TI ARM Compiler Tips & Tricks

George Mock

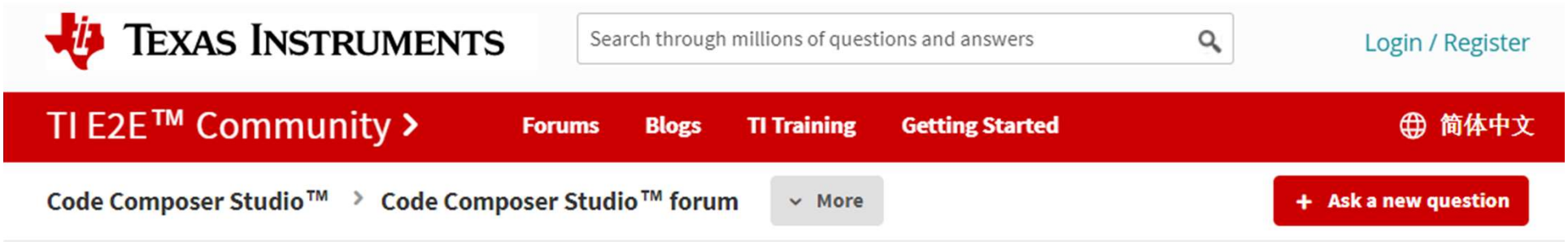
August 2019

Agenda

- Resources
- Compiler Options
- ARM FP Performance
- Diagnostics
- Other Tips

Online Resource - CCS Forum

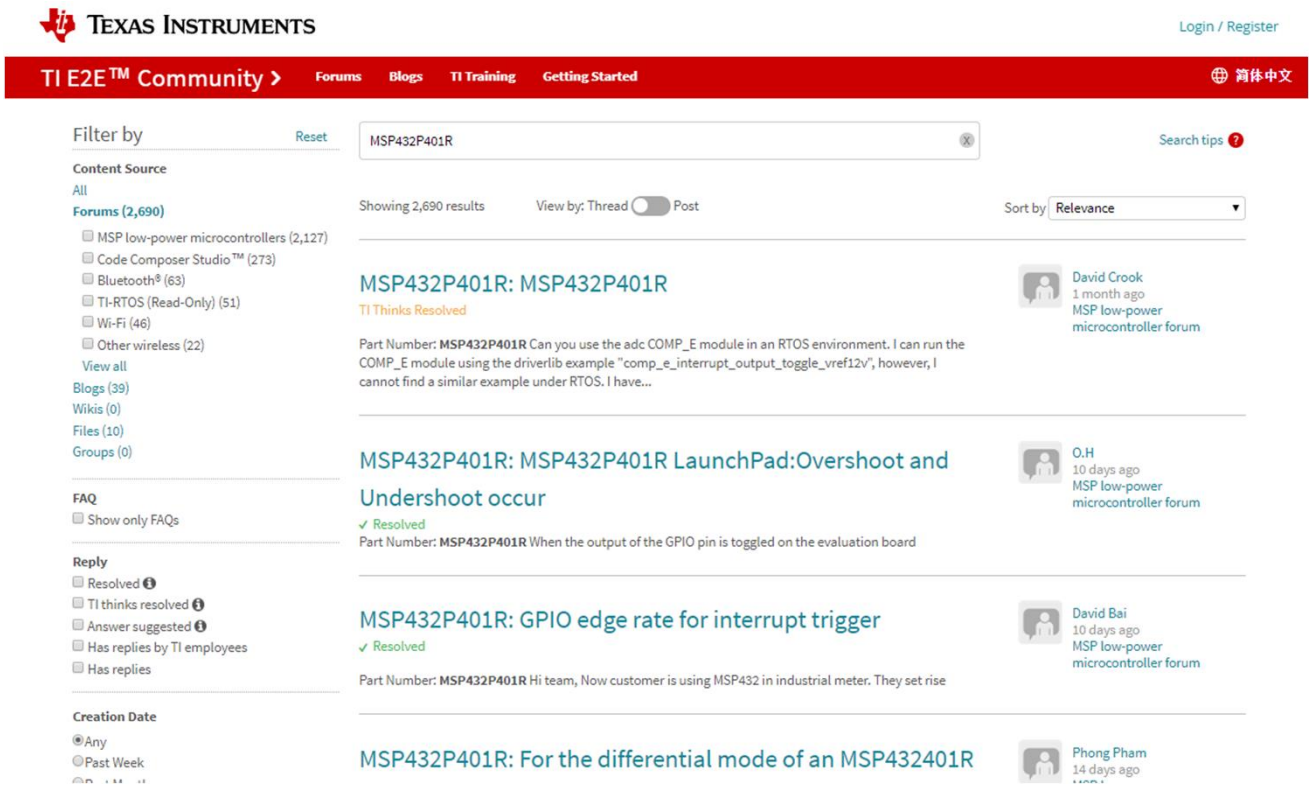
- http://e2e.ti.com/support/tools/ccs/f/81(link)
- Questions about CCS and Compiler
- Many responses are the same business day
- Thousands of posts to search



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.

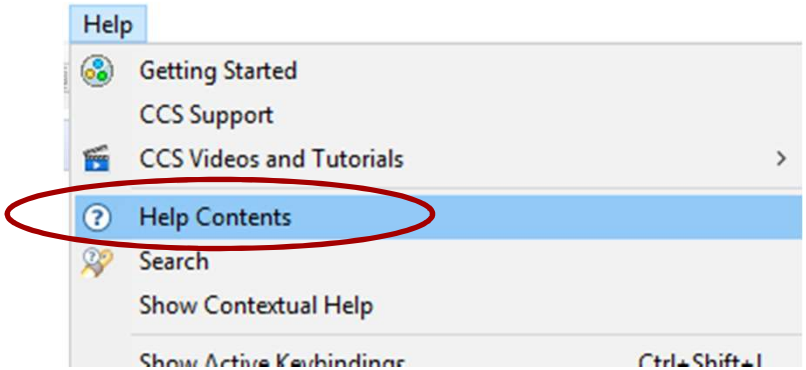
Eclipse Concepts	Information on various concepts that are part of the Eclipse/CCS environment
Getting Started View	The Getting Started View in CCS
Resource Explorer	Resource Explorer helps you find all the latest examples, libraries, demo applications, datasheets, and more for your chosen platform
App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

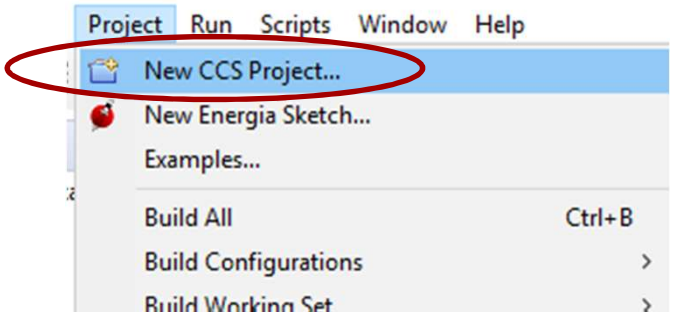
- Latest information on compiler
- Critical details
- Well worth the 30-60 minutes it takes to read it
- In root directory of compiler install
- Typical path
  - C:\ti\ccs901\ccs\tools\compiler\ti-cgt-arm_18.12.1.LTS\README.txt

Agenda

- Resources
- **Compiler Options**
- ARM FP Performance
- Diagnostics
- Other Tips

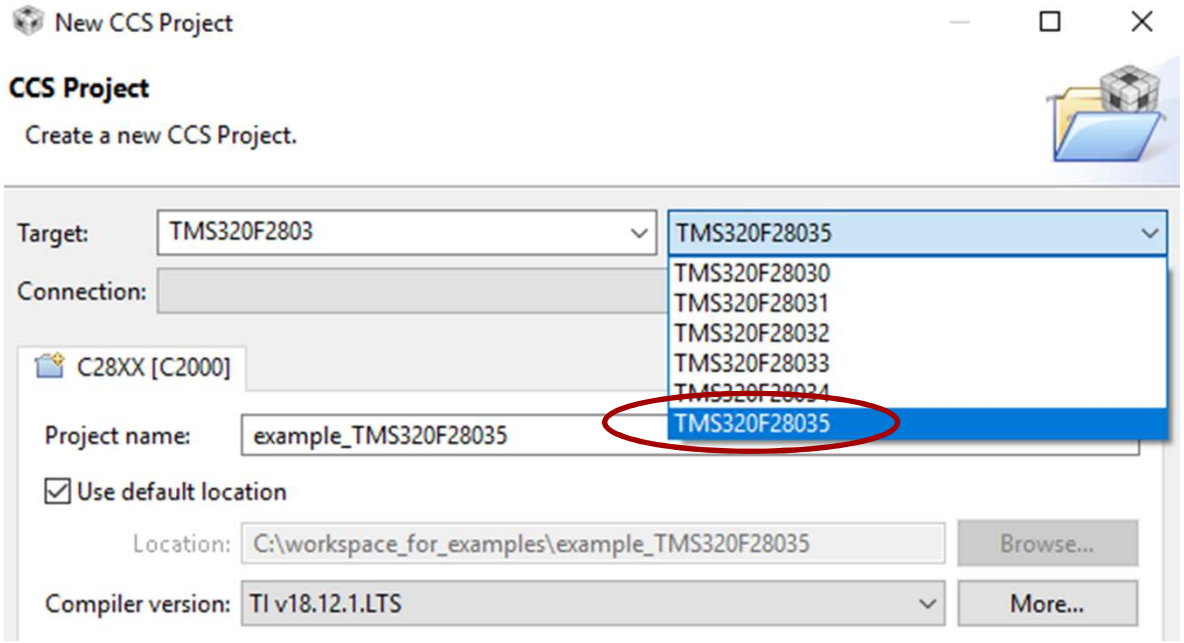
Compiler Options Chosen by CCS

- When you start a new project in CCS



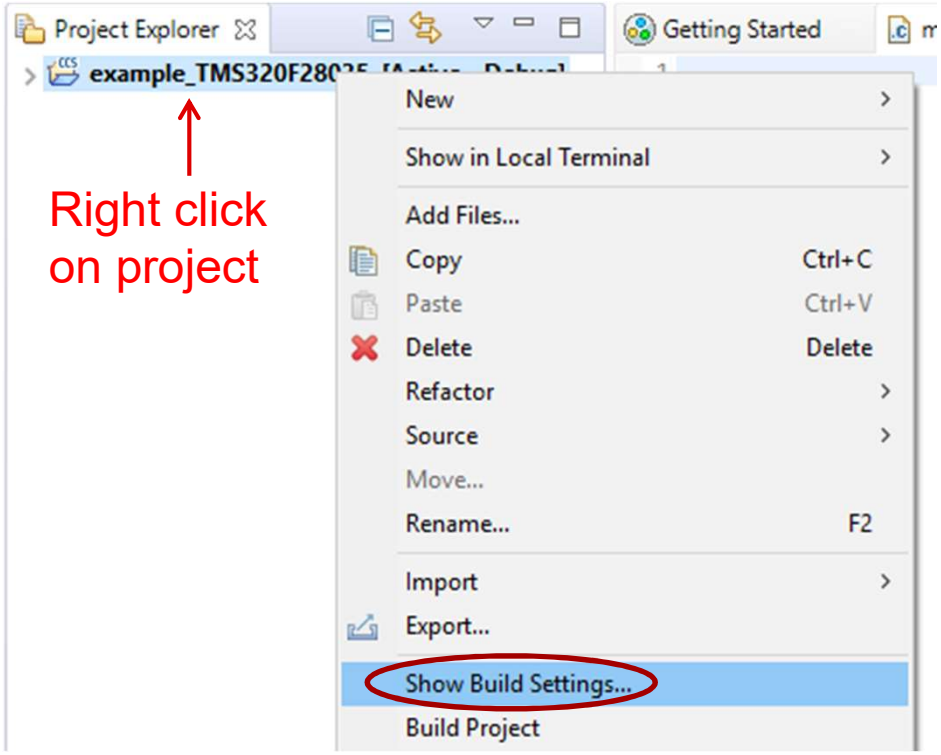
Compiler Options Chosen by CCS

- You specify the processor



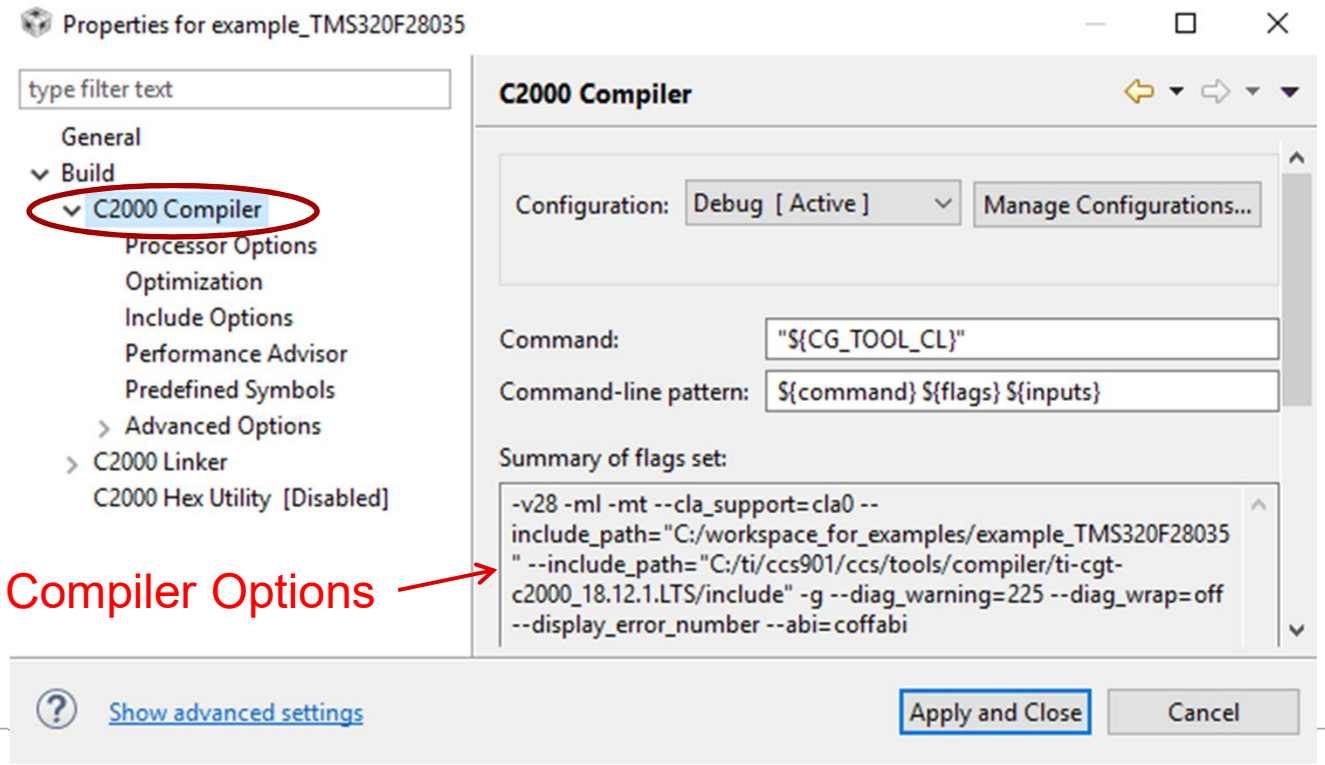
Compiler Options Chosen by CCS

- 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



Compiler Options

Notable Compiler Options - ARM

- Select ARM variant --silicon_version=*cpu*
  - Valid cpu's: 4, 5e, 6, 6M0, 7A8, 7M3, 7M4, 7R4, 7R5
  - Nothing higher than Cortex-A8
- ARM or Thumb instructions --code_state=*value*
  - Valid values: 16, 32
- Floating point HW --float_support=*vfp*
  - Which VFP (Vector Floating Point) Coprocessor?
  - Valid vfp's: VFPv2, VFPv3, VFPv3D16, vfplib, fpalib, FPv4SPD16, none
- CCS sets the options mentioned above

Select ARM Variant

Processor	Arch Ver	Switch	--code_state=16
ARM7	ARMv4T	--silicon_version=4	Thumb
ARM9	ARMv5TE	--silicon_version=5e	Thumb
ARM11	ARMv6	--silicon_version=6	Thumb
Cortex-M0	ARMv6M0	--silicon_version=6M0	Thumb2 only
Cortex-M3	ARMv7M3	--silicon_version=7M3	Thumb2 only
Cortex-M4	ARMv7M4	--silicon_version=7M4	Thumb2 only
Cortex-R4	ARMv7R4	--silicon_version=7R4	Thumb2
Cortex-R5	ARMv7R5	--silicon_version=7R5	Thumb2
Cortex-A8	ARMv7A8	--silicon_version=7A8	Thumb2

- Default is ARM7. Avoid that!
- Cortex only supports Thumb2
  - Combines 16-bit and 32-bit instructions
  - M0, M3 and M4 do not support 32-bit ARM mode

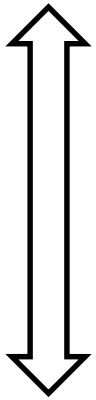
ARM or Thumb Instructions

- ARM --code_state=32
  - Invalid for Cortex-M0, Cortex-M3 and Cortex-M4
- Thumb --code_state=16
  - Thumb2 on Cortex
  - No effect for Cortex-M0, Cortex-M3 and Cortex-M4
- --code_state=16 usually best for Cortex-R4
  - Integer divide instructions
  - Can use higher --opt_for_speed for the same code size
  - Due to higher code density, CPU can prefetch more instructions

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

0 - Smallest Size



5 - Highest Speed

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 - ARM

- No –g? --opt_level=3
- Use –g? --opt_level=off
- Use of –g implies intention to debug, which is made easier with lower optimization
- Advice: Always explicitly specify --opt_level

Link Time Optimization --opt_level=4

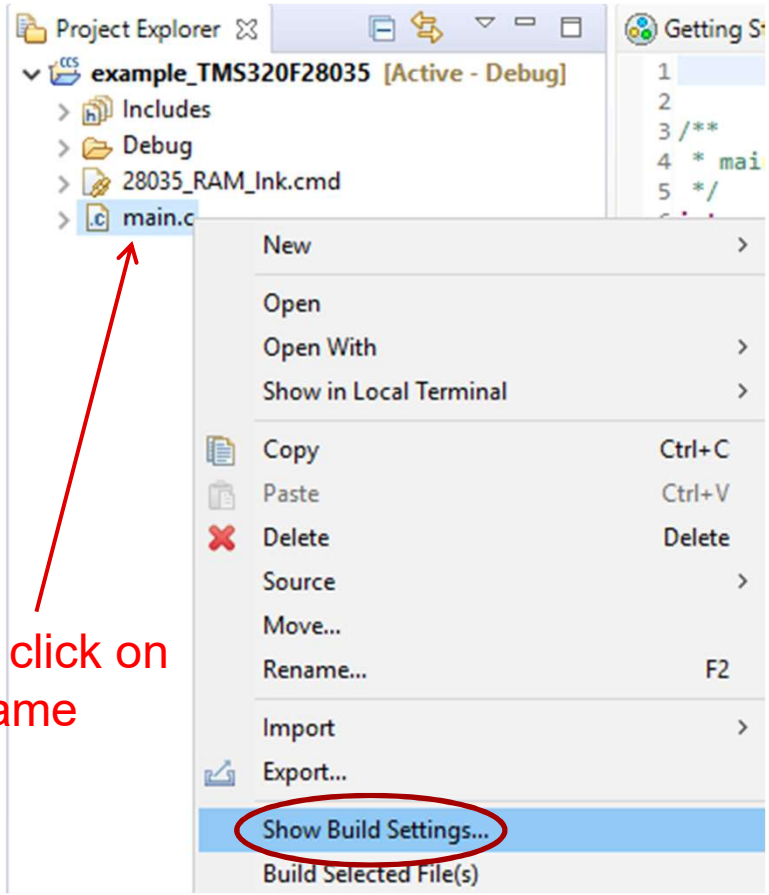
- Optimizes across the entire program
- Linking takes longer
- Presents opportunities rarely seen within files
  - May see all the calls to a function
  - If one argument is always the same, just replace it
- Use --opt_level=4 during compile <u>and</u> link
  - CCS takes care of this for you
- Information 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
- **ARM FP Performance**
- Diagnostics
- Other Tips

ARM FP Performance - Avoid double

- FP ➔ Floating Point
- Avoid type double!
- Type double 64-bits
  - Precise but slow
- Type float 32-bits
  - Less precision, but often good enough. Much faster.
- Hard to avoid type double in C

ARM FP Performance - Avoid double

- What is the type for 1.1?

```
float f1, f2;
f1 = f2 + 1.1;
```

ARM FP Performance - Avoid double

- What is the type for 1.1? double

```
float f1, f2;
f1 = f2 + (double) 1.1;
```

- What is the type for the addition?

ARM FP Performance - Avoid double

- What is the type for 1.1? double

```
float f1, f2;
f1 = (double) f2 + (double) 1.1;
```

- What is the type for the addition? double
- What is the type for the assignment?

ARM FP Performance - Avoid double

- What is the type for 1.1? double

```
float f1, f2;
f1 = (float) ((double) f2 + (double) 1.1);
```

- What is the type for the addition? double
- What is the type for the assignment? Float

ARM FP Performance - Avoid double

- Typical fix: use float suffix on the constant

```
float f1, f2;
f1 = f2 + 1.1f;
```

- Causes entire operation to done with type float

ARM FP Performance - Avoid double

- RTS functions default to double. Prefer float.

```
f1 = sin(f2);  // converts to and from double
f1 = sinf(f2); // no conversions
```

ARM FP Performance – Avoid double

- In C, double is to floating point math what int is to integer math
- Use --float_operations_allowed=32 to find hidden double operations
  - --float_operations_allowed=*value*
  - Supported values: none, 32, 64, all
  - See error when sizeof(float_operation) > *value*

Agenda

- Resources
- Compiler Options
- ARM FP Performance
- **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 <u>not</u> 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

workspace_for_examples - example_TMS320F28035/ex.c - Code Composer Studio

File Edit View Navigate Project Run Scripts Window Help

Project Explorer

example_TMS320F28035
- Includes
- Debug
- 28035_RAM_lnk.cmd
- ex.c
- main.c

Getting Started | main.c | ex.c

```
1 int contrived_example(int i)
2 {
3     switch (i)
4     {
5         case 10 :
6             return val();                /* line 6                  */
7             #pragma diag_suppress 112    /* suppress diag on break */
8             break;                       /* line 8                  */
9             #pragma diag_default 112     /* restore diag level      */
10    }
11    return 0;
12 }
13
```

Console

CDT Build Console [example_TMS320F28035]

```
c2000_18.12.1.LTS/include" -g --diag_warning=225 --diag_wrap=off --
display_error_number --verbose_diagnostics --abi=coffabi --
preproc_with_compile --preproc_dependency="ex.d_raw"  "../ex.c"
"../ex.c", line 6: warning #225-D: function "val" declared implicitly
            return val();                /* line 6                  */
                   ^

Finished building: "../ex.c"
```

Problems | Advice

0 errors, 1 warning, 0 others

Description	Resou
Warnings (1 item)	
#225-D function "val" declared implicitly	ex.c

Agenda

- Resources
- Compiler Options
- ARM FP Performance
- Diagnostics
- **Other Tips**

Other Tips

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

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

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

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, C2000Ware and SimpleLink SDK

Other Tips

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

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

workspace_for_examples - Code Composer Studio

File Edit View Navigate Project Run Scripts Window Help

Project Explorer

example_TMS320F28035 [Acti

Includes

Debug

28035_RAM_lnk.cmd

main.c

Getting Started | 28035_RAM_lnk.cmd

```
79
80 MEMORY
81 {
82 PAGE 0 :
83    /* BEGIN is used for the "boot to SARAM" bootloader mode   */
84
85    BEGIN       : origin = 0x000000, length = 0x000002
86    RAMM0       : origin = 0x000050, length = 0x0003B0
87    RAML0L1     : origin = 0x008000, length = 0x000C00
88    RESET       : origin = 0x3FFFC0, length = 0x000002
89    IQTABLES    : origin = 0x3FE000, length = 0x000B50     /* IQ Math Ta
90    IQTABLES2   : origin = 0x3FEB50, length = 0x00008C     /* IQ Math Ta
91    IQTABLES3   : origin = 0x3FEBDC, length = 0x0000AA     /* IQ Math Ta
92
93    BOOTROM     : origin = 0x3FF27C, length = 0x000D44
94
95
96 PAGE 1 :
97
98    BOOT_RSVD   : origin = 0x000002, length = 0x00004E     /* Part of M
99    RAMM1       : origin = 0x000480, length = 0x000380     /* on-chip R
```

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/spnu118 (link)

Other Tips

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

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

Questions?