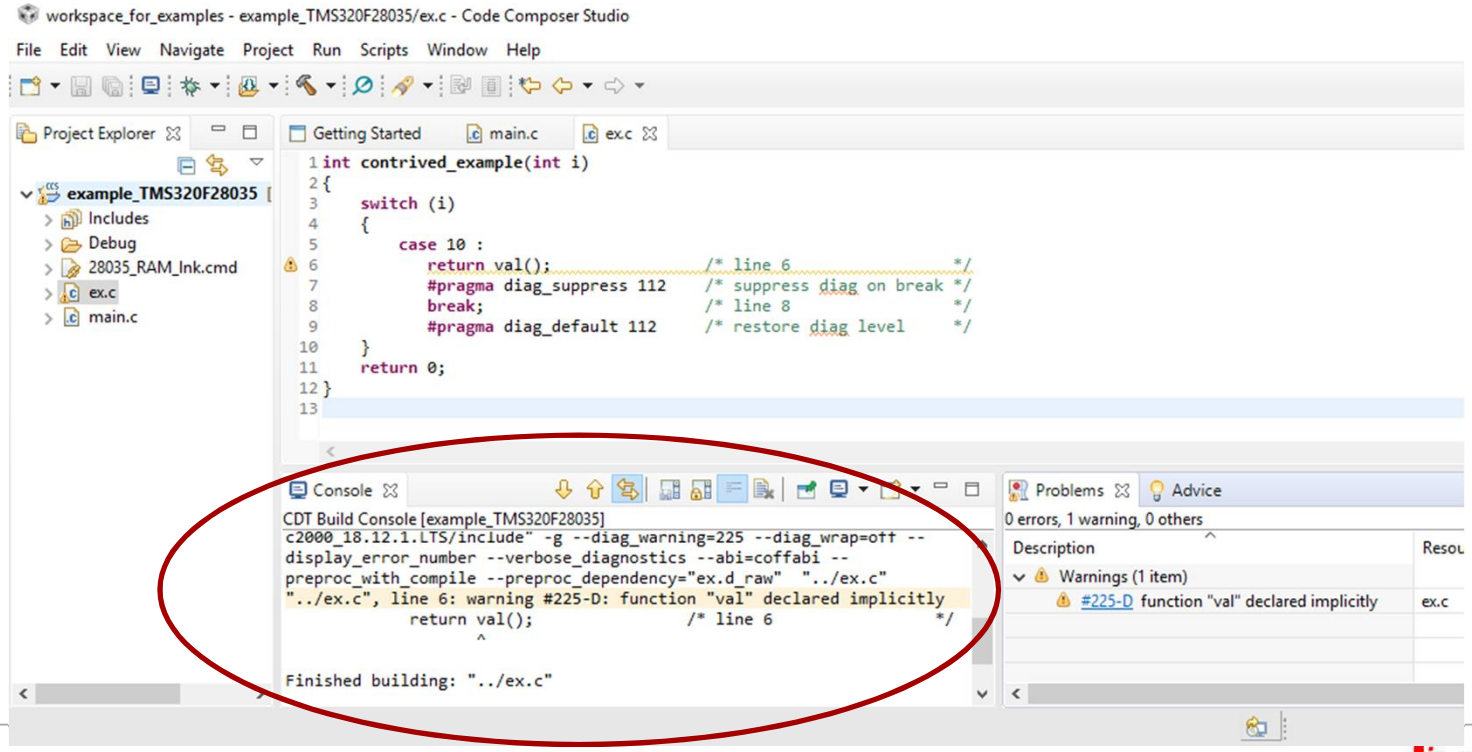
```
C:\dir>cl430 --diag_error=225 --verbose_diagnostics ex.c
"ex.c", line 6: error: function "val" declared implicitly
          return val();                 /* line 6                  */
                 ^

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

>> Compilation failure
```

Verbose Diagnostics

- Appear in CCS Console view



Agenda

- Resources
- Compiler Options
- Compiler Qualification Kit
- Diagnostics
- **Other Tips**

Other Tips

- Functions in RAM
- CLA
- Type Sizes
- Compiler Version Numbers
- Avoid printf
- Linker Command File

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

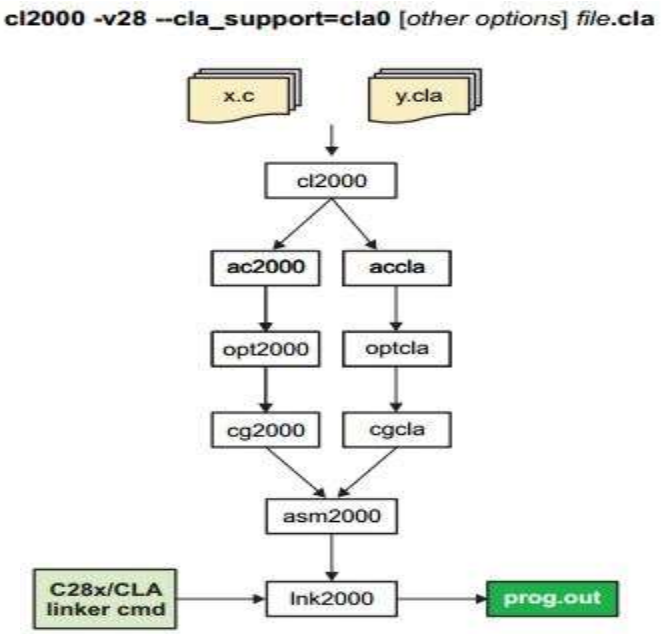
- Requires special code in linker command file
  - Already provided
- Functions are allocated in both FLASH and RAM
  - Load allocation FLASH
  - Run allocation RAM
- Startup code automatically copies from FLASH to RAM before main starts
  - No special initialization steps
  - Startup code provided in 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
- **CLA**
- Type Sizes
- Compiler Version Numbers
- Avoid printf
- Linker Command File

C28x CLA: Control Law Accelerator

- CLA is programmed with a subset of C
- Many standard C features are not supported
- Source files have the extension .cla
- Details in CLA chapter of C28x compiler manual



Other Tips

- Functions in RAM
- CLA
- **Type Sizes**
- Compiler Version Numbers
- Avoid printf
- Linker Command File

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

Type Sizes C28x

- char is 16-bits, not 8-bits
  - Affects exchanging data with 8-bit byte systems
- sizeof(int) == sizeof(char) == 1
- Does not violate ANSI standard
  - Only specifies minimum widths, not maximum
- http://processors.wiki.ti.com/index.php/Byte_Accesses_with_the_C28x_CPU (link)
- CLA nothing > 32 bits

Include <stdint.h>

```
#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
- CLA
- Type Sizes
- **Compiler Version Numbers**
- Avoid printf
- Linker Command File

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
- CLA
- Type Sizes
- Compiler Version Numbers
- **Avoid printf**
- Linker Command File

Avoid printf

- Classic first program

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

- Fine for hosted systems
- Bad for embedded systems
- Requires lots of memory, especially stack
- 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
- CLA
- Type Sizes
- Compiler Version Numbers
- Avoid printf
- **Linker Command File**

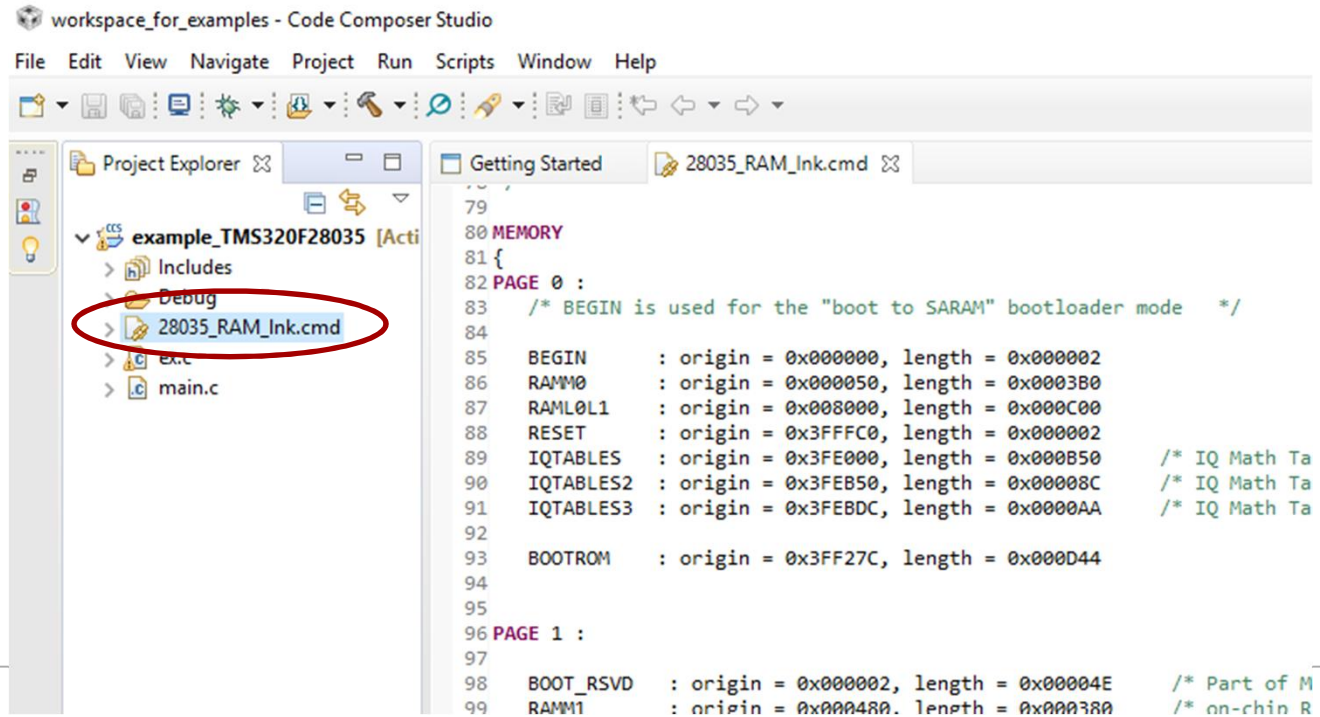
Linker Command File

- 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/spru513 (link)

Questions?

Backup Slides

C28x SafeTI™ Compiler Qualification Kit

- Assists customers in qualifying the TI C28x C/C++ Compiler to functional safety standards (ASIL-B or ASIL-C/D)
  - As applied by customers, allows demonstrating that their use of the compiler is compliant with safety standards

- QKIT documents/templates were developed in collaboration with **Validas** and approved by **TÜV Nord**
  - Validas is a consulting company specializing in qualification of software tools for functional safety development
  - QKIT content was assessed and approved by **TÜV Nord** to comply with IEC 61508 and ISO 26262

- Qualification can be based on ONE of the following:
  - *Process compliance*: Bare minimum for ASIL B qualification
    - Customers provided with evidence that a TI software dev process exists and is followed (*process-compliant*)
    - Customers use a latest (more mature) version of the compiler to increase confidence (*confidence-from-use*)

  - *Release validation*: Necessary for ASIL C/D qualification. Optional for ASIL-B qualification.
    - Compare compiler coverage for customer use cases against coverage of TI compiler release validations
      - This is done with an instrumented compiler; coverage data is sent to TI for assessment
      - TI adjusts its release validations to accommodate any observed coverage gaps
    - More comprehensive than process compliance

  - Both require: Locking compiler version/options, track any new issues, review and assess known compiler bugs

C28x SafeTI™ Compiler Qualification Kit

- **Includes**
  - QKIT User's Guide
  - Qualification documentation templates (filled out by customer):
    - Tool Qualification Plan
    - Tool Qualification Report
    - Tool Safety Manual
  - Tool Definition Report
  - Tool Validation Report
    - Internal compiler release validation results
  - TÜV Nord CQKIT assessment approval report

- **Free of charge for TI customers**