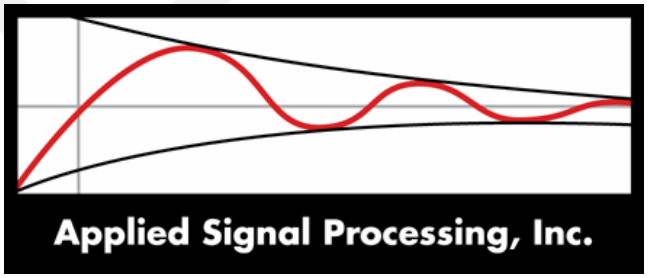
TI Developer Conference
March 7-9, 2007 • Dallas, TX

Minds in Motion

Improving Embedded Software Using Lint with Code Composer Studio™ (CCStudio) IDE

Jerry J. Trantow
Partner, Applied Signal Processing



Applied Signal Processing, Inc.

- Statement: Improving Embedded Software Using Lint with CCStudio.
- Education
  - BS-EE, CS UW-Madison 84
  - MS-EE UW-Madison 87
  - Math Minor, ABD EE UW-Madison
- Experience
  - Active Noise and Vibration Control (Automotive, Aerospace)
  - Acoustic Projects
  - File System Projects
- Bias
  - “Ancient”
  - TI Third Party (always looking for work!)
  - No affiliation with PC-Lint other than satisfied customer.

Target Audience

- DSP software developers
- Microcontroller software developers
- Software managers
- Project managers

Session Topics

- C Language Challenges - Why you need Lint*
- What Lint can do for you
- How to integrate Lint into CCStudio IDE
- How to configure Lint for your HW/Application
- Examples where Lint saved the day

*C++ is similar but not covered by this talk

Software Challenges

- A Byte is not necessarily 8 bits
- Various size of pointers, floats, doubles, long long, etc…
- Access to hardware registers (r/w, volatile, cregister, various width)
- Variables in ROM/Flash rather than RAM blocks with different performance characteristics
- Libraries with different development standards/styles

Table 5–1. TMS320C55x C/C++ Data Types

Type	Size	Representation	Minimum Value	Maximum Value
char, signed char	16 bits	ASCII	–32 768	32 767
unsigned char	16 bits	ASCII	0	65 535
short, signed short	16 bits	2s complement	–32 768	32 767
unsigned short	16 bits	Binary	0	65 535
int, signed int	16 bits	2s complement	–32 768	32 767
unsigned int	16 bits	Binary	0	65 535
long, signed long	32 bits	2s complement	–2 147 483 648	2 147 483 647
unsigned long	32 bits	Binary	0	4 294 967 295
long long	40 bits	2s complement	–549 755 813 888	549 755 813 887
unsigned long long	40 bits	Binary	0	1 099 511 627 775
enum	16 bits	2s complement	–32 768	32 767
float	32 bits	IEEE 32-bit	1.175 494e–38	3.40 282 346e+38
double	32 bits	IEEE 32-bit	1.175 494e–38	3.40 282 346e+38
long double	32 bits	IEEE 32-bit	1.175 494e–38	3.40 282 346e+38
pointers (data) small memory mode large memory mode	 16 bits 23 bits	Binary	0	 0xFFFF 0x7FFFFF
pointers (function)	24 bits	Binary	0	0xFFFFFF

What can Lint do? (for you)

- Static analysis tool
- Lint is a C preprocessor. Checks C/C++ source code and find bugs, glitches, inconsistencies, non-portable constructs, redundant code, and much more.
- Lint can look across multiple modules and enjoys an added perspective
- Does not replace compiler/linker
- Does not perform run time analysis

Integrating Lint to CCStudio

- Unit Lint command
- `Lint-nt.exe –u HW_options.lnt module0.c`
- Global Lint command
- `Lint-nt.exe HW_options.lnt *.lob`
- Modify HW_Options.lnt for processor
- Modify ApplicationSpecific.lnt for project

HW_options.lnt

- Data Types
  - -sb8, -si4 // C67xx
  - -sb16, -si1 // C54xx
  - -s32, -si1 // C3x, C4x
- Compiler specific predefined macros
  - _TMS320C6X, _LITTLE_ENDIAN
- Reserved keywords
  - inline
  - warn
  - Cregister
- Command line defines
  - _DEBUG
  - CHIP_6713
- Include Directories
- Message formatting and suppression
- Libraries can have different (less stringent) lint policy

ApplicationSpecific.Int

- Command line defines
- Application specific include paths
- Suppress messages about linker defined symbols
  - esym(526,_STACK_SIZE)
- Suppress messages about specific functions
  - esym(528,cos)
- Suppress messages about application variables and functions
  - esym(552,assert_line)

Lint Inline

- Possible to suppress specific messages within a module
- Suppress within a function or block of code
- Suppress for a line of code
  - **ASSERT**(0**==CheckFreeDiskSpace**());
    //lint!e666

Review

- Lint is a static analysis tool
- Easy to integrate unit and global Lint checks within CCStudio
- Initial flood of messages can be handled by correct option configuration
- Review of messages will probably stimulate coding changes
- Get rid of the false-positives detections
- Next: Subtle Coding Bugs

Real Life Examples

- Mixed size
- Operator precedence
- Structure initialization
- Overflow of shift
- Global wrap-up

Mixed sizes.

```
int function3(Uint32 a, int b)
{
    int ret_value;

   ret_value=a;
    return(ret_value);
}   //  function3().
```

"Module1.c", line 59: warning: Info 712: Loss of precision (assignment) (unsigned long to int)

Operator Precedence

```
#if (0)
    ret_value=a<<shift+1;   // Implies a<<(shift+1)
  #else
    ret_value=(a<<shift)+1;   // Desired.
  #endif
```

"Module1.c", line 73: warning: Info 701: Shift left of signed quantity (int)

"Module1.c", line 73: warning: Warning 504: Unusual shift operation (right side unparenthesized)

Structure Initialization

```
DMA_Config my_dma_config=
{
    0,0,0,0,0
};
```

"module2.c", line 21: warning: Info 785: Too few initializers for aggregate 'my_dma_config'

Overflow in shift constant

```
/*!
    Macro for building Q values from non-negative values of floating point x less than
     2^(15-Q).
*/
#if (0)
   #define QCONST16_POS(x,q) ( (Int16)(+0.5f+(x)*( 1<<(q))) )
#else
   #define QCONST16_POS(x,q) (Int16)min(32767,+0.5f+(x)*(1U<<(q)))
#endif

Int16 one_12=QCONST16_POS(1.0,12);
Int16 one_15=QCONST16_POS(1.0,15);
```

"Module0.c", line 41: warning: Warning 648: Overflow in computing constant for operation: 'shift left'

Global Wrapup

- "Module1.c", line 20: warning: Error 15: Symbol 'x_array' redeclared (size,precision) (line 39, file main.c)
- Info 765: external 'one_12' (line 40, file Module0.c) could be made static
- Info 714: Symbol 'one_12' (line 40, file Module0.c) not referenced
- Info 765: external 'one_15' (line 41, file Module0.c) could be made static
- Info 714: Symbol 'one_15' (line 41, file Module0.c) not referenced
- Info 765: external 'function2(unsigned long, int)' (line 16, file Module1.c) could be made static
- Info 714: Symbol 'function2(unsigned long, int)' (line 16, file Module1.c) not referenced

Summary

- Easy to integrate into CCStudio
- Option files can manage messages
- Incredibly inexpensive relative to cost of finding a single bug
- Regular use improves software coding quality
- Topics not covered
  - C++
  - Strong Types (Q math, enumerations)
  - Dynamic checks
  - ASSERT(), VERIFY()

References

- "Writing Solid Code", Steve Maguire, Microsoft Press.
- "Debugging Applications", John Robbins, Microsoft Press.
- PC-Lint for C/C++, Gimpel Software.
- "The C Programming Language", Kernighan and Ritchie, Prentice Hall Software Series.

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

TI warrants performance of its hardware products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are used to the extent TI deems necessary to support this warranty. Except where mandated by government requirements, testing of all parameters of each product is not necessarily performed.

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
Low Power Wireless	www.ti.com/lpw	Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265