MULTI Compiler v2023.5 Release Notes for Embedded ARM64

Green Hills Software
30 West Sola Street
Santa Barbara, California 93101
USA
Tel: 805-965-6044
Fax: 805-965-6343
www.ghs.com

LEGAL NOTICES AND DISCLAIMERS

PubID: release_notes_arm64-744926
Branch: http://toolsvc/branches/release-branch-2023-5-comp
Date: September 1, 2023

Contents

1. Changes in Version 2023.5 1

System Requirements 2
Windows 2
Linux 2
Product Compatibility 4
Feature Status Definitions 5
New Features and Enhancements 5
New Option to Set Default Buffer Size for stdio Libraries 5
Annex K Functions Available in All Language Dialects 5
Changes to Linker -C and -D Options 5
New Options to Treat Warnings as Errors 6
New Option to Control Whether dblink Errors Result in a Build Failure 6
Improved Debugging for C11, C++11, C++14, and C++17 6
New Option to Allow Flexible Array Members in All Dialects of C and C++ 7
MIRROR Attribute Added to Memory Map Definitions 7
Newly Supported ARM64 CPUs 7
Changes in Behavior 8
Map Files Exclude Non-Allocated Sections 8
Libraries Reject Zero as Second Byte of Multibyte Character in EUC and Shift-JIS 8
Increased Instrumentation with -check=memory,nilderef 8
Packing no longer allowed on non-POD classes 9
Removed Features 9
Deprecated Features 9
Code Generation Defects 9

2. Changes in Version 2023.1 11

New Features and Enhancements 12
New Supported Host: Windows 11 12
New DoubleCheck Assert Fail Intrinsic 12
Improved DWARF Information with -delete 12
gaddr2line DWARF Support 12
New gbincmp Options 12
libc++ Now Provided as a Standard C++ Library Alternative 13
New Attribute set_offset 13
Changes in Behavior 13
New Version of ind_io_arrays.c 13
Change in Linkage for Const-Qualified Variable Templates (C++ CWG Defect 2387) 14
BUFSIZ increased to 8192 on INTEGRITY 15
Removed Features 15
Deprecated Features 15
Code Generation Defects 15

3. Changes in Version 2022.1 17

New Features and Enhancements 18
New Option to Constrain Functions 18
New gstack Options 18
New Option for Floating-Point Divide Conversions 18
New Optimization Attribute and #pragma 18
New Pragma and Attribute Syntax for Confirming Section Placement of Variables 19
Default Optimization Levels 19
New CPU Support 19
Preventing Literal Data in Text 20
FPSR Status Bits Now Supported on simarm64 20
New DoubleCheck Warning 20
New Option to Remap Paths in DWARF Information 21
New Support for Splitting Relocatable Objects Across Memory Blocks 21
New POSIX Wide Character Conversion Functions 21

Improved Homogeneous Floating Point Aggregate Code Quality 21
Changes in Behavior 21
Change to Defaults Shown for Optimization Options in Builder . . . 21
Decreased Ability to Toggle Loop Optimizations 22
Optimization Strategies Selected for -O1, -O2, and -O3 23
Using -Olimit= Without Debugging Information No Longer Requires -glimits 23
Parallel Execution Policies No Longer Defined in std::execution 23
Options --fast=all and --small=all Now Apply to Only One Function 23
Change in Value for std::numeric_limits<floating-point-types>::has_signaling_NaN 24
Change in -MMD Option 24
Change in Parsing for Illegal UTF-8 Characters in Comments 24
The --thread_safe_initializations Option is Now Subsumed by -single_threaded 24
DoubleCheck Warning #15 25
Behavior Change of unsigned Bitfields in Certain Situations 25
Removed Features 27
Deprecated Features 28
Code Generation Defects 29

4. Changes in Version 2021.1 31

New Features and Enhancements 32
New Option to Annotate Call/Return Instructions 32
New Option to Mitigate Against Forward-Edge CFI Attacks 32
New Option for Disabling Misaligned Accesses 32
New Option to Customize Arena malloc 32
New Option to Include Line Numbers in Temporary Preprocessor Output 32
New Support for Splitting Sections Across Memory Blocks 33
New CPU Support 33
Improved Register Support 33
New Option for Improved Header Trace Diagnostics 33
New Predefined Macro for Secure Coding Standards 33

New Predefined Macro to Represent Toolchain Version 34
New Attribute to Cancel Generation of Non-Assembly Code 34
New Support for Cryptographic Extension Intrinsics for ARMv8 34
Changes in Behavior 34
Default align_val_t Overloads of operator new/delete Now Follow Proper Forwarding Order 34
Change in -globalreg Limitations 35
String Literals in Different Sections No Longer Merged Together 35
Options --section_prefix and --section_suffix No Longer Affect Special Sections 35
Deducing Types with Initializer List Constructors 35
Change in Defaults for Inlined C String Functions 36
Change in Treatment of memcpy() and memset() 36
Removed Features 36
Deprecated Features 36
Code Generation Defects 37

5. Changes in Version 2020.5 39

New Features and Enhancements 40
New DoubleCheck Features 40
New Linker Option to Prevent Function Deletion 40
New Documentation for Writing Secure Code 40
New Support for CERT C Rules 40
New Options to Mitigate Spectre Vulnerabilities 41
New Option to Instrument Function Prologues and Epilogues 41
New Support for Half-Precision Floating-Point Type 41
New gemfile Option to Improve Boot Loader Compatibility 41
New Constant Propagation Support 42
New CPU Support 42
New gstack Option to Set All Stack Usages at Once 43
New Support for Subnormal Results in expf() 43
Changes in Behavior 43
Stricter Type Checking for Strong Function Pointers 43
Placement of String Literal Data in Object Files 44
gstack Returns Non-Zero Status if User Attention is Required 45

Change in gbuild Evaluation of :reference Path 45
Deprecated Features 45
Code Generation Defects 45

6. Changes in Version 2020.1 47

Licensing 48
New Features and Enhancements 48
C11, C17, and C18 Support 48
New Support for Arithmetic Intrinsic 48
C++17 Support 49
Improved Deferral of Function Template Parsing 49
Changes in Behavior 49
Error for -fNaN_cmp_unordered with -fsoft or -fnone 49
ax D Modifier Change 49
Discretionary Errors Now Given in C++ Standard Library Headers 49
Linker Directives Files with Preprocessor Directives May Trigger Warnings 50
Removal of Non-Standard Members in Unordered Containers 50
Removed Features 51
Deprecated Features 51
Code Generation Defects 51

7. Changes in Version 2019.5 53

New Features and Enhancements 54
New Option for Trivial Function Inlining 54
New CPU Support 54
New Linker Option for Expanded Section Layout Information 54
New Ability to Customize the Value of FOPEN_MAX 55
New Option to Align Standalone Objects 55
New gsrec Option for Selective Data Output 55
New #pragma Directive to Influence Loop Unroll Factor 55
New Option to Control Number of Callers Tracked with Run-Time Memory Checking 55
Improved Support for Input Files in gasmlist 56

Changes in Behavior 56
Increased Number of Callers Tracked with Run-Time Memory Checking 56
Modified Returns for isnormal and isfinite 56
L'\x80' and L'\xff' No Longer Accepted as Wide Characters in Shift-JIS or EUC Modes 56
Removed Features 57
Deprecated Features 57
Code Generation Defects 57

8. Changes in Version 2019.1 59

New Features and Enhancements 60
Updated Compiler Front-End 60
New Option for Single-Threaded Performance 60
New Arena malloc 60
Improved Option to Generate a Target-Walkable Stack 60
Improved ARM64 Instruction Scheduling 60
Improved Wildcard Matching in Linker Directives Files 61
New Linker Option to Identify Separately Linked Absolute Images 61
New Linker Options to Provide Errors for Symbols Importable from Multiple Files 61
New Option to Assign Nameless Variables to Individual Data Sections 61
C++ Class Array Construction No Longer References new/malloc 62
New Attribute and Option to Enforce Section Order 62
Duplicate Strings Removed from DWARF Information 62
Improved Memory Functions for INTEGRITY Applications 63
Changes in Behavior 63
EDOM and ERANGE Not Defined in <math.h> 63
Changes to Alignment Reduction Behavior with __attribute__((aligned(x))) 63
Updated Aligned Parameter Register Calling Convention 64
When std::terminate() is Called, std::uncaught_exception() Now Returns "false" 64
Real-Time malloc is No Longer Experimental 64

New malloc Default 64
Scoped enum Without Explicit Base Type Always Given Base Type of int 64
Deprecated Features 65
Removed Features 65
Code Generation Defects 65

9. Changes in Version 2018.5 67

New Features and Enhancements 68
Support for MISRA 2012 68
New Option for Enhanced Loop Unrolling 68
Updated Support for C++11 and C++14 68
Improved gsrec Capabilities 68
New Option and Attribute to Perform Address Space Layout Randomization 69
Enhanced Line Information in Conditional and Logical Operators 69
New CPU Support 69
Changes in Behavior 70
Change in Accepted Multibyte Characters for Shift-JIS and EUC 70
Change in servecode Utility Support 70
Removed Features 70
Code Generation Defects 71

10. Changes in Version 2018.1 73

New Features and Enhancements 74
C++14 Support 74
New Version of malloc 74
New Option to Support Floating-Point/SIMD Register Use 74
New Predefined Symbols for Optimization Strategies 74
New Intrinsic to Mitigate Against Spectre Attacks 75
Improved Allocation of Data Into Program Sections 75
Changes in Behavior 75
INTEGRITY Small Block malloc Moved to libfmalloc 75
Updated Debugger Register View 75

Updated Support of Software Floating Point 76
Updated __MULTI_HOST__ Macro 76
New -delete Behavior 76
New Default Individual Sections Behavior 76
Deprecated Features 76
Removed Features 77
Code Generation Defects 77

11. Changes in Version 2017.5 79

New Features and Enhancements 80
64-Bit Toolchain Availability 80
Updated Compiler Front-End 80
Improved Implementation of C rand() Function 80
Improved Preprocessor Options 80
Improved Register Support 80
Increased Flexibility with Fused Multiply Accumulate Operations 81
Improved Assembler Support 81
Improved C++11 Support 81
New Option for Automatic NEON SIMD Vectorization 81
Improved Memory Function Speed 81
Improved #pragma ghs section 82
Enhanced Options for Retaining Temporary Preprocessor Files 82
Changes in Behavior 82
Updated alignof() Extension Behavior in C++11 82
Updated ABI for ARM64 82
Changes in Use of __asm__ Keyword 83
Change in Linker Directive Files 83
Temporary Demo Incompatibility 83
Deprecated Features 83
Removed Features 84
Code Generation Defects 84

12. Changes in Version 2017.1 85

New Features and Enhancements 86
Individual Function and Variable Sections 86

Option to Add Extra Dot to Individual Section Names 86
Enhanced Support for ARM64 Intrinsics 86
New Option for ax Librarian 86
CodeFactor Support 87
NEON Intrinsics Support 87
Improved INTEGRITY Support for --large_vtbl_offsets 87
Changes in Behavior 87
Modification of memmove Behavior 87
The Compiler Now Generates Template Information Files in Fewer Cases 88
Modification of Boolean Comparison Behavior 88
Change in Rebuilding Behavior of gbuild 88
New Option Name for -display_error_number 89
Change in Accepted Constraints with GNU asm 89
Removed Features 89
Code Generation Defects 90

13. Changes in Version 2016.5 91

New Features and Enhancements 92
C++11 Support 92
New Options Related to C++11 92
Support for Disabling Run-Time Type Information (RTTI) in C++11 Mode 93
Argument Type Checking 93
New Floating-Point Rigor Option 93
New Architecture Support for Trace and TimeMachine 93
New Option to Set Lower Bound of p_align 93
Added Support for CRC Instruction Set 93
Changes in Behavior 94
Default C Language Dialect is Now C99 94
Changes in fenv.h 94
Changes in abort() to Conform to C++ Standard 94
Changes in Standard C++ Behavior 94
C++ Binary Compatibility Across Compiler Versions 95
C++ Stream Locking 95
Treatment of Duplicate Inline Functions and Linkonce Template Instantiations 96

Stricter Diagnosis of Constructor Inaccessibility 96
Changes in Allowed Extensions 97
Modifying Libraries with ax Librarian 97
Deprecated Features 97
Removed Features 98
Code Generation Defects 98

14. Changes in Version 2015.1 99

New Features and Enhancements 100
Project Build Configurations 100
Automated Setup of Multicore Targets 100
Improved Configurability of the Tools for INTEGRITY 100
Using Interprocedural Optimizations with Makefiles 100
Duplicate Symbol Detection in Linker and Archiver 101
Preserving Temporary Preprocessor Files 101
Support for Stripping Debug Information From Relocatable Objects and Libraries 101
Section Sorting in Linker 102
Generating Both Numerically and Alphabetically Sorted Symbols in Map File 102
Annotating ELF Files with Toolchain Version Information 102
Link-time Global Variable Type Checking 102
64-Bit Linker and dblink on Windows and Linux Hosts 103
Specify the Output Name of a Dependency File 103
Additional Documentation of Diagnostic Messages for Assemblers and elxr 103
Ability to Specify Diagnostics by Symbolic Tag Names 103
Support for %= Format String in GNU Extended Assembly 104
Changes in Behavior 104
Diagnostic Messages When Building INTEGRITY 5.x and 10.x 104
bcmp and bcopy now treat their size arguments as unsigned 104
Removed and Deprecated Features 104

15. Changes in Version 2014.1 107

New Features and Enhancements 108
Cross-Project Implicit Dependency Analysis 108
New Undefined Behavior Checking 108
Range-Based For Loops 108
auto Type Keyword 108
Improved Inlining in C++ 109
Ability to Check for Use of Floating Point 109
Intermodule Variable Allocation 109
Memory Models 110
Inlining Constant Math Functions 110
Changes in Behavior 110
Changes in the Behavior of Some #pragma Directives 110
index() and rindex() Prototypes Now Hidden 110
Link-Once Template Instantiation Now Enabled by Default 111
Limited Support for C-Like Structure Packing in C++ 111
-gs No Longer Implies -gtws 112
sqrt() and sqrtf() Now Return NAN for Negative Input 112
Removed and Deprecated Features 112

16. Known Issues 113

Code Generation Defects 114
Installation and Licensing 114
Name of Install Directory Cannot Contain Spaces 114
Compiler and Probe Missing From ginstall 114
Empty Parent Directory Deletion During Uninstall on Windows Hosts 114
Host Support 115
UNC Paths Are Not Supported in the MULTI Debugger 115
UNC Paths to Virtual Machines Are Not Supported 115
Toolchain Issues 115
INTEGRITY Fails to Build 115
gstack Provides Warnings for Green Hills Library Functions 115
gstack Provides Warnings for INTEGRITY 116

Use of -a, -p, or -pg in INTEGRITY KernelSpace Requires -Onomemfuncs 116
Common Data Syntax 116
Building Projects with Absolute Path Output Locations 116
Building a Project with Multiple Instances of a File 117
Error Associated with Calling sqrt 117
Toolchain Components Failing to Launch 117
Increased Compile Time and Host Memory Usage with Complex Templates 117
Toolchain File Size Limitation 118
Target Connection Issues 118
Flash Driver Library libflash.a Limited to Four CFI Drivers 118
Debugging Issues 118
Target Agent May Cause Failures When Programming Flash 118
References Not Available for Inlined Functions 118
Cannot Debug Executables in Eclipse Projects with a Space in the Name 119
TimeMachine Does Not Track Floating-Point/SIMD Registers . . . 119
C11 Debugging Issues 119
C++11 Debugging Issues 119
Memory Leak Detection May Result in False Positives/Negatives 120
NEON Vectors Incorrectly Displayed in MULTI 120
Stack Trace Causes Incorrect Disassembly 120
File System Issues 120
Two Dots (..) Not Supported After Symbolic Links in Paths 120
NFS May Cause Poor Performance 121
Miscellaneous 121
Links to Connecting Book 121

Chapter 1

Changes in Version 2023.5

Contents

System Requirements 2
Product Compatibility 4
Feature Status Definitions 5
New Features and Enhancements 5
Changes in Behavior 8
Removed Features 9
Deprecated Features 9
Code Generation Defects 9

System Requirements

This section details the system requirements for the current compiler release.

The MULTI IDE may have different system requirements than the Compiler. For information about IDE requirements, see the release notes for the MULTI IDE.

Note that if you intend to build INTEGRITY, please contact Green Hills Support.

Windows

Running a full installation on Windows requires:

- A modern, multi-core, 64-bit x86 processor.
- One of the following:
  - Windows 8 (64-bit).
  - Windows 10 (64-bit).
  - Windows 11 (64-bit).
- 4 GB of RAM (16 GB is recommended).
- 10 GB of free disk space per installation (100 GB of free space and a solid state drive are recommended).
- A monitor running at a display resolution of 1024x768 or higher. (Two high-resolution monitors are recommended.)
- A DVD-ROM drive or access to the Green Hills Support site [https://support.ghs.com]. If neither is available, contact Green Hills Support.
- The ability to write to the Windows System directories. On most Windows systems, you must also be a member of the Administrators group. End users must have full access to installed files.

Linux

Running a full installation on Linux requires:

- A modern, multi-core, 64-bit x86 processor.

- One of the following:
  - Ubuntu 22.04.x LTS (64-bit).
  - Ubuntu 20.04.x LTS (64-bit).
  - Ubuntu 18.04.x LTS (64-bit).
  - Ubuntu 16.04.x LTS (64-bit).
  - Ubuntu 14.04.x LTS (64-bit).
    - Note: Support for Ubuntu 14.04.x is deprecated and may be removed in a future release.

Note

Ubuntu is distributed with "HWE" kernels that are updated frequently and are not supported by Green Hills Software. If you encounter a toolchain issue while using an HWE kernel, we strongly advise trying a non-HWE kernel prior to contacting Green Hills Support.

  - CentOS 7.x (64-bit).
  - CentOS 6.x (64-bit).
    - Note: Support for CentOS 6.x is deprecated and may be removed in a future release.
- 32-bit compatibility libraries are required. If you are using Ubuntu 14 or newer, run:

```
sudo dpkg --add-architecture i386
sudo apt-get update
sudo apt-get install libc6:i386  libncurses5:i386
sudo apt-get install libstdc++6:i386 libx11-6:i386 lib32z1
sudo apt-get install linux-libc-dev:i386
sudo apt-get install libxcursor1:i386
sudo apt-get install libc6-dev-i386
```

If you are using CentOS, run:

```
sudo yum install glibc.i686 glibc-devel.i686
sudo yum install libX11.i686 zlib.i686 libstdc++.i686
```

If you find that the tools are not working, especially because of a licensing issue, you may need to install additional 32-bit compatibility libraries. For

example, if you are using LDAP with SSS, you must install **libnss_sss**, which is usually part of the sssd-client package.

- The GNU C library (**glibc**) version 2.11 - 2.35.
  - Note: Support for **glibc** versions 2.11 - 2.16 is deprecated and may be removed in a future release.
- 4 GB of RAM (16 GB is recommended).
- 10 GB of free disk space per installation (100 GB of free space and a solid state drive are recommended).
- A monitor running at a resolution of 1024x768 or higher. (Two high-resolution monitors are recommended.)
- A graphical display running the X Window System.
- A mounted DVD-ROM drive or access to the Green Hills Support site [https://support.ghs.com]. If neither is available, contact Green Hills Support.
- An Ethernet device.
- Write permissions to the installation directory.

Product Compatibility

Green Hills Compiler 2023.5 is officially supported with:

- MULTI version 8.x.
- INTEGRITY version 11.0.4, 11.2.4, 11.4.4, 11.4.6, 11.7.8, 11.7.9, and 19.1.
- INTEGRITY IoT version 2017.0.
- u-velOSity version 2.6.4, 2.7.1, and 2.7.2.
- Green Hills Debug Probe software version 6.6.4.

For information about compatibility with newer versions of INTEGRITY, see the INTEGRITY release notes.

Note

If your MULTI IDE is newer than the compiler, see the IDE release notes for up-to-date product compatibility.

Feature Status Definitions

Toolchain features may have differing levels of support:

- **Supported** — Unless otherwise stated, toolchain features are considered supported by Green Hills.
- **Unsupported** — This feature exists in the toolchain but is not considered supported. It is offered as-is.
- **Deprecated** — This feature is currently supported but will be unsupported or removed in a future release.
- **Removed** — This feature is no longer available for the associated toolchain release and all subsequent releases unless otherwise stated.

New Features and Enhancements

New Option to Set Default Buffer Size for stdio Libraries

A new option (**-stdio_bufsiz=*size***) allows you to set the default size of buffers used by the `stdio` libraries. For more information, see "Default buffer size for stdio libraries" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Annex K Functions Available in All Language Dialects

The bounds-checking functions described in C11 Annex K are now provided in all dialects of C and C++. (Previously they were only available in C11, C++11, and later dialects.)

Changes to Linker -C and -D Options

The **-C** and **-D** linker options now support expanded expressions. In addition to immediate values as before, these options will now accept arithmetic expressions consisting of immediates and/or linker constants.

New Options to Treat Warnings as Errors

A new option (**--quit_after_build_warnings**) allows you to treat warnings as errors in many toolchain components, similar to the existing driver option **--quit_after_warnings**. Individual options have also been added to control this functionality for each program separately. To clarify its scope, **--quit_after_warnings** has been renamed to **--quit_after_compiler_warnings**. For more information, see "Quit Building if Warnings are Generated" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5* and "Quit Building if Compiler Warnings are Generated" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Option to Control Whether dblink Errors Result in a Build Failure

A new option (**--fail_build_on_dblink_error**) allows you to specify that errors generated by **dblink** should result in a build failure. For more information, see "Fail Build if dblink Errors are Generated" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Improved Debugging for C11, C++11, C++14, and C++17

When used with Compiler 2023.5, the MULTI 8.x Debugger supports a number of new features, including:

- C++11 lambda functions
- C++14 variable templates
- Variables with thread-local storage in u-velOSity and INTEGRITY user space
- `char16_t`, `char32_t`, and `wchar_t` base types

The following options are not required and should not be used when debugging with MULTI 8.x:

- **--dbo_version=**
- **-dblink.args=-dloVersion=**
- **-dblink.args=-dnmVersion=**

Note: The debugger cannot debug `thread_local` variables in kernel space. If you attempt to display them in the Debugger, it will falsely report that they are optimized away.

New Option to Allow Flexible Array Members in All Dialects of C and C++

A new option (**--flexible_array_members**) allows you to use C99 conformant flexible array members in all dialects of C and C++. For more information, see "Allow Flexible Array Members in All Dialects" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

MIRROR Attribute Added to Memory Map Definitions

The `MIRROR` attribute has been added to memory map definitions, which can be used to represent overlapping regions of memory. For more information, see "Defining a Memory Map with the MEMORY Directive" in Chapter 8, "The elxr Linker" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Newly Supported ARM64 CPUs

The following CPUs are now supported:

- Cortex-A34 (**-cpu=cortexa34**)
- Cortex-A65 (**-cpu=cortexa65**)
- Cortex-A65AE (**-cpu=cortexa65ae**)
- Cortex-A715 (**-cpu=cortexa715**)
- Cortex-X1C (**-cpu=cortexx1c**)
- Cortex-X3 (**-cpu=cortexx3**)
- Neoverse-V2 (**-cpu=neoversev2**)

Changes in Behavior

Map Files Exclude Non-Allocated Sections

By default when the linker emits a map file, it will now exclude unallocated sections (sections without the `SHF_ALLOC` flag set). Their addresses do not correspond to target memory and so they may appear as if they overlap allocated sections in memory. To restore the old behavior of including these sections in the map file, pass the **-Mc** option to the driver.

Libraries Reject Zero as Second Byte of Multibyte Character in EUC and Shift-JIS

When the target character encoding is set to EUC or Shift-JIS (option **-kanji=euc**, **-kanji=shiftjis**, or **-kanji=euc/shiftjis**), the libraries no longer allow multibyte characters in which the second byte is zero. For example, if `mbtowc` is called on a string consisting of the bytes `0x81` and `0x00`, it will return `-1` (indicating invalid input); previously it would have returned `2` and produced the wide character `0x8100`. Likewise, the function `wctomb` rejects wide characters in which the low byte is 0 (except the null wide character `L'\0'`). Note that no legal non-null character in EUC or Shift-JIS ends with a zero byte; the libraries used for EUC and Shift-JIS are more permissive than either character encoding. This change is to better conform with the standard, which forbids a non-null multibyte character from containing a zero byte.

Increased Instrumentation with -check=memory,nilderef

When both **-check=memory** and **-check=nilderef** are enabled, the compiler now adds both types of instrumentation to the program. Previously, **-check=nilderef** was ignored in this case. This results in detecting more types of NULL dereferences than before, at the cost of increased executable size. Additionally, some NULL dereferences will now be detected by both types of instrumentation, resulting in two run-time errors being reported. The old behavior can be restored by disabling **-check=nilderef** wherever **-check=memory** is enabled, using either **-check=nonilderef** or **#pragma ghs check=nonilderef**. For more information, see "Run-Time Memory Checks" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Packing no longer allowed on non-POD classes

Packing is no longer allowed on non-POD structures and classes, even with **-misalign_access**. If you attempt to pack a non-POD, the request will be ignored with the following warning:

```
warning #2020-D: packing ignored on non-POD type
```

Note that since previous releases allowed packing of non-PODs under **-misalign_access**, this change may affect the layout of non-PODs for which packing is requested. For more information, see "Limitations of Structure Packing in C++" in Chapter 2, "Developing for ARM64" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Removed Features

The following features are no longer available:

- The function `gamma()`
- The **gproxy** utility

Deprecated Features

The following features are deprecated and may be removed in a future release:

- The option **--quit_after_warnings**. For the same behavior, use **--quit_after_compiler_warnings**.
- The option **-memory_access**.

Code Generation Defects

For a list of known code generation defects, see "Code Generation Defects" on page 114.

Chapter 2

Changes in Version 2023.1

Contents

New Features and Enhancements 12
Changes in Behavior 13
Removed Features 15
Deprecated Features 15
Code Generation Defects 15

New Features and Enhancements

New Supported Host: Windows 11

Windows 11 is now a supported host for the compiler.

New DoubleCheck Assert Fail Intrinsic

A new `__double_check_assert_fail()` intrinsic is now supported. For more information, see "Intrinsic Functions" in Chapter 14, "Macros, Pragma Directives, and Intrinsics" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Improved DWARF Information with -delete

Using the **-delete** option no longer causes the generated DWARF debugging information to be out of sync with the program in memory.

gaddr2line DWARF Support

The **gaddr2line** utility program can now use either the Green Hills **.dbo** information or DWARF information to correlate instruction addresses with source positions.

Two new options can be used to control this behavior: if **--dbo** is used, **gaddr2line** will only attempt to use the **.dbo** information. If **--dwarf** is used, it will only consider the program's DWARF information. If neither of these options is used, **gaddr2line** will only use the DWARF information if the **.dbo** information is missing.

See "The gaddr2line Utility Program" in Chapter 10, "Utility Programs" in *MULTI: Building Applications for Embedded ARM64 v2023.5* for more information.

New gbincmp Options

The **gbincmp** utility now supports three new options: **-[no]empty**, **-[no_]empty_not_found**, and **-[no]prefix**. For more information, see "The gbincmp Utility Program" in Chapter 10, "Utility Programs" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

libc++ Now Provided as a Standard C++ Library Alternative

The Green Hills Compiler now offers the choice between two different implementations of the standard C++ libraries.

In addition to the existing Dinkumware standard C++ libraries, a Green Hills specific version of the libc++ libraries is now also provided and may be selected with the **--libcxx** option. Both libraries are intended to conform to the C++ standard and the choice between the two is primarily personal preference, although the choice must be consistent throughout your application.

There are some restrictions to the usage of the libc++ libraries, the most notable of which is that they may only be used with C++11 or higher. For more information, see "Specifying C++ Libraries" in Chapter 12, "Green Hills C++" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Attribute set_offset

The attribute `set_offset` can now be added to struct members, allowing a custom offset to be specified without the need for padding. This may be particularly useful when reading only a few fields in a file header or network packet, or when dealing with sparse device registers. Note that this attribute cannot reorder fields or cause them to overlap.

Changes in Behavior

New Version of ind_io_arrays.c

The file **ind_io_arrays.c**, used to customize the value of `FOPEN_MAX`, has been updated. If you have an existing project that includes a copy of that file, you will need to replace it with the new version. Failure to do so will result in an error similar to the following:

```
[elxr] (error #412) unresolved symbols: 1
 __ghs_alternate_file_arrays_v2         from myprogram.o
```

For more information, see “Customizing the Value of FOPEN_MAX” in Chapter 15, “Libraries and Header Files” in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Change in Linkage for Const-Qualified Variable Templates (C++ CWG Defect 2387)

The compiler now implements the resolution to C++ Core Working Group issue 2387, which changes the default linkage of variable templates that are const-qualified.

Previously, an instantiation of a variable template would have internal linkage if the variable had a non-volatile const-qualified type and did not have an explicit 'extern' keyword on the declaration. It did not matter whether the 'const' keyword was specified explicitly in the declaration or whether the 'const' type was specified via a template argument.

Now, however, a variable template will never implicitly have internal linkage because of a const-qualified type (although it may have internal linkage for other reasons, such as being declared in an unnamed namespace).

```
template <typename T>
T const templ_var1 = 0;

template <typename T>
T templ_var2 = 0;

/* As part of the example, take the addresses
   of the variable templates to force their
   implicit instantiation. */

/* templ_var1<int> used to have internal (i.e.
   file) linkage, now has external linkage */
int const *ptr1 = &templ_var1<int>;

/* templ_var2<const int> used to have internal
   linkage, now has external linkage */
int const *ptr2 = &templ_var2<const int>;

/* templ_var2<int> used to have external linkage
```

```
   (no change). */
int const *ptr3 = &templ_var2<int>;
```

BUFSIZ increased to 8192 on INTEGRITY

The default buffer size used by the **stdio** library, `BUFSIZ`, has been increased to 8192 on INTEGRITY (from its old value of 512). This is to improve performance when doing large reads or writes, by reducing the number of input/output system calls required. Note that the **stdio** library allocates its buffers on the heap, so this will result in increased heap usage. Additionally, Tasks that allocate buffers of size `BUFSIZ` on the stack may need to have their `StackLength` attribute increased. Note also that the function `setbuf` sets a stream buffer which is implicitly of size `BUFSIZ`. Calls to that function which pass a buffer of hard-coded size may need to be changed to use `setvbuf`, which allows specifying the buffer size explicitly.

Removed Features

The following features are no longer available:

- INTEGRITY 5.x and 10.0.2 are no longer supported.

Deprecated Features

Code Generation Defects

For a list of known code generation defects, see “Code Generation Defects” on page 114.

Chapter 3

Changes in Version 2022.1

Contents

New Features and Enhancements 18
Changes in Behavior 21
Removed Features 27
Deprecated Features 28
Code Generation Defects 29

New Features and Enhancements

New Option to Constrain Functions

A new option (**-constraint_rules_file**) allows function use to be restricted, such as for secure coding purposes. For more information, see "Constraint Rules" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New gstack Options

The **gstack** utility now supports three new options, all of which alter the information provided in **gstack** reports: **-list_funcs**, **--link_names_only**, and **--show_disamb**. For more information, see "gstack Options" in Chapter 10, "Utility Programs" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Option for Floating-Point Divide Conversions

A new option (**-const_fdiv_to_fmul**) allows the compiler to convert a floating-point divide by constant into an equivalent multiply, and is implied by **-frigor=fast**. For more information, see "Divide By Constant Becomes Multiply By Reciprocal" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Optimization Attribute and #pragma

A new attribute (`__attribute__((ghs_optimize(O`*`strategy`*`)))`) and `#pragma` (`#pragma ghs O`*`strategy`*) allow configuring the high-level optimization strategy for individual functions. These should be used instead of the old `#pragma ghs OL`, `#pragma ghs OS`, and `#pragma ghs ZO` (which are now deprecated). The new attribute and `#pragma` should more closely match the corresponding command line options. For more information, see "Green Hills Extension Pragma Directives" in Chapter 14, "Macros, Pragma Directives, and Intrinsics" in *MULTI: Building Applications for Embedded ARM64 v2023.5* and "Attributes" in Chapter 11, "Green Hills C" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Pragma and Attribute Syntax for Confirming Section Placement of Variables

The `#pragma ghs section` directive and `__attribute__((section(...)))` notations now accept an additional syntax that causes the compiler to issue an error if an affected variable would be placed into a different type of section than the type specified by the `#pragma` directive or attribute.

This behavior can be enabled using the `require` and `endrequire` argument in `#pragma ghs section`, or using the two-argument form of `__attribute__((section(...)))`. For more information, see "Green Hills Extension Pragma Directives" in Chapter 14, "Macros, Pragma Directives, and Intrinsics" in *MULTI: Building Applications for Embedded ARM64 v2023.5* and "Attributes" in Chapter 11, "Green Hills C" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Default Optimization Levels

The options **-O1**, **-O2**, **-O3**, **-Os**, **-Oz**, **-Og**, and **-Od** have been added to provide the ability to select a default program characteristic to optimize for. For more information, see "Default Optimization Level" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New CPU Support

The following CPUs are now supported:

- Cortex-A78AE (**-cpu=cortexa78ae**)
- Cortex-A78C (**-cpu=cortexa78c**)
- Cortex-A510 (**-cpu=cortexa510**)
- Cortex-A710 (**-cpu=cortexa710**)
- Cortex-X2 (**-cpu=cortexx2**)
- Kryo560-Gold (**-cpu=kryo560g**)
- Kryo560-Silver (**-cpu=kryo560s**)
- Kryo570-Gold (**-cpu=kryo570g**)
- Kryo570-Silver (**-cpu=kryo570s**)

- Kryo670-Gold (**-cpu=kryo670g**)
- Kryo670-Silver (**-cpu=kryo670s**)
- Kryo670-Prime (**-cpu=kryo670p**)
- Kryo680-Gold (**-cpu=kryo680g**)
- Kryo680-Silver (**-cpu=kryo680s**)
- Kryo680-Prime (**-cpu=kryo680p**)
- Neoverse-E1 (**-cpu=neoversee1**)
- Neoverse-N1 (**-cpu=neoversen1**)
- Neoverse-N2 (**-cpu=neoversen2**)
- Neoverse-V1 (**-cpu=neoversev1**)

Preventing Literal Data in Text

The option **-no_data_in_text** is now supported on ARM64 processors. This option can prevent generating literal data in executable sections by the toolchain. For more information, see "Placement of Literal Data in Text" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

FPSR Status Bits Now Supported on simarm64

Floating-point instructions that trigger any bits of `FPSR.{IOC,DZC,OFC,UFC,IX,QC}` on hardware will now reflect this behavior in **simarm64**. These bits can be cleared by writing `0` to them using the `MSR` instruction. The traps for these bits are still unsupported. For details, see the ARM processor manual.

New DoubleCheck Warning

DoubleCheck warning #22 is now available to detect dead writes to function parameters. For more information, see "Source Analysis Error and Warning Messages" in Chapter 5, "The DoubleCheck Source Analysis Tool" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Option to Remap Paths in DWARF Information

A new option (**-remap_dwarf_path**) allow the tools to alter paths in the generated DWARF debugging information, which makes it easier to generate identical ELF files in separate builds. For more information, see "Remap Path in DWARF Information" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Support for Splitting Relocatable Objects Across Memory Blocks

When a section is designated in the link map to be split across multiple memory blocks, it may now split up different instances of that section coming from the same relocatable object.

New POSIX Wide Character Conversion Functions

The library functions `mbsnrtowcs` and `wcsnrtombs`, from POSIX, are now supported. These functions convert between wide and multibyte strings. For more information, see the POSIX standard and "Standard C Functions and Dependencies" in Chapter 15, "Libraries and Header Files" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Improved Homogeneous Floating Point Aggregate Code Quality

Composite types consisting of one to four elements of a single floating point type (such as complex numbers or vectors) may now be allocated to registers within a function, and not merely passed that way as parameters. This results in reduced stack usage and code size when using these types.

Changes in Behavior

Change to Defaults Shown for Optimization Options in Builder

Most options in the **Advanced → Optimization Options** category now always show a default value of **Allowed** in the Builder, regardless of optimization strategy. Previously, the default was dependent on the optimization strategy. The new behavior

is required because of how these options interact with the new optimization attribute and `#pragma` (see "New Optimization Attribute and #pragma" on page 18).

As an example, **Pipeline Instruction Scheduling** is performed under **-Ospeed**, but not **-Onone**. In previous releases, passing **-Onone** on the command line prevented pipelining for the entire file. In contrast, it is now possible for pipelining to be performed within such a file – for example, if it contains a function declared with `#pragma ghs Ospeed`. However, if **-Onopipeline** is also passed, then pipelining will be suppressed for the entire file, regardless of optimization attributes or `#pragma` directives. The old behavior could be implemented by having **-Onone** imply **-Onopipeline**, but the new behavior cannot, because **-Onone -Onopipeline** now has a different effect than **-Onone** by itself.

The names for these option settings have been changed to clarify their behavior: **Allowed** (instead of **On**) to indicate that the compiler may perform the optimization, and **Disabled** (instead of **Off**) to indicate that the compiler may not perform the optimization anywhere in the file. The underlying driver options are the same, and any existing projects that pass these options will still pass the same options after upgrading. For more information, see "Optimization Options" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Decreased Ability to Toggle Loop Optimizations

It is no longer possible to toggle all loop optimizations as a unit. The compiler will choose appropriate loop optimizations to perform according to the optimization strategy. Individual loop optimizations which had separate options (such as loop unrolling) can still be manually disabled using those options, but cannot be manually enabled if not included in the optimization strategy. For example, it is no longer possible to manually enable loop unrolling under **-Ogeneral**. (Previously, this would have required passing **-OL** to enable loop optimizations.) To gain the speed benefits of loop optimizations without the size costs of applying them globally, we recommend selectively applying speed optimizations to those functions that make up an appreciable fraction of execution time. This can be done using the `ghs_optimize` attribute, `#pragma ghs Ospeed`, or the option **--fast=**. See "Optimize All Appropriate Loops" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*, "New Optimization Attribute and #pragma" on page 18 and "Optimize Specific Functions for Speed"

in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Optimization Strategies Selected for -O1, -O2, and -O3

The options **-O1**, **-O2**, and **-O3** have been updated to better suit desired performance characteristics. For more information, see "Default Optimization Level" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Using -Olimit= Without Debugging Information No Longer Requires -glimits

The options **-Olimit=peephole** and **-Olimit=pipeline** no longer require passing `-glimits` in order to have an effect when not generating debug information. The option **-glimits** is still recognized for backward compatibility, but it has no effect. For more information, see "-Olimit= Without Debug Information" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Parallel Execution Policies No Longer Defined in std::execution

The `std::execution` namespace no longer defines the `parallel_policy` or `parallel_unsequenced_policy` types, nor the `par` and `par_unseq` variables. The goal of this change is to make it clearer that the parallel behaviors denoted by these entities are not supported. If your application requires the existence of these entities to build (but does not require parallel behavior), you can define them again by defining the macro `__GHS_KEEP_UNSUPPORTED_PARALLEL_ALGORITHM_DEFS` prior to the inclusion of `<execution>`.

Options --fast=all and --small=all Now Apply to Only One Function

When the string `all` is passed to the **--fast** or **--small** option, it now indicates to the compiler that the function named `all()` should be optimized in the indicated manner. Previously, it would result in every function being optimized in that manner. For more information, see "Optimize Specific Functions for Speed" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5* and "Optimize Specific Functions for Size" in Chapter 3, "Builder

and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Change in Value for std::numeric_limits<floating-point-types>::has_signaling_NaN

The value of `std::numeric_limits<`*T*`>::has_signaling_NaN` is now `false` when *T* is a floating point type (such as `float` or `double`). Previously, this value was `true`. This prior value reflected the fact that signaling NaNs had their own representation and could be differentiated from quiet NaNs. However, with the exception of C++ functions dedicated to signaling NaNs, neither the C nor C++ libraries would specifically detect or create signaling NaNs. When a signaling NaN was passed as input to a library function it would be treated the same as a quiet NaN. Additionally, most hardware will silently ignore signaling NaNs or otherwise convert them to quiet NaNs when in default configurations. For these reasons, the value of `std::numeric_limits<`*T*`>::has_signaling_NaN` has been changed to `false` to more accurately reflect current behavior.

Change in -MMD Option

The option **-MMD** now omits the standard C headers in addition to the standard C++ headers and headers from system `#include` directories.

Change in Parsing for Illegal UTF-8 Characters in Comments

When the input source encoding is set to `UTF-8` (e.g. **-kanji=utf8**), the compiler will now give an error if an illegal `UTF-8` character is present in the comments of a source file. The error given is: `error #7: unrecognized token`.

The --thread_safe_initializations Option is Now Subsumed by -single_threaded

The **Thread Safe Initializations** option has been subsumed by the **Program is Single Threaded** option. The behavior previously enabled by **--no_thread_safe_initializations** is now enabled by **-single_threaded**, while **-no_single_threaded** corresponds to **--thread_safe_initializations**. This is true regardless of the C++ dialect selected.

As a consequence of this change, the **--thread_safe_initializations** behavior is now the default even in C++03 (assuming **-no_single_threaded** is in effect). As before, **--thread_safe_initializations** will not enforce thread safety unless the `_Mtx_*` functions are defined. For more information, see "Program is Single-Threaded" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

DoubleCheck Warning #15

DoubleCheck warning #15 now detects dead writes to variables that are not function parameters, instead of all variables. Additionally, it is now off by default but may be re-enabled with **-double_check.enable=15**. This change is in tandem with the new DoubleCheck warning number #22 described in the New Features above. If your project previously used an ignore option or `#pragma` for #15, you may need to remove the ignore, replace #15 with #22, or ignore #22 as well, depending on your project's specifics.

Behavior Change of unsigned Bitfields in Certain Situations

The behavior of unsigned bitfields whose values are used in certain situations has changed. These situations only arise when:

- Both of the following are true:
  - The bitfield has an unsigned base type whose size is at least the size of an `int`.
  - The width is less than the number of bits in an `int`.
- And when the value of the bitfield is read in one of the following contexts:
  - The first operand of an assignment or compound assignment operator.
  - The operand of unary prefix or unary postfix increment or decrement.
  - The second operand of a comma operator.

Previously, the type of the value returned by the operator would have the unsigned base type of the field. Now, the type of the value will be a signed integer type (the result of integral promotion), which is consistent with how the bitfield's value is treated in ordinary read contexts. An exception is that the postfix increment and

decrement cases in C++ continue to have the unsigned base type of the field. The new behavior is in accordance with the intent of the language standards.

This may cause a change of behavior in cases such as the following:

```
struct foo {
     unsigned int field: 4;
 };
 bool func(void)
 {
     int negative = -1;
     struct foo bar;
     bar.field = 0;
     // Previously, the += operator's value would be taken to have
     // an unsigned type, so the usual arithmetic conversions would
     // convert negative to UINT_MAX, and the comparison
     // 0 <= UINT_MAX would return true.
     // Now, the += operator's value will be taken to be signed, and
     // the comparison 0 <= -1 will return false.
     return (bar.field += 0) <= negative;

     // Note: If the code had been written:
     // bar.field += 0;
     // return bar.field < negative;
     // The code would have returned false both before and after
     // this change.
 }
```

Note that the behavior of assignment to a bitfield itself has not changed – what has changed is the observed type in these contexts (that is, the type of the assignment expression when it is used as part of a larger expression).

Furthermore, integral promotion on unsigned bitfields that have a width larger than the size of an `int` but less than the size of a `long long` has changed. Previously, these fields were given a type of `signed long long`. These types now have a type of `unsigned long long`. This behavior is also in accordance with the intent of the language standards.

Removed Features

The following features are no longer available:

- The `#pragma ghs OM`.
- The macro `__Char_Is_Signed__`.
- The macro `__Char_Is_Unsigned__`.
- The macro `__GHS_Inline_Memory_Functions`.
- The macro `__Int_Is_32`.
- The macro `__Int_Is_64`.
- The macro `__LL_BIT`.
- The macro `__LL_Is_64`.
- The macro `__Long_Is_32`.
- The macro `__Long_Is_64`.
- The macro `__MISRA_`*i*, where *i* is a MISRA 1998 rule.
- The macro `__ONLY_STANDARD_KEYWORDS_IN_C`.
- The macro `__Ptr_Is_32`.
- The macro `__Ptr_Is_64`.
- The macro `__Reg_Is_32`.
- The macro `__Reg_Is_64`.
- The macro `__WChar_Is_Int__`.
- The macro `__WChar_Is_Long__`.
- The macro `__WChar_Is_Short__`.
- The macro `__msw`.
- The option **--bool**.
- The option **--legacy** for the **gstack** utility.
- The option **--saferc**. Note that this means that MISRA 1998 rules can no longer be enforced.
- The option **--unique_strings**.
- The option **-C** (**HTML File Compression**).
- The option **-OM**.

- The option **-Oconstprop**.
- The option **-Omemfuncs**.
- The option **-dyload**.
- The option **-funroll-loops**
- The option **-fvolatile**.
- The option **-h** (**Alternate Header Name**).
- The option **-irel**.
- The option **-libbsp**.
- The option **-o** (**Alternate Source Name**).
- The option **-p** (**Alternate Compression Table**).
- The option **-pic_compat**.
- The option **-tekhex** for the **gsrec** utility.
- The preprocessing directive `#assert`.
- The preprocessing directive `#import`.
- The preprocessing directive `#include_next`.
- The use of conditional `#pragma` directives (such as `ifnooptimize`).
- The following library functions: `mktemp()`, `bcmp()`, `bcopy()`, `bzero()`, `eprintf()`, `getw()`, `truncate()`, `putw()`, `rand_r2()`, `setlinebuf()`.
- The following non-standard C++ headers: **fstream.h**, **iomanip.h**, **iostream.h**, **new.h**, **rope**, **stdiostream.h**, **stl.h**, **strstream.h**.
- Specifying a C++ mangled name for *foo* in `#pragma weak foo`.

Deprecated Features

The following features have been deprecated and will be removed in a future release:

- The `#pragma ghs OL`, `#pragma ghs OS`, and `#pragma ghs ZO`. Use the new `#pragma ghs Ospeed`, `#pragma ghs Osize`, and `#pragma ghs Onone` (respectively) instead. See “New Optimization Attribute and #pragma” on page 18.
- The function `gamma()`.

- The option **-glimits**.
- The option **-OL**.
- The option **-OL=*function***.
- The option **-Oloop**.
- Assembly comments beginning with a number character (#).

Code Generation Defects

For a list of known code generation defects, see “Code Generation Defects” on page 114.

Chapter 4

Changes in Version 2021.1

Contents

New Features and Enhancements 32
Changes in Behavior 34
Removed Features 36
Deprecated Features 36
Code Generation Defects 37

New Features and Enhancements

New Option to Annotate Call/Return Instructions

A new option (**-annotate_call_return**) allows you annotate call/return instructions for improved debugging. For more information, see "Annotate Call/Return Instructions" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Option to Mitigate Against Forward-Edge CFI Attacks

A new option (**-indirect_function_call_checks**) allows you to mitigate against forward-edge CFI attacks using indirect function-call checks. For more information, see "Indirect Function Call Checks" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Option for Disabling Misaligned Accesses

A new option (**-no_misalign_access**) allows you to disable misaligned accesses when running on configurations that lack support for such behavior. This may be relevant when working with certain memory-mapped I/O or when running code without the MMU enabled. For more information, see "Misaligned Memory Access" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Option to Customize Arena malloc

A new option, **-malloc_num_arenas**, allows for overriding the number of arenas used by the arena `malloc` on supported operating systems. For more information, see "Number of Arenas for Malloc" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Option to Include Line Numbers in Temporary Preprocessor Output

A new option (**-keepcppfileswithlines**) causes `#line` directives to be emitted in the preprocessed output files produced by **-keepcppfiles**. For more information,

see "Line Numbers in Temporary Preprocessor Output" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Support for Splitting Sections Across Memory Blocks

The linker now supports the ability to split sections across multiple memory blocks using a section map. For more information, see "Splitting a Section Across Multiple Memory Blocks" in Chapter 8, "The elxr Linker" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New CPU Support

The Cortex-R82 (`-cpu=cortexr82`) is now supported.

Improved Register Support

New named ARMv8.5 registers related to `RNDR`, `RNDRRS`, and memory tagging extensions are now supported. For more information, see the table row pertaining to `__MRS()` in "System Intrinsics" in Chapter 14, "Macros, Pragma Directives, and Intrinsics" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Option for Improved Header Trace Diagnostics

A new option (**--diagnostic_header_trace**) allows diagnostic messages emitted from header files to be preceded by a trace of the directives to `#include` said header files. For more information, see "Show Header Trace for Diagnostics" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Predefined Macro for Secure Coding Standards

A new predefined macro (`GHS_STANDARDS`) is available to aid compliance with secure coding standards. For more information, see "Green Hills Secure Coding Options Files" in Appendix D, "Secure Coding Practices" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Predefined Macro to Represent Toolchain Version

A new predefined macro (`GHS_TOOLS_VERSION`) is available to represent the toolchain version in use. For more information, see "Predefined Macros" in Appendix F, "Customization Files" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Attribute to Cancel Generation of Non-Assembly Code

The new attribute `naked` is now supported, which can be used to tell the compiler to not generate a prologue, epilogue, or other non-`asm` code for a given function. This can be used to hand-write the body of a function in assembly, but define the function in a C module. For more information, see "Attributes" in Chapter 11, "Green Hills C" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Support for Cryptographic Extension Intrinsics for ARMv8

The compiler now supports the AArch64 Cryptographic Extension instructions intrinsics. For more information, see "Cryptographic Extension Intrinsics" in Chapter 14, "Macros, Pragma Directives, and Intrinsics" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Changes in Behavior

Default align_val_t Overloads of operator new/delete Now Follow Proper Forwarding Order

The default versions of `operator new/delete` (and their array versions) now follow proper forwarding when using an `align_val_t` parameter. For example, the default version of `operator new[](size_t, align_val_t)` will now forward to `operator new(size_t, align_val_t)`, as specified by the C++17 standard.

For backwards compatibility, the behavior of overloads without an `align_val_t` parameter is unchanged and still follows the C++03 specification (pre-LWG 206). This may change in future releases. Please keep this in mind if you replace these

global operators with your own versions, as it may affect which overloads you must replace for correct behavior.

Change in -globalreg Limitations

The **-globalreg** option can no longer be set higher than `3` when building a C++ project. For more information, see "Global Registers" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

String Literals in Different Sections No Longer Merged Together

Previously, the compiler could merge together duplicate string literals even if they were allocated in different sections (for example, due to `#pragma ghs section` directives). Now the compiler will keep duplicate strings separate if they have any difference in their section mapping. This includes cases where the difference in section mapping has no effect on the section targeted for string allocation.

Options --section_prefix and --section_suffix No Longer Affect Special Sections

The names of special sections, such as those used to record profiling data with **-coverage**, are no longer affected by **--section_prefix** and **--section_suffix**. For more information about these options, see "Prepend String to Every Section Name" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5* and "Append String to Every Section Name" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*. options.

Deducing Types with Initializer List Constructors

When performing template argument deduction against a single-element initializer list, the compiler will now correctly deduce the expression as a copy constructor call rather than a call to a constructor taking an initializer list. This matches the behavior described by the C++ p0702r1 paper.

```
#include <initializer_list>

template<typename T> struct X
```

```
{
  X() {};
  X(std::initializer_list<T>);
  X(const X& other) = default;
};

X xi = {0};
X xxi = {xi};   // Previously deduced as type X<X<int>>, now X<int>
```

Change in Defaults for Inlined C String Functions

Previously, the option **Inlined C String Functions** defaulted to **Off** (**-Onostrfuncs**) when using the optimization strategies **-Ogeneral** and **-Ospeed**. This is no longer the case: these strategies now enable the option **-Ostrfuncs**. In addition, this change coincides with an improvement in the speed of the `strcmp()` function when inlined.

Change in Treatment of memcpy() and memset()

Some options may cause `memcpy()` and `memset()` to redirect to alternate symbol names that implement these functions with additional constraints. If you are providing your own implementation of these functions, consider passing **-mem_funcs=fast**. For more information, see "Memory Functions" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Removed Features

The following features are no longer available:

- The option **-bsp=s32v234_evb** (NXP S32V234-EVB).

Deprecated Features

The following features have been deprecated and will be removed in a future release:

- The option **-Onomemory**.
- The option **--section_prefix**.
- The option **--section_suffix**.

- The option **--thread_safe_intiailizations**.
- The **gstack** option **--legacy**.
- The **gsrec** option **-tekhex**.

Code Generation Defects

For a list of known code generation defects, see “Code Generation Defects” on page 114.

Chapter 5

Changes in Version 2020.5

Contents

New Features and Enhancements 40
Changes in Behavior 43
Deprecated Features 45
Code Generation Defects 45

New Features and Enhancements

New DoubleCheck Features

A new option, **-double_check.enable**, is available to undo the effect of any previously applied ignore option(s) as well as to enable some checkers that are off by default. In addition, the existing **-double_check.ignore** option now supports the argument `all`. For more information, see "DoubleCheck Errors to Enable" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5* and "Ignoring and Enabling Source Analysis Errors and Warnings" in Chapter 5, "The DoubleCheck Source Analysis Tool" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Linker Option to Prevent Function Deletion

When using **-delete**, a new linker option (**-keep_from**) prevents the deletion of all functions in a specified section. For more information, see "Linker-Specific Options" in Chapter 8, "The elxr Linker" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Documentation for Writing Secure Code

A new appendix has been added to the documentation that provides guidance on writing software for safety/security-critical systems. This includes a summary of compiler features and recommended coding practices to help prevent bugs and security vulnerabilities. For more information, see Appendix D, "Secure Coding Practices" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Support for CERT C Rules

The toolchain now supports the 2016 Edition of the SEI CERT C Coding Standard, a set of secure coding rules and recommendations for the C programming language. For more information, see "Supporting CERT C Rules" in Appendix D, "Secure Coding Practices" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Options to Mitigate Spectre Vulnerabilities

Several options have been added to mitigate the straight-line Spectre vulnerability described by CVE-2020-13844. For more information, see "Straight-line Spectre Compiler Mitigation" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*, "Straight-line Spectre Linker Mitigation Check" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*, and "Straight-line Spectre Assembler Mitigation" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Option to Instrument Function Prologues and Epilogues

A new option (**-instrument_prologue_epilogue**) now allows you to combine instrumentation and code snippets to assist with tasks such as logging functions or stack overflow detection. For more information, see "Instrument Prologue/Epilogue" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Support for Half-Precision Floating-Point Type

A new half-precision floating point type, `__fp16`, is now supported. The `__fp16` data type is capable of representing `NAN` and `INFINITY`, as well as finite and subnormal numbers. It can be used for reducing memory usage in cases where reduced range and precision are acceptable. For more information, see "Half-Precision Floating-Point Support" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New gemfile Option to Improve Boot Loader Compatibility

A new **gmemfile** option (**-arm64_kernel**) allows for using a custom text offset value for loading images to RAM during boot. For more information, see "The gmemfile Utility Program" in Chapter 10, "Utility Programs" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Constant Propagation Support

The link-time constant propagation optimization is now supported (**--link_constprop**). This option will instruct the linker to optimize a program when new constants are resolved at link time. For more information, see "Link-Time Constant Propagation" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New CPU Support

The following CPUs are now supported:

- Kryo465-Gold (**-cpu=kryo465g**)
- Kryo465-Silver (**-cpu=kryo465s**)
- Kryo468-Gold (**-cpu=kryo468g**)
- Kryo468-Silver (**-cpu=kryo468s**)
- Kryo470-Gold (**-cpu=kryo470g**)
- Kryo470-Silver (**-cpu=kryo470s**)
- Kryo475-Gold (**-cpu=kryo475g**)
- Kryo475-Silver (**-cpu=kryo475s**)
- Kryo485-Gold (**-cpu=kryo485g**)
- Kryo485-Silver (**-cpu=kryo485s**)
- Kryo490-Gold (**-cpu=kryo490g**)
- Kryo490-Silver (**-cpu=kryo490s**)
- Kryo495-Gold (**-cpu=kryo495g**)
- Kryo495-Silver (**-cpu=kryo495s**)
- Kryo585-Gold (**-cpu=kryo585g**)
- Kryo585-Silver (**-cpu=kryo585s**)
- Cortex-A77 (**-cpu=cortexa77**)
- Cortex-A78 (**-cpu=cortexa78**)
- Cortex-X1 (**-cpu=cortexx1**)

New gstack Option to Set All Stack Usages at Once

A new option for **gstack** (**-constant_stack_usage**) allows you to set the individual stack usage for every function to a single value. For more information, see "The gstack Utility Program" in Chapter 10, "Utility Programs" in *MULTI: Building Applications for Embedded ARM64 v2023.5* and "Shadow Stack" in Appendix D, "Secure Coding Practices" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Support for Subnormal Results in expf()

On configurations where subnormal values are supported and not flushed to zero, `expf()` now returns a subnormal value when the correct result is subnormal rather than returning `0`.

Changes in Behavior

Stricter Type Checking for Strong Function Pointers

The compiler now performs stricter type checking for strong function pointers when taking the address of a function. Specifically, types with `__attribute__((strong_fptr))` are now taken into consideration when they occur as parameter or return types for a function type. This includes the function type for the function whose address is being taken and the function type of the object to which the address is assigned. If the strong function pointer type does not match the destination type, this will either result in an error (typically for non-relaxed `strong_fptr`s) or the compiler will apply a conservative analysis for that strong function pointer (for relaxed `strong_fptr`s).

```
typedef void (*FPTR)(void);
__attribute__((strong_fptr)) typedef void (*STRONG_FPTR)(void);
__attribute__((strong_fptr("relax"))) typedef void (*RELAX_STRONG_FPTR)(void);

void foo(STRONG_FPTR);
void bar(RELAX_STRONG_FPTR);

void example()
{
  void (*fptr1)(FPTR) = &foo; // Now rejected because parameter type mismatch

  void (*fptr2)(FPTR) = &bar; // Allowed because RELAX_STRONG_FPTR is relaxed,
```

```
                                    // but causes analysis to be very conservative.
}
```

Types with `__attribute__((strong_fptr("relax")))` are allowed to mismatch if the mismatch occurs within immediate parameter types or return types of the top-most function type. Parameter types or return types that are more deeply nested are not allowed to mismatch, even if they are relaxed strong function pointers.

By default, the compiler will enforce this stricter checking when the **-act_like** version is set to `2020.5` or greater. For other **-act_like** values, the compiler will allow many of these conversion mismatches for backwards compatibility, but will perform an extremely conservative **gstack** analysis. Note that if you are compiling code for INTEGRITY, **-act_like** may be automatically set to an appropriate version. You should be cautious about changing this value.

Placement of String Literal Data in Object Files

In prior versions, string literals inside function templates or inline functions were allocated in `.ghs.linkonce` sections that corresponded to their containing function. For example:

```
.ghs.linkonce.7..rodata.foo
```

These `.ghs.linkonce` sections were collapsed to their destination section (such as `.rodata`) at link time. Now the majority of these string literals will be allocated directly in their destination section instead. Additionally, the ordering of string literals in their respective object file section (for example, `.rodata`) may be substantially different than previous releases. This is true even for string literals in non-template and non-inline functions.

A side effect of these changes is that the compiler is now better at merging duplicate string literals from different functions, which may result in fewer string literals being allocated in the program.

Section remappings (for example, via the `#pragma ghs section` directive) are still honored for string literals. Note, however, that a duplicate string literal may not be allocated at all, and may refer to an identical string literal allocated elsewhere.

gstack Returns Non-Zero Status if User Attention is Required

The **gstack** utility will now return a non-zero exit status if it detects a problem that requires user attention. For example, **gstack** will return such a status if a program is missing the information needed to compute an accurate upper bound on the program's stack usage, or if the program computes an upper bound that exceeds the user-specified stack limit. Previously, **gstack** would return an exit status of zero in these cases.

Change in gbuild Evaluation of :reference Path

The **gbuild** utility now honors conditional control statements when evaluating the project path specified using the **:reference** option in a **Reference** project. Thus, if a path entry is conditionally disabled, the reference will not be resolved, and **gbuild** will output an error message such as `Error: Unable to resolve reference`. For example, the following error may manifest for INTEGRITY 11.7.8 when building **multivisor_mono_common_refs.gpj**:

```
Error: Unable to resolve reference:
servers.gpj;servers_ext.gpj;ghswitch_server.gpj;ghswitch_module.gpj
```

If this occurs, either upgrade to INTEGRITY 11.7.9 or contact Green Hills Support.

Deprecated Features

The following features have been deprecated and will be removed in a future release:

- The functions `flockcreate()` and `flockdestroy()`.

Code Generation Defects

For a list of known code generation defects, see “Code Generation Defects” on page 114.

Chapter 6

Changes in Version 2020.1

Contents

Licensing 48
New Features and Enhancements 48
Changes in Behavior 49
Removed Features 51
Deprecated Features 51
Code Generation Defects 51

Licensing

Beginning with Compiler 2020.1, licensing details such as local cache behavior and seat usage are specified in the licensing agreement itself. One or more instances of a new licensing binary **lic_lls** will run on hosts where the toolchain is used.

New Features and Enhancements

C11, C17, and C18 Support

The compiler now supports C11, C17, and C18. These language dialects include threads, atomics, bounds-checking library functions, aligned memory, and generics. Bear in mind that common data is not supported in C11 or later. This applies both to the option **--commons** and to the `common` attribute. For more information, see "ISO C11, ISO C17, and ISO C18" in Chapter 11, "Green Hills C" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Note

Some versions of INTEGRITY enable **--commons** in their configuration files, which is not supported for these language modes. If one of these language modes is selected in such an environment, the compiler driver will issue a warning that **--commons** will be ignored. This warning can be disabled by explicitly enabling the **--no_commons** setting.

New Support for Arithmetic Intrinsic

The following intrinsic is now supported:

- `__ADD_WITH_CARRY64();`

For more information, see "Arithmetic Operation Intrinsics" in Chapter 14, "Macros, Pragma Directives, and Intrinsics" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

C++17 Support

The compiler now supports C++17. C++17 includes hexadecimal floating-point literals, `if constexpr` statements, and enhanced support for overaligned types with `new` expressions. As with all supported C++ variations, files built or linked in C++17 mode should not be mixed with files built or linked using other dialects. For more information about C++17, see "C++17" in Chapter 12, "Green Hills C++" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Improved Deferral of Function Template Parsing

The option **--defer_parse_function_templates** is now more effective at deferring the instantiation of inline template functions.

Changes in Behavior

Error for -fNaN_cmp_unordered with -fsoft or -fnone

A fatal error is now given for **-fNaN_cmp_unordered** in conjunction with the options **-fnone** (no floating point) or **-fsoft** (software floating point). Note that **-fsoft** is enabled by default on some processors and may be set explicitly on most processors. Previously, **-fNaN_cmp_unordered** was ignored in these cases.

ax D Modifier Change

Using the **ax** librarian's **D** modifier now sets a consistent file mode independent of the host or file system permissions. Previously, the mode of a file in an archive would depend on said properties.

Discretionary Errors Now Given in C++ Standard Library Headers

Previously, the compiler suppressed discretionary error diagnostics that were caused by code in the C++ Standard Library headers. (That is, errors with a `-D` suffix on their diagnostic number.) Now, however, the compiler will emit diagnostic messages with discretionary error severity even if they arise from C++ Standard Library headers. This can cause code to be rejected with Compiler 2020.1 even though it

compiled successfully with previous releases. This should only occur with programs that are invalid since most such diagnostic messages result from violating library feature requirements.

For example: Access errors are a common discretionary error (such as from accessing a private or protected member of a class). While it is invalid to use `std::make_unique` to construct an object via a private constructor, the compiler would have previously allowed it because the access error would have been suppressed. Now the access error diagnostic will be given and compilation will fail.

Linker Directives Files with Preprocessor Directives May Trigger Warnings

The linker will now output a warning when passed linker directives files containing preprocessor directives such as `#if` or `#include`. Previously, these directives were ignored because the `#` character also signifies the beginning of a comment.

Preprocessing of linker directives files is handled by the driver when **-preprocess_linker_directive** is in effect. In this circumstance, the compiler will be used as a preprocessor and the resulting linker directives files will not have such directives.

Removal of Non-Standard Members in Unordered Containers

The unordered containers (`std::unordered_[multi]set` and `std::unordered_[multi]map`) previously defined several non-standard members:

- The member types `reverse_iterator`, `key_compare`, and `value_compare`.
- The member functions `rbegin()`, `crbegin()`, `rend()`, `crend()`, `lower_bound()`, `upper_bound()`, `key_comp()`, and `value_comp()`.

These member types and functions are no longer defined by default. If your program makes use of them, you can define the `__GHS_UNORDERED_CONTAINER_EXTENSIONS` macro to go back to the old behavior. (Note that these non-standard members, as well as the `__GHS_UNORDERED_CONTAINER_EXTENSIONS` macro, are deprecated.)

Removed Features

The following features are no longer available:

- Support for 32-bit hosts.

Deprecated Features

The following features have been deprecated and will be removed in a future release:

- The option **-Onoconstprop**.
- The functions `mktemp()` and `setlinebuf()`.

Code Generation Defects

For a list of known code generation defects, see “Code Generation Defects” on page 114.

Chapter 7

Changes in Version 2019.5

Contents

New Features and Enhancements 54
Changes in Behavior 56
Removed Features 57
Deprecated Features 57
Code Generation Defects 57

New Features and Enhancements

New Option for Trivial Function Inlining

A new option allows for inlining calls to trivial functions without deleting callees (**-inline_trivial**). For more information, see "Link-Time Trivial Function Inlining" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New CPU Support

The following CPUs are now supported:

- Cortex-A76 (**-cpu=cortexa76**)
- Cortex-A76AE (**-cpu=cortexa76ae**)
- Kryo250-Gold (**-cpu=kryo250g**)
- Kryo250-Silver (**-cpu=kryo250s**)
- Kryo260-Gold (**-cpu=kryo260g**)
- Kryo260-Silver (**-cpu=kryo260s**)
- Kryo280-Gold (**-cpu=kryo280g**)
- Kryo280-Silver (**-cpu=kryo280s**)
- Kryo360-Gold (**-cpu=kryo360g**)
- Kryo360-Silver (**-cpu=kryo360s**)
- Kryo385-Gold (**-cpu=kryo385g**)
- Kryo385-Silver (**-cpu=kryo385s**)
- Kryo460-Gold (**-cpu=kryo460g**)
- Kryo460-Silver (**-cpu=kryo460s**)

New Linker Option for Expanded Section Layout Information

A new option for map files (`-Ms`) will add information about the correlation between output sections and section inclusion rules in the linker directives file. For more

information, see “Display Section Layout in Map File” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Ability to Customize the Value of FOPEN_MAX

It is now possible to customize the value of `FOPEN_MAX`. For more information, see “Customizing the Value of FOPEN_MAX” in Chapter 15, “Libraries and Header Files” in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Option to Align Standalone Objects

A new option (**-align_standalone_objects**) allows for the direct control of optimizations that can overalign objects. For more information, see “Align Standalone Objects” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New gsrec Option for Selective Data Output

A new option for **gsrec** (**-sec** *s*) allows you to exclude sections other than *s* from data output. For more information, see “The gsrec Utility Program” in Chapter 10, “Utility Programs” in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New #pragma Directive to Influence Loop Unroll Factor

The compiler now supports the `#pragma ghs unroll=`*n* directive, which can be used to influence the unroll factor the compiler uses when unrolling a loop. This can be useful if the compiler's default choice of unroll factor is suboptimal for a given loop. For more information, see “Green Hills Extension Pragma Directives” in Chapter 14, “Macros, Pragma Directives, and Intrinsics” in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Option to Control Number of Callers Tracked with Run-Time Memory Checking

A new option (**-memcheck_num_callers**=*n*) allows you to specify how many callers should be tracked for each allocation when run-time memory checking is

enabled. For more information, see "Number of Callers to Track with Run-Time Memory Checks" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Improved Support for Input Files in gasmlist

The **gasmlist** utility now supports fully linked ELF files with DWARF information, as well as object files with DWARF or Green Hills **.dbo** information. For details, see "The gasmlist Utility Program" in Chapter 10, "Utility Programs" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Changes in Behavior

Increased Number of Callers Tracked with Run-Time Memory Checking

The memory-checking `malloc` library (**-check=alloc**/**-malloc_version=memcheck**) now tracks 8 callers for each allocation by default, up from the old value of 5. This will lead to increased heap usage when run-time memory checks are enabled. For information about configuring this value, see "Number of Callers to Track with Run-Time Memory Checks" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Modified Returns for isnormal and isfinite

The functions `isnormal()` and `isfinite()` now return a non-zero integer when their conditions are met, and `0` otherwise. Previously, they returned `1` or `0`. The new implementation is more efficient and conforms to the ISO C99 standard. However, any programs that incorrectly depend on them to return exactly `1` will no longer behave as expected.

L'\x80' and L'\xff' No Longer Accepted as Wide Characters in Shift-JIS or EUC Modes

When target character encoding set to Shift-JIS or EUC (**-kanji=shiftjis**, **-kanji=euc**, or **-kanji=euc/shiftjis**) library functions that convert from a wide character or string to a multibyte character or string now reject the values `L'\x80'` and `L'\xff'`.

These are not valid wide characters in EUC or Shift-JIS encoding. Previously, attempting to convert these characters resulted in multibyte strings that would be rejected by functions that convert in the other direction.

Removed Features

The following features are no longer available:

- Support for MULTI 6.x.

Deprecated Features

The following features have been deprecated and will be removed in a future release:

- The library functions `bcmp`, `bcopy`, and `bzero`. Use `memcmp`, `memmove`, and `memset` instead.
- The option **-Omemfuncs**.
- Support for 32-bit hosts.

Code Generation Defects

For a list of known code generation defects, see “Code Generation Defects” on page 114.

Chapter 8

Changes in Version 2019.1

Contents

New Features and Enhancements 60
Changes in Behavior 63
Deprecated Features 65
Removed Features 65
Code Generation Defects 65

New Features and Enhancements

Updated Compiler Front-End

Compiler 2019.1 is based on a newer version of the EDG C++ front-end. The new front-end includes many bug fixes and improvements in standard compliance. The default severity levels of some diagnostics may have changed.

New Option for Single-Threaded Performance

A new optimization option (**-single_threaded**) can improve performance in single-threaded programs. For more information, see "Program is Single-Threaded" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Arena malloc

A new version of `malloc` can divide the heap into separate arenas, allowing less contention when allocations occur in multi-threaded applications. For more information, see "Malloc Version" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Improved Option to Generate a Target-Walkable Stack

The **-gtws** option now has a smaller performance impact. In fully optimized code, it will no longer inhibit the tail call optimization or add a frame to simple leaf functions.

Improved ARM64 Instruction Scheduling

The compiler is now more efficient in ordering instructions to prevent structural hazards and scheduling delays.

Improved Wildcard Matching in Linker Directives Files

The toolchain's link stage has been made faster for certain projects with detailed linker directives files. These detailed files typically contain large numbers of complex wildcard patterns in their section inclusion rules. As a side effect of this improvement, some ambiguous cases in which multiple rules apply may be prioritized differently. In general, the toolchain will prioritize more specific rules over less specific rules. Ties between equally specific rules will be resolved by their order in the linker directives file. Note that this new implementation will make subtler distinctions about rule specificity. To restore the old behavior for backwards compatibility, pass `-lnk=-old_section_wildcard_matching`. For more information, see "Linker-Specific Options" in Chapter 8, "The elxr Linker" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Linker Option to Identify Separately Linked Absolute Images

A new option (**-only_data_absolute**) will produce errors any time a function is imported from a separately linked absolute image. For more information, see the table "Linker-Specific Options" in Chapter 8, "The elxr Linker" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Linker Options to Provide Errors for Symbols Importable from Multiple Files

When importing symbols from a separately linked absolute image, a new option (**-ambiguous_absolute_error**) will produce errors for symbols that can be imported from multiple import files. A companion option (**-allow_ambiguous_abs**) allows you to exclude specific symbols from these errors. For more information, see the table "Linker-Specific Options" in Chapter 8, "The elxr Linker" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Option to Assign Nameless Variables to Individual Data Sections

When used in conjunction with **-individual_data_sections**, a new option allows you to control whether compiler-created objects will be assigned to individual data sections (**-individual_data_sections_for_nameless_vars**). This new behavior is enabled by default when using **-individual_data_sections**. For more information,

see “Individual Variable Sections for Nameless Variables” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

C++ Class Array Construction No Longer References new/malloc

To ease the development of C++ applications that do not reference `malloc`, the compiler no longer generates calls to the `__vec_new` run-time library function (or related functions) when emitting code to construct arrays with automatic or static storage duration. Instead, the compiler will generate calls to `__vec_new_no_alloc` (or related functions) that do not directly reference `new`/`malloc`.

Previously, although `__vec_new` did not actually call `new`/`malloc` for arrays with automatic or static storage duration, it did have a direct reference to `new` (and thus `malloc`) that would not be optimized away.

Note

When writing a C++ application that does not reference `malloc`, considering using the **-delete** option, as it may optimize away some indirect references to `malloc`.

New Attribute and Option to Enforce Section Order

A new section attribute (`MONOTONIC`) can now be used in linker directives files to prevent reordering of sections for which multiple memory blocks are specified. For more information, see “Using Section Attributes” in Chapter 8, “The elxr Linker” in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

A new option (**-monotonic_block_order**) can also be used to instruct the linker to treat all sections as `MONOTONIC` by default. For more information, see “Monotonic Block Order” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Duplicate Strings Removed from DWARF Information

The toolchain now removes duplicate strings from the DWARF information generated by **-dual_debug** (see “Generate MULTI and DWARF Information” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for*

Embedded ARM64 v2023.5). This can significantly reduce the size of DWARF information in cases of numerous duplicated strings. For example, if the same header is included in multiple modules, only one instance of each string from the header is now included in the DWARF output.

Improved Memory Functions for INTEGRITY Applications

The toolchain can now link in its versions of memory functions when building INTEGRITY applications. This may improve run-time performance when these functions are being used. For more information, see "Enable Toolchain Versions of Certain Memory Functions" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Changes in Behavior

EDOM and ERANGE Not Defined in <math.h>

To improve conformance with the C standard, the macros `EDOM` and `ERANGE` are now defined only in **<errno.h>**. In prior versions, they were also defined in **<math.h>**. Note that this change in behavior may cause build errors if your code uses `EDOM` or `ERANGE` without including **<errno.h>**. If upgrading to 2019.1 coincides with an error like the following:

```
error #20: identifier "EDOM" is undefined
```

add this line to the error-triggering file:

```
#include <errno.h>
```

Changes to Alignment Reduction Behavior with __attribute__((aligned(x)))

The compiler will now ignore and give a warning for instances of `__attribute__((aligned(x)))` that attempt to reduce the alignment of fields, classes, structs, or unions. This behavior is designed to minimize accidental under-alignment of entities and is more consistent with the GNU compiler. Purposeful under-alignment can still be achieved with structure packing.

Updated Aligned Parameter Register Calling Convention

Parameters that are both passed in integer registers and aligned to 16 bytes or more (by an attribute or `#pragma`) are now allocated to even-numbered integer registers. This process is based on the natural alignment of a struct (derived from its most-aligned member), rather than adjustments to the struct as a whole. This change increases conformance with the AAPCS.

When std::terminate() is Called, std::uncaught_exception() Now Returns "false"

The C++ standard now directly conveys that an exception is considered implicitly handled when `std::terminate()` is called. This means that said exception is no longer "uncaught" and thus `std::uncaught_exception()` should return `false`. The compiler now implements this behavior.

Real-Time malloc is No Longer Experimental

The real-time `malloc` option **-malloc_version=realtime** is now fully supported.

New malloc Default

Arena `malloc` is now the default setting of **-malloc_version**. For more information, see “Malloc Version” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Scoped enum Without Explicit Base Type Always Given Base Type of int

A scoped `enum` without explicit base type is now correctly assigned a base type of `int`, even when **-short_enum** is enabled. Prior releases used a smaller type in some cases. Note that this change creates incompatibility with object files built with prior releases.

Deprecated Features

The following features have been deprecated and will be removed in a future release:

- Support for MULTI 6.x.

Removed Features

The following features are no longer available:

- The prelinker.
- The **--no_link_once_templates** option. (The **--link_once_templates** option is now on by default.)
- The **--implicit_include** option.
- The **--export** option (for C++ export templates).
- The **stabs2dbo** utility.

Code Generation Defects

For a list of known code generation defects, see “Code Generation Defects” on page 114.

Chapter 9

Changes in Version 2018.5

Contents

New Features and Enhancements 68
Changes in Behavior 70
Removed Features 70
Code Generation Defects 71

New Features and Enhancements

Support for MISRA 2012

The toolchain now supports the 2012 version of the Motor Industry Software Reliability Association (MISRA) standards. MISRA 2012 is a set of C guidelines designed to enforce good engineering practices when developing embedded automotive systems. For more information, see "MISRA C 2012" in Chapter 13, "Coding Standards" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Option for Enhanced Loop Unrolling

A new option (**-Ounrollmore**) is available to direct the compiler to unroll larger loops, sometimes with larger unrolling factors. This option replaces **-Ounrollbig**. For more information, see "Unroll Loops More" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Updated Support for C++11 and C++14

The Compiler has improved conformity with the C++11 and C++14 standards. The following features are now supported:

- Namespace members `std::exception_ptr` and `std::nested_exception`.
- Class Members `std::promise::set_exception` and `std::promise::set_exception_at_thread_exit`.
- Exceptions thrown by functions called with `std::packaged_task` and `std::async`.

Improved gsrec Capabilities

The **gsrec** utility now allows output of up to 248 data bytes in a single hex record, and supports padding of multi-core archives. For more information, see "The gsrec Utility Program" in Chapter 10, "Utility Programs" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Option and Attribute to Perform Address Space Layout Randomization

A new option (**--aslr_seed**) and attribute (`RANDOMIZE`) can now be used to jointly perform address space layout randomization (ASLR). ASLR can improve the security of your program. For more information, see "Address Space Layout Randomization" in Chapter 8, "The elxr Linker" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Enhanced Line Information in Conditional and Logical Operators

The compiler now generates better debugging line information for the operands of the conditional (`?:`) and logical (`||, &&`) operators. When written on separate lines, these operands can be stepped through individually. For example:

```
void foo(int *p_int)
{
  if (p_int != 0 &&
      *p_int > 10 &&      /* Now has its own breakdot */
      *p_int < 20) {      /* Also now has its own breakdot */
    /* do something */
  }
}
```

There are some limitations for this behavior. Different settings (particularly optimization strategies) may result in fewer or differently placed breakdots. Additionally, there may be code that needs to be evaluated unconditionally at the end of an expression. This code will usually be associated with the last operand of the operator. This is illustrated in the example below; the code for the assignment to `some_int_val` will typically be associated with the `(some_int_value - 1);` line. Since the assignment must happen unconditionally, it may falsely appear that the `(some_int_value - 1);` operand was also evaluated unconditionally.

```
some_int_value = (p_int != 0) ?
                   *p_int :
                   (some_int_value - 1);   /* May appear to be evaluated even
                                              if *p_int is non-NULL */
```

New CPU Support

The following CPUs are now supported:

- Cortex-A35 (**-cpu=cortexa35**).

- Cortex-A55 (**-cpu=cortexa55**).
- Cortex-A72 (**-cpu=cortexa72**).
- Cortex-A73 (**-cpu=cortexa73**).
- Cortex-A75 (**-cpu=cortexa75**).
- Kryo (**-cpu=kryo**).

Changes in Behavior

Change in Accepted Multibyte Characters for Shift-JIS and EUC

When target character encoding is set to Shift-JIS or EUC, library functions that convert from multibyte to wide characters/strings now reject the values `\x80` and `\xff`. These values are not valid starting bytes for a multibyte character in either encoding.

Change in servecode Utility Support

The utility used to generate licensing machine identifiers (**servecode**), will no longer generate or report Elan server codes for Compiler 2017.5 or later. If you need a server code for an older Elan-compatible release, use that version's **servecode**.

Removed Features

The following features are no longer available:

- The option **-Ounrollbig**. For a replacement, use **-Ounrollmore** (see "Unroll Loops More" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*).
- The legacy profiling options **-p**, **-pg**, and **-a**. For a replacement, consider **-coverage=*** (see "Profiling - Block Coverage" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*).
- Support for u-velOSity version 2.2.9.

Code Generation Defects

For a list of known code generation defects, see “Code Generation Defects” on page 114.

Chapter 10

Changes in Version 2018.1

Contents

New Features and Enhancements 74
Changes in Behavior 75
Deprecated Features 76
Removed Features 77
Code Generation Defects 77

New Features and Enhancements

C++14 Support

The compiler now supports C++14. C++14 includes support for variable templates, binary literals, and relaxed `constexpr` restrictions. As with all supported C++ variations, files built or linked in C++14 mode should not be mixed with files built or linked using other dialects. For more information about C++14, see "C++14" in Chapter 12, "Green Hills C++" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Version of malloc

The compiler now includes an experimental version of `malloc` that uses lockless instructions to speed up small block operations under contention. This improved version can also run in bounded time. For more information, see "Malloc Version" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Option to Support Floating-Point/SIMD Register Use

By default, if your target supports hardware floating-point, the compiler will now attempt to use FPU/SIMD registers (**-implicit_float_reg**). This can be disabled using **-no_implicit_float_reg**. For more information, see "Implicit Floating-Point/SIMD Register Use" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Predefined Symbols for Optimization Strategies

The Compiler now provides predefined macros for each optimization strategy. For more information, see "Optimization Predefined Macro Names" in Chapter 14, "Macros, Pragma Directives, and Intrinsics" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Intrinsic to Mitigate Against Spectre Attacks

A new intrinsic `__builtin_speculation_safe_load()` creates a barrier to branch prediction with a sequence of instructions that cannot be exploited by a variant 1 Spectre attack. Please refer to your processor manufacturer's documentation on Spectre before employing this intrinsic. For more information, see "Built-In Intrinsics" in Chapter 14, "Macros, Pragma Directives, and Intrinsics" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Note that for Spectre variant 2, ARM recommends against using retpolines. As such, we do not offer the Spectre 2 mitigation option **-indirect_retpolines** on ARM-based systems. For more information, see the relevant ARM FAQ [https://developer.arm.com/support/security-update/frequently-asked-questions]

Improved Allocation of Data Into Program Sections

The compiler now consistently allocates constants of `_Complex` type to read-only data sections. Previously, in some cases those constants would be allocated to read-write data sections. For example:

```
float _Complex foo(void) { return -5.0f + I*10.0f; }
```

Changes in Behavior

INTEGRITY Small Block malloc Moved to libfmalloc

On INTEGRITY, the small block `malloc` is now provided in **libfmalloc.a** rather than **libansi.a**. If **-act_like** is set to `2017.5` or earlier, passing `:deflibraries=ansi` will also cause the linker to link against **libfmalloc.a**. This matches the behavior of the Compiler prior to 2018.1 by using an additional library to provide the same version of `malloc` as before.

Updated Debugger Register View

The Debugger **Register View** window now displays the floating-point control registers (FPCR) and floating-point status register (FPSR).

Updated Support of Software Floating Point

Software floating point (**-fsoft**) now supports subnormal inputs on double-precision multiply operations.

Updated __MULTI_HOST__ Macro

In order to accurately reflect differences between the 32-bit and 64-bit toolchain, the `__MULTI_HOST__` macro has new values. For more information, see "Predefined Macros" in Appendix F, "Customization Files" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New -delete Behavior

The deletion of unused functions via **-delete** now takes place before reporting unresolved symbols. This means that programs referencing unresolved symbols that only occur in unused, deleted functions can now link successfully. Previously, they would give an error for unresolved symbols.

New Default Individual Sections Behavior

When individual function sections or individual data sections are enabled, these options now apply to items in link-once sections such as template instances. This new behavior can be disabled with **-no_individual_linkonce_sections**.

Deprecated Features

The following features have been deprecated and will be removed in a future release:

- Using `#pragma weak` *`foo`* where *`foo`* is a C++ mangled symbol name. To make symbols weak, use `__attribute__((weak))` instead.
- The option **-Ounrollbig**.
- The options **-fast_malloc** and **-no_fast_malloc**. Use **-malloc_version=smallblock** and **-malloc_version=legacy** instead.

Removed Features

The following features are no longer available:

- The **-nofloatio** option.
- The options **--one_instantiation_per_object** and **--hybrid_one_instantiation_per_object**.
- The **--instantiation_dir** option.

Code Generation Defects

For a list of known code generation defects, see “Code Generation Defects” on page 114.

Chapter 11

Changes in Version 2017.5

Contents

New Features and Enhancements 80
Changes in Behavior 82
Deprecated Features 83
Removed Features 84
Code Generation Defects 84

New Features and Enhancements

64-Bit Toolchain Availability

The Green Hills toolchain is now available as a set of 64-bit binaries. For more information, see “System Requirements” on page 2.

Updated Compiler Front-End

Compiler 2017.5 is based on a newer version of the EDG C++ front-end. The new front-end includes many bug fixes and improvements in standard compliance. The default severity levels of some diagnostics may have changed.

Improved Implementation of C rand() Function

The `rand()` function is now smaller, more efficient, and provides better statistical output. Note, however, that the implementation uses a linear congruential generator (LCG). This is a common and fast means of generating pseudo-random numbers that requires minimal RAM usage. However, its mathematical properties are such that it should still not be used when secure pseudo-random properties are required.

Improved Preprocessor Options

The option **-dM** may now be used with the **-E** option to generate a list of `#define` symbols. For more information, see “Driver Options for Intermediate Forms of Output” in Chapter 1, “The Compiler Driver” in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Improved Register Support

Several hundred named ARM64 registers are now supported. For more information, see the table row pertaining to `__MRS()` in “System Intrinsics” in Chapter 14, “Macros, Pragma Directives, and Intrinsics” in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Increased Flexibility with Fused Multiply Accumulate Operations

You can now disable the default behavior that allows the compiler to generate fused multiply accumulate instructions (**-no_fused_madd**). For more information, see "Floating-Point Fused Multiply and Add" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Improved Assembler Support

For ARMv8, the **asarm64** assembler now supports the AArch64 instructions specified in ARMv8.1, ARMv8.2, and the Cryptographic Extension. (The AArch32 versions are not currently supported.) Simulator support is also present for the new instructions, excepting the Cryptographic Extension.

Improved C++11 Support

The compiler now supports `std::condition_variable`, along with all library functions related to it (for example, `std::notify_all_at_thread_exit`). The compiler also now supports the use of parameter pack expansions in `alignas` specifiers.

New Option for Automatic NEON SIMD Vectorization

There is now an option to automatically transform loops performing repeated operations on arrays (**-OV**). This setting makes use of the NEON SIMD instruction set to perform multiple operations in parallel. For more information, see "Automatic Vector Optimization" in Chapter 4, "Optimizing Your Programs" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Improved Memory Function Speed

The functions `memset`, `memcpy`, `memmove`, and `memcmp` have been rewritten for improved speed.

Improved #pragma ghs section

The `#pragma ghs section` directive now supports up to 10,000 user-defined sections.

Enhanced Options for Retaining Temporary Preprocessor Files

You can now specify the directory to retain temporary assembly files (**-asmfile_dir**) and temporary preprocessor files (**-cppfile_dir**). For more information, see "Temporary Assembly File Directory" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5* and "Temporary Preprocessor Output Directory" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Changes in Behavior

Updated alignof() Extension Behavior in C++11

The C++11 standard only allows the application of the `alignof()` operator to a type-id, but most implementations allow the operator to be applied to an expression as an extension. For compatibility with other compilers, the behavior of a particular form has changed:

```
char x1 alignas(8);
int x2 = alignof(x1);
```

In prior releases, `x2` had the value of `1`. It now has the value of `8`.

Updated ABI for ARM64

The calling convention for AArch64 now follows the official ARM64 standard for procedure calls (AAPCS). Certain versions of INTEGRITY may continue to use the older calling convention for compatibility. For more information, see "Argument Passing" in Chapter 2, "Developing for ARM64" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Changes in Use of __asm__ Keyword

The compiler now rejects many attempts to allocate variables to specific registers using the `_asm_` keyword. On rejection, the compiler may serve any of the following diagnostics:

- 2034 (**ghs_invalid_register_number**): `register is not a globalreg-reserved register`.
- 2035 (**ghs_cannot_allocate_specified_register**): `cannot allocate "xxxx" to specified register`.
- 2036 (**ghs_cannot_allocate_specified_caller_saved_register**): `cannot allocate "xxxx" to specified caller-saved register`.

We strongly recommend that you avoid using this syntax whenever possible. For more information, see "GNU C/C++ Extensions" in Chapter 11, "Green Hills C" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Change in Linker Directive Files

Linker directive files (**.ld**) can now reference symbols imported from external images using the **-A** option.

Temporary Demo Incompatibility

The "Data Graphing (C++)" demo included with MULTI 7.1.6 and earlier is not compatible with this compiler release. Demo compatibility will be restored upon the availability of MULTI 8.

Deprecated Features

The following features have been deprecated and will be removed in a future release:

- The `rand_r2()` function.

Removed Features

The following features are no longer available:

- Support for Solaris-hosted toolchain components.
- The **-host64** option. This option was previously enabled by default on 64-bit operating systems. It allowed the selective use of some 64-bit toolchain components (notably the linker and **dblink**) if your project required more build-time resources than 32-bit binaries could accommodate. If your host machine is running a 64-bit operating system, you should use the newly available 64-bit toolchain (see "System Requirements" on page 2).
- The **grawrite** utility. This utility was designed to write raw binaries to floppy disks on Windows. (Unix-like systems have the **dd** command available for this purpose.) If you need to write raw binaries to floppy disks, free utilities are available online. For more information, contact Green Hills Support.

Code Generation Defects

For a list of known code generation defects, see "Code Generation Defects" on page 114.

Chapter 12

Changes in Version 2017.1

Contents

New Features and Enhancements 86
Changes in Behavior 87
Removed Features 89
Code Generation Defects 90

New Features and Enhancements

Individual Function and Variable Sections

Individual function and variable sections are now supported for assigning functions (including static functions), global variables and static variables (including function scope local static variables) in default sections or custom sections to their own individual sections by the compiler. These individual sections can then be placed at arbitrary addresses via linker directives file or be merged back to the sections they derived from implicitly at link time. For more information, see "Individual Function and Variable Sections" in Chapter 2, "Developing for ARM64" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Option to Add Extra Dot to Individual Section Names

For individual section names, the compiler now supports an option to add an extra dot to separate base section names from variable or function names (**-individual_section_name_extra_dot**). For example, the individual section name for function `<func>` in `.text` would be `.text..<func>`. For more information, see "Extra Dot in Individual Section Name" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Enhanced Support for ARM64 Intrinsics

Additional ARM64 ASP intrinsics are now supported. New additions include barrier instruction, exception, hint, and system intrinsics. For more information, see "Intrinsic Functions" in Chapter 14, "Macros, Pragma Directives, and Intrinsics" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Option for ax Librarian

The ax librarian now supports the **D** command modifier for compatibility with the GNU archiver. For more information, see "Librarian Command Modifiers" in Chapter 9, "The ax Librarian" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

CodeFactor Support

CodeFactor is now supported on ARM64. This optimization reduces code size by merging redundant sequences of object code at link time. For more information, see “Code Factoring” in Chapter 8, “The elxr Linker” in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

NEON Intrinsics Support

NEON instruction set intrinsics are now supported on ARM64. For more information, see “NEON Intrinsics” in Chapter 14, “Macros, Pragma Directives, and Intrinsics” in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Improved INTEGRITY Support for --large_vtbl_offsets

In previous releases, the compiler did not support the use of **--large_vtbl_offsets** with INTEGRITY version 11.4 or earlier. With Compiler 2017.1, **--large_vtbl_offsets** is supported on all versions of INTEGRITY and is enabled by default. With this change, the use of **--no_large_vtbl_offsets** on INTEGRITY is no longer directly supported. Contact Green Hills Support for assistance if **--no_large_vtbl_offsets** is required on INTEGRITY.

Note that this change accompanies a change in library names. Previously, Compiler-provided run-time libraries that included large virtual table offsets also included "`_xvtbl`" as part of their filenames. Now that this is the default state, the "`_xvtbl`" text has been removed. Legacy builds of the run-time libraries without large virtual table offsets are now designated with the addition of "`_svtbl`" to their names. The compiler driver automatically selects the correct library build variant for the supplied virtual table offset size.

Changes in Behavior

Modification of memmove Behavior

The inline version of `memmove()` has been removed, and thus the function will no longer be affected by **-Omemfuncs**.

The Compiler Now Generates Template Information Files in Fewer Cases

The **.ii** and **.ti** file types are now only generated if the prelinker will be invoked as part of the build process. If you are using a custom build environment with special handling for these files, it may need to be updated to reflect this change.

Modification of Boolean Comparison Behavior

In C, the compiler will no longer output diagnostic 1937 ("`possibly unintentional comparison of bool to non-bool`") when comparing a `bool` against a literal `0` or `1`. This new behavior is useful for the C99 dialect because the `true`/`false` macros in **stdbool.h** are required to expand to integer literals. Without the change, the compiler would flag expressions such as `(mybool == false)`.

Note

Diagnostic 1937 is normally a remark, but is elevated to a warning by **-ghstd=2010**. Note also that this change in behavior does not apply to C++ because the language has proper `true`/`false` literals.

Change in Rebuilding Behavior of gbuild

In previous releases, a rebuild of a project containing multiple instances of the same file could cause **gbuild** to unexpectedly produce different code than the prior build (see “Building a Project with Multiple Instances of a File” on page 117). This was caused by using multiple instances of the same file with non-identical options and the same output location. Now, **gbuild** will always build any such file multiple times, once for each set of different options. If the resulting object files are configured to output to the same location, **gbuild** will output the message "`final command has changed`" and any subsequent rebuild will cause this process to repeat and require relinking the application. To revert to the old behavior, pass **-no_rebuild_for_opt_change**.

If you are building INTEGRITY 11.4 or earlier and you see the prior **gbuild** message, it is likely because these versions of INTEGRITY included multiply instanced files in the manner described above. Note that this issue has never been observed to result in a difference in linked INTEGRITY images.

New Option Name for -display_error_number

The option **-display_error_number** is now **-display_diag_number**. The name has been changed to more accurately reflect that the option pertains to all diagnostic messages, not simply errors. The old name (**-display_error_number**) will continue to be recognized by the compiler.

Change in Accepted Constraints with GNU asm

When using GNU `asm`, the non-standard `w` and `x` constraints are no longer accepted for 32-bit and 64-bit integer registers. For example, previously the compiler would accept the following code:

```
uint64_t old_way(void)
{
    uint64_t i;
    // Set i to 0
    __asm("mov %0, xzr\n" : "=x" (i));
    return i;
}
```

Now, the `w` and `x` operand modifiers should be used instead:

```
uint64_t new_way(void)
{
    uint64_t i;
    // Set i to 0
    __asm("mov %x0, xzr\n" : "=r" (i));
    return i;
}
```

As a result of the above change, projects using certain versions of INTEGRITY for ARM64 may no longer build. If you encounter build problems with INTEGRITY for ARM64, please contact Green Hills Support for a patch.

Removed Features

The following features are no longer available:

- K&R C support.
- Windows XP support.
- Agilent Processor Probe (**hpserv**) support.

- For INTEGRITY versions after 11, the option **--munch**.
- For INTEGRITY 11 and newer, the option **-generatemonolithdeletions**.
- The <varargs.h> Facility.

Code Generation Defects

For a list of known code generation defects, see “Code Generation Defects” on page 114.

Chapter 13

Changes in Version 2016.5

Contents

New Features and Enhancements 92
Changes in Behavior 94
Deprecated Features 97
Removed Features 98
Code Generation Defects 98

New Features and Enhancements

C++11 Support

C++11 brings numerous improvements, including enhancements to performance, multithreading, and template metaprogramming. As with all supported C++ variations, files built or linked in C++11 mode should not be mixed with files built or linked using other dialects. For more information about C++11, see "C++11" in Chapter 12, "Green Hills C++" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Options Related to C++11

Along with C++11 support, the compiler now provides a number of related options:

- **--user_defined_literals** — Controls support for C++11-style user-defined literals. See "User-Defined Literals" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.
- **--thread_local_storage** — Controls support for the C++11 `thread_local` storage class specifier. (Used to indicate that a variable should reside in thread-local storage.) See "Thread-Local Storage" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.
- **--deprecated_nonconst_string_literal_conv** — Controls support for the deprecated conversion from string literal to `non-const char *` in C++11 mode. See "Allow String Literals to be Assigned to non-const char Pointers" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.
- **--narrowing_conversions** — Controls whether narrowing conversions are allowed in C++11 mode. See "Allow Narrowing Conversions in C++11 Dialects" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.
- **--thread_safe_initializations** — Controls whether initializations are thread-safe.

Support for Disabling Run-Time Type Information (RTTI) in C++11 Mode

Disabling run-time type information is now supported for C++11 mode. For more information, see "Run-Time Type Information Support" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Argument Type Checking

A new option to perform link-time argument type checking is now available. The **-argcheck** option will direct the linker to issue a warning when an externally visible function is called with a type that does not match its definition. For more information, see "Link-Time Argument Type Checking" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Floating-Point Rigor Option

A new option (**-frigor**) allows you to make accuracy/performance tradeoffs when calculating floating-point values. For more information, see "Floating-Point Rigor" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Architecture Support for Trace and TimeMachine

Trace and TimeMachine are now supported on 64-bit ARM targets when used in conjunction with MULTI 7 or newer.

New Option to Set Lower Bound of p_align

The option **-min_p_align** allows you to set a minimum value for the `p_align` field in ELF headers. For more information, see the table in "Linker-Specific Options" in Chapter 8, "The elxr Linker" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Added Support for CRC Instruction Set

The CRC instruction set extension is now supported by the assembler and simulator.

Changes in Behavior

Default C Language Dialect is Now C99

The default C language dialect is now ISO C99 (**-c99**). Note that if **-act_like** is set to `2015.1` or earlier, the compiler will still default to ANSI (**-ansi**). For more information, see "C Language Dialect" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Changes in fenv.h

The **fenv.h** header file no longer defines the floating-point exception macros `FE_DIVBYZERO`, `FE_INEXACT`, `FE_INVALID`, `FE_OVERFLOW`, and `FE_UNDERFLOW`. In addition, the `FE_ALL_EXCEPT` macro is now defined as `0`. This change more clearly indicates the compiler's lack of support for floating-point exceptions.

Changes in abort() to Conform to C++ Standard

In previous releases, `abort()` would call `exit()` if the abort signal was not implemented or if it returned. This had the side effect of calling static destructors and `atexit()` functions, which is incompatible with the C++ standard. When faced with the same circumstances, `abort()` now calls `_Exit()`. This will not invoke static destructors or `atexit()` functions.

Changes in Standard C++ Behavior

A number of changes have been made to Standard C++ mode. The primary differences visible to a Standard C++ application are listed below.

- The `std::ios_base::failure` class now uses the C++11 specification, even in Standard C++ mode. This is done to ensure binary compatibility between C++11 and Standard C++ modes.
- The class `std::system_error` is now the parent class of `std::ios_base::failure`. Previously, the parent class was `std::exception`.

- The string returned by the `what()` member function now contains the user-provided message string as a sub-string. Previously, the returned string was identical to the user-provided message string.
- The `char` specialization of `std::char_traits::lt()` now behaves the same as the built-in operator < for type `unsigned char` rather than `char`. The old behavior can be re-enabled by building with `-D__CHAR_TRAITS_CHAR_LT_COMPAT`.
- The class function `std::money_get::do_get()` now requires at least one whitespace character where `money_base::space` appears in any of the format pattern's initial elements.
- The first argument to `std::codecvt::length()` is no longer declared as `const`.
- The `std::istream:operatoor>>(streambuf *)` extraction operator performs unformatted input to conform to the C++11 standard. This means it will update the number of characters successfully read and stored for the `gcount()` member function.

C++ Binary Compatibility Across Compiler Versions

C++ headers and libraries in 2016.5 are not binary compatible with C++ code compiled with previous versions of the toolchain. Any projects with modules or libraries containing C++ should be fully rebuilt prior to linking with the updated toolchain. (Note that INTEGRITY 10 or later always includes the source files necessary to rebuild the C++ libraries.) Attempting to link together modules built with incompatible versions of the toolchain will give an error of the form:

```
[elxr] (error #104) Possible C++ ABI incompatibility between
modules libcxinit.a and hello.o
```

This diagnostic can be temporarily suppressed by passing `-elxr_diag_suppress 104`. However, this is not long-term solution. You should rebuild or obtain updated libraries to avoid this error in the future.

C++ Stream Locking

The `std::basic_streambuf` class no longer implements locking, along with the non-template versions of `streambuf` in the Embedded/Extended Embedded C++

libraries. All `streambuf`-derived classes that do not override the `_Lock()` and `_Unlock()` members are similarly affected. (For example, `std::basic_stringbuf`.) If your program requires locking in any of these classes, you must implement your own subclass that overrides the `_Lock()` and `_Unlock()` members.

Note that the `std::basic_filebuf` class in the C++ library and non-template versions of `filebuf` in the Embedded/Extended Embedded C++ libraries are unaffected by this change.

Treatment of Duplicate Inline Functions and Linkonce Template Instantiations

Previously, the linker would strictly check the sizes of linkonce template instantiation duplicates, along with inline function duplicates. Now the linker will ignore these size differences. To restore the previous behavior, pass the linker option **-lnk=-strict_linkonce_warning**.

Stricter Diagnosis of Constructor Inaccessibility

The compiler is now stricter in diagnosing constructor inaccessibility. For example, in permissive dialects, warnings used to be output when referencing inaccessible copy constructors if calls to them would be elided. These cases now result in an error, as illustrated below:

```
struct Foo {
  Foo() { }
  Foo(int a) { }
private:
  Foo(const Foo &other) { }
};

int main()
{
  Foo f = Foo(0); /* Now an error, even in permissive modes
                     and even though the call to the copy
                     constructor will be elided. */
}
```

Changes in Allowed Extensions

The following extension of `basic_string::append` is no longer supported:

```
append(const CharT* first, const CharT* last)
```

An alternative is as follows:

```
append(const CharT* s, size_type count)
```

Modifying Libraries with ax Librarian

When using the **ax** Librarian with the `-rC` argument, the specified old *libraryfile* (if it exists) is now removed before any *memberfiles* are added. If the operation fails before the final *libraryfile* is written, no *libraryfile* will exist. Previously, the old *libraryfile* was not removed prior to adding *memberfiles* and was left unchanged if the operation failed. To revert to the old behavior, pass the **-no_remove_old_archive_with_C** option to the librarian. For more information, see "Modifying Libraries" in Chapter 9, "The ax Librarian" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Deprecated Features

The following features have been deprecated and will be removed in a future release:

- The **stabs2dbo** utility.
- The option to disable `bool` type support in C++. When this option is removed, `bool` type support will be obligatory.
- The option to turn off link-once template instantiation (**--no_link_once_templates**).
- The options **--one_instantiation_per_object** and **--hybrid_one_instantiation_per_object**. For the purpose of preventing multiple definition errors with template instantiations when building libraries with C++ code, the **--link_once_templates** option may be used as a replacement. In addition, the prelinker is also deprecated, as its sole purpose was to support the above deprecated options along with **--export**, which is already deprecated.
- Support for INTEGRITY version 5.x and earlier.

Removed Features

The following features have been removed:

- Instantiation of the C++ template `std::complex<T>` for types `T` other than `float`, `double`, and `long double`.

Code Generation Defects

For a list of known code generation defects, see “Code Generation Defects” on page 114.

Chapter 14

Changes in Version 2015.1

Contents

New Features and Enhancements 100
Changes in Behavior 104
Removed and Deprecated Features 104

New Features and Enhancements

Project Build Configurations

Compiler 2015.1 makes it much easier to set up multiple build configurations of the same project. In previous releases, you might have created multiple Top Projects—each of which included shared source—to manually set up multiple build configurations. Now you can easily create your own build configurations within a single project structure. For more information, see the description of **defineConfig** in "Group 1 Directives" in Appendix C, "Green Hills Project File Format" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Note

This feature is only available with MULTI 7.x and newer.

Automated Setup of Multicore Targets

Two new options, **-ghsmc_file** and **-ghsmc_core_count**, enable the automated setup of multicore targets. Note that these options are only available for MULTI 7 or newer. For more information, see "Multi-Core Configuration File" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*. See also "Multi-Core Core Count" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Improved Configurability of the Tools for INTEGRITY

The options **-rtos_library_directory** and **-rtos_include_directory** have been added to improve configurability of the tools for INTEGRITY. These options will be used by a future release of INTEGRITY.

Using Interprocedural Optimizations with Makefiles

Interprocedural optimizations are now officially supported for programs built using makefiles. See "Interprocedural Optimizations" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*, and see

Example 1.4. Using Interprocedural Optimizations With Makefiles in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Duplicate Symbol Detection in Linker and Archiver

The linker now has an option to detect symbols that are present in more than one library being linked. While legal, and often useful, symbols being defined in multiple libraries can also be the source of subtle problems. The options **-link_reject_duplicate_lib_syms**, **-link_allow_duplicate_ghs_syms**, **-link_allow_duplicate_linkonce_syms**, and **-duplicate_whitelist** are added to control the behavior of the linker. For more information, see "Report an Error When Multiple Libraries on a Link Line Contain a Definition of the Same Symbol" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

The archiver now has an option to detect multiple definitions of a symbol when creating a library. The options **-reject_duplicates**, **-allow_cxx_duplicates**, and **-allow_linkonce_duplicates** are added to control the behavior of the archiver. For more information, see "Error on Duplicate Symbols in an Archive" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Preserving Temporary Preprocessor Files

Two new options, **-keepcppfiles** and **-keepasmfiles** can be used to retain temporary preprocessor output and temporary assembly files, respectively. See "Temporary Preprocessor Output" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5* and "Temporary Assembly Files" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5* for more information.

Support for Stripping Debug Information From Relocatable Objects and Libraries

A new option, **-stripreloc** has been added to the **gstrip** utility. When this option is passed, **gstrip** can be run on relocatable object files and libraries, in addition to executable files. When stripping relocatable object files and libraries, **gstrip** will

preserve any symbols necessary for linking. See “The gstrip Utility Program” in Chapter 10, “Utility Programs” in *MULTI: Building Applications for Embedded ARM64 v2023.5* for more information.

Section Sorting in Linker

The linker now supports the new `SORT` and `SORT_BY_NAME` commands in linker directives files to allow sections included with wildcards to be sorted by name. See “Using Wildcards in Section Inclusion Commands” in Chapter 8, “The elxr Linker” in *MULTI: Building Applications for Embedded ARM64 v2023.5* for more information.

Generating Both Numerically and Alphabetically Sorted Symbols in Map File

A new option, **-Man** is now supported by the linker to produce both a list of symbols sorted alphabetically (as with the **-Ma** option) and numerically (as with the **-Mn** option). See “Map File Sorting” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM64 v2023.5* for more information.

Annotating ELF Files with Toolchain Version Information

A new option, **-version_info** is now supported to add information about the compiler and assembler version and command line options to the `.comment` section of the output file. See “Write Version Info to .comment Section” in Chapter 3, “Builder and Driver Options” in *MULTI: Building Applications for Embedded ARM64 v2023.5* for more information.

Link-time Global Variable Type Checking

Two new options, **-globalcheck=[full|normal|none]** and **-globalcheck_qualifiers**, are now supported to perform link-time checks of global variable types. When **-globalcheck** is enabled, the compiler generates information about global variable data-types at their definition and uses. The linker checks that the types are compatible with respect to size and signedness. With **-globalcheck_qualifiers**, the qualifiers (such as `const` and `volatile`) are also checked. See “Link-Time Global Variable Type Checking” in Chapter 3, “Builder and Driver Options” in *MULTI: Building*

Applications for Embedded ARM64 v2023.5 and "Link-Time Global Variable Type Qualifiers" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5* for more information.

64-Bit Linker and dblink on Windows and Linux Hosts

The `elxr` linker and **dblink** debug information linker are now supported in 64-bit mode on 64-bit Windows and Linux hosts.

Specify the Output Name of a Dependency File

A new option, **-MF**, has been added to specify the output name of a dependency file. This name will be used for files created by the **-MD** or **-MMD** options. For more information, see "Generating Dependency and Header File Information" in Chapter 1, "The Compiler Driver" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Additional Documentation of Diagnostic Messages for Assemblers and elxr

Additional documentation of diagnostic messages has been added to the *MULTI: Toolchain Error Messages and Other Diagnostics* book.

Ability to Specify Diagnostics by Symbolic Tag Names

Several constructs that previously required a diagnostic to be specified by number now accept symbolic tag names as well. These constructs include:

- The **--diag_[suppress|remark|warning|error]** compiler options. (See "Varying Message Severity" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.)
- The `ghs nowarning` `#pragma` directive. (See "Green Hills Extension Pragma Directives" in Chapter 14, "Macros, Pragma Directives, and Intrinsics" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.)

For example, these two options are now equivalent:

- **--diag_error=9**
- **--diag_error=nested_comment**

See the *MULTI: Toolchain Error Messages and Other Diagnostics* book for the new symbolic tag names.

Support for %= Format String in GNU Extended Assembly

The compiler now accepts the `%=` special format string in GNU Extended Assembly. The string expands to a number that is unique to each specific instantiation of an `asm` statement. For more information, see Chapter 18, "GNU Extended Assembly" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Changes in Behavior

Diagnostic Messages When Building INTEGRITY 5.x and 10.x

In some rare cases, building INTEGRITY 5.x or 10.x from source may generate additional diagnostic messages. To disable these messages, see "Varying Message Severity" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5* or contact Green Hills Support.

bcmp and bcopy now treat their size arguments as unsigned

The library functions `bcmp()` and `bcopy()` now treat a size argument larger than `INT_MAX` as representing a large size, rather than a negative one. This will cause a long run time due to iteration over large arrays. Previously, both functions would do nothing if their size arguments were larger than `INT_MAX`.

Removed and Deprecated Features

The following feature has been deprecated:

- Support for the **--prelink_objects** option. As an alternative for build systems that use **clearmake**, please see the **--link_once_templates** option.

Chapter 15

Changes in Version 2014.1

Contents

New Features and Enhancements 108
Changes in Behavior 110
Removed and Deprecated Features 112

New Features and Enhancements

Cross-Project Implicit Dependency Analysis

Implicit dependency analysis is now supported across Top Projects. As a result, building a Top Project that depends on libraries or objects in another Top Project now builds the defined dependencies. Previously, you had to remember to build these dependencies yourself. This feature is especially useful for INTEGRITY users whose applications depend on source code in the INTEGRITY project tree. For more information, see "Implicit Dependency Analysis" in Chapter 10, "Utility Programs" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

New Undefined Behavior Checking

The compiler now checks for undefined behavior resulting from improperly sequenced expressions. By default, these diagnostics are issued as warnings, but they may be issued as errors via the **--diag_error=2017,2018** option. Alternatively, these diagnostics may be suppressed altogether via the **--diag_suppress=2017,2018** option.

For more information, see "Sequence Related Undefined Behavior Checking" in Chapter 11, "Green Hills C" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Range-Based For Loops

This release adds support for C++11-style range-based for loops. For more information, see "Range-Based For Loop Support" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

auto Type Keyword

This release allows you to use the `auto` keyword as a type specifier, as opposed to a storage class, for C++11-style type deduction. For more information, see "auto Type Keyword Support" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Improved Inlining in C++

The heuristics used by the compiler to determine whether or not to inline functions in C++ have been updated. Depending upon the circumstances, this may result in slight improvements in both size and speed in C++ programs.

Additionally, when using GNU C++ compatibility mode, function templates are now considered for inlining under a broader range of compiler options. Previously, function templates were only considered for inlining in GNU C++ compatibility mode when the function qualified for automatic inlining or when either intermodule inlining or **--max_inlining** was enabled for the compilation.

For more information, see:

- "Using Inlining Optimizations" in Chapter 4, "Optimizing Your Programs" in *MULTI: Building Applications for Embedded ARM64 v2023.5*,
- "Intermodule Inlining" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*, and
- "C++ Inlining Level" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Ability to Check for Use of Floating Point

The compiler is now able to give a warning or error for the use of floating point. Furthermore, this diagnostic can be given for the use of double precision and long double precision floating type, or for all floating point types. Previously, the only checking the compiler offered was to give a fatal error for all floating point types. For more information, see "Do not allow types double or long double" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Intermodule Variable Allocation

This release allows global and static variables to be allocated during interprocedural analysis, which can improve code quality when functions reference globals or are inlined into callers that are defined in other modules. For more information, see "IP Allocate Variables" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Memory Models

By default, the compiler generates signed 33-bit PC-relative relocations when loading the addresses of code and data. This is efficient for programs whose address spaces are smaller than 8 GB, but larger programs should be compiled following the Large Memory Model, which uses full 64-bit absolute relocations. For more information, see "Memory Model" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Inlining Constant Math Functions

Calls to certain math functions with constant parameters are now optimized into a constant expression. For more information, see "Inline Constant Math Functions" in Chapter 3, "Builder and Driver Options" in *MULTI: Building Applications for Embedded ARM64 v2023.5*.

Changes in Behavior

Changes in the Behavior of Some #pragma Directives

The following `#pragma` directives now have no effect when used inside a function. Additionally, using any of these `#pragma` directives inside a function will trigger a compiler warning:

- `#pragma ghs extra_stack`
- `#pragma ghs max_stack`
- `#pragma ghs max_instances`
- `#pragma alignfunc`

Previously, when used inside a function, these `#pragma` directives silently affected the next function declared (not the enclosing function).

index() and rindex() Prototypes Now Hidden

The prototypes for functions `index()` and `rindex()`, which are non-standard, are no longer visible unless:

- The `__BSD_INDEX_PROTOTYPE` symbol is defined, or
- The operating system is INTEGRITY and `_POSIX_SOURCE` is not defined.

Link-Once Template Instantiation Now Enabled by Default

The **--link_once_templates** option is now enabled by default.

Note

For targets that support large virtual function table offsets, disabling link-once template instantiation (via the **--no_link_once_templates** option) will imply **--no_large_vtbl_offsets**.

Limited Support for C-Like Structure Packing in C++

The compiler now has more explicit support for C-like structure packing in C++. In general, only C-like usage of packed structures (or classes) is supported. Most C++ specific features (for example, references, pointers to members, etc.) are not supported in combination with structure packing, nor were they before.

Packing of non-POD (Plain Old Data) classes is not supported unless the **-misalign_pack** option is enabled. (For the purposes of this restriction, the C++98/03 definition for POD is used.)

In conjunction with the above restriction, a warning will be issued if you explicitly attempt to pack a non-POD. Additionally, a warning will be issued if the current structure packing level, which is determined by the **-pack=*n*** command line option and any active `#pragma pack` directives, would have otherwise had an effect on a non-POD (that is, only if one of the fields of a non-POD would have had its alignment changed by the current structure packing level).

Note

A “Plain Old Data” class is a class that has no constructors, no destructors, no private non-static data members, no protected non-static data members, no base classes, no virtual functions, and no non-POD members.

-gs No Longer Implies -gtws

The option **-gs** no longer implies **-gtws**. This improves code size and speed.

sqrt() and sqrtf() Now Return NAN for Negative Input

Library functions `sqrt()` and `sqrtf()` now return a quiet `NAN` instead of `0.0` when the input is less than zero.

Removed and Deprecated Features

The following feature has been deprecated:

- Support for Exported Templates (**--export**).

Chapter 16

Known Issues

Contents

Code Generation Defects 114
Installation and Licensing 114
Host Support 115
Toolchain Issues 115
Target Connection Issues 118
Debugging Issues 118
File System Issues 120
Miscellaneous 121

Code Generation Defects

This section contains a summary of known code generation defects for compiler releases. These defects cover problems that could silently result in an incorrect output binary program from otherwise correct source input. For an up-to-date listing, please contact Green Hills Support.

Installation and Licensing

Name of Install Directory Cannot Contain Spaces

The MULTI Compiler cannot be run from a directory whose name contains spaces.

Compiler and Probe Missing From ginstall

If **ginstall** does not list the Compiler and Probe as installation options, you may be trying to install a 64-bit version of the Compiler on a 32-bit host, an invalid setup choice. On Linux, you may additionally see the following errors:

```
ginstall_probe: cannot execute binary file
ginstall_comp: cannot execute binary file
```

To remedy this issue, check your installation tools and note whether they are for a 64-bit host. If so, check if your host supports 64-bits. On Linux hosts, use the **arch** command. On Windows hosts, open **Control Panel → System** and look next to "System type". If your host only supports 32-bits, replace it with a 64-bit host. If you need additional assistance, contact Green Hills Support.

Empty Parent Directory Deletion During Uninstall on Windows Hosts

If the MULTI IDE, compiler, or licensing utility are uninstalled on Windows, the uninstaller will also delete the product's parent directory if said directory is now empty. This behavior functions recursively. That is, the uninstaller will delete all successive empty parent directories as it moves up their hierarchy.

Host Support

UNC Paths Are Not Supported in the MULTI Debugger

On Windows hosts with MULTI 6.1.4 or older, creating new projects or debugging projects over a UNC path is not supported. In general, we recommend mapping UNC paths to network drives before trying to access them in the MULTI IDE.

UNC Paths to Virtual Machines Are Not Supported

When remotely accessing a MULTI or Green Hills Compiler installation on a Windows virtual machine using a UNC path, you may get errors that the application could not be found or that it could not obtain a license. To work around this issue, map the path to a network drive and use the drive letter instead of the UNC path.

Toolchain Issues

INTEGRITY Fails to Build

If you install a version of the toolchain and attempt to build INTEGRITY, you may receive an error message like the following:

```
Error: Program 'C \ghs\int1144\multi\bin\win64\intex'
does not exist in this distribution
```

or

```
Error: Program '/bin/sh'
does not exist in this distribution
```

For information about building INTEGRITY, contact Green Hills Support.

gstack Provides Warnings for Green Hills Library Functions

Some functions in the Green Hills libraries are not annotated for **gstack**. If you link your program with these libraries, you may receive warnings about these Green Hills library functions when using **gstack**.

gstack Provides Warnings for INTEGRITY

INTEGRITY has not been modified to take advantage of new **gstack** features. If you use **gstack** with an INTEGRITY program, you may receive warnings about INTEGRITY functions.

Use of -a, -p, or -pg in INTEGRITY KernelSpace Requires -Onomemfuncs

If **-a**, **-p**, or **-pg** are used with INTEGRITY KernelSpace code, **-Onomemfuncs** should be used at the same time.

Common Data Syntax

The ARM64 compiler defaults to treating uninitialized global variables as unique definitions (**--no_commons**). If your code contains multiple declarations such as `int foo;` (which would cause the linker to find multiple definition spots for `foo`), this setting may also cause `multiply defined symbol` linker errors. If you rely on common data syntax and cannot modify code to avoid it, set the **--commons** option to enable that behavior. This may result in less efficient compiled code and is not a recommended practice when writing new code.

Building Projects with Absolute Path Output Locations

Building a project that specifies output locations with absolute paths will embed these paths in any resultant files. This may cause unexpected behavior if the output files are copied or moved. For example, performing a clean build on a copy of the output files may remove these files from their original directory rather than their current directory. This behavior applies to paths specified with the `%expand_path` macro function. (Note that project files generated by the **New Project** wizard may use the `%expand_path` macro function.)

To work around this issue, perform a clean build prior to copying or moving. This will ensure that no output files reference the old location.

Building a Project with Multiple Instances of a File

If you are using Compiler 2016.5 or earlier and your project contains a source, object, or executable file more than once—and the options associated with the files are not identical—**gbuild** will reuse the output file as long as it was newer than its inputs, even if the options were not identical. To correct this behavior, indicate unique output directories for each instance of the file using **-object_dir** or **:outputDir**.

Error Associated with Calling sqrt

In C++03 dialects, calling the `sqrt` function with an integral type argument may result in the following error:

```
more than one instance of overloaded function "sqrt" matches
the argument list
```

The error occurs because the full set of `sqrt` overloads are now indirectly pulled in by more headers (such as `<algorithm>`). A workaround is to cast the argument to `sqrt` to the desired type (for example, `double`).

Toolchain Components Failing to Launch

Components of the Green Hills toolchain may fail to launch on Linux systems using a 32-bit **libc.so** that is built assuming a 16-byte stack alignment. This problem has been observed with Manjaro Linux and may affect other distributions. If you experience any of these issues, we recommend that you switch to a supported Linux version. For more information, see “System Requirements” on page 2.

Increased Compile Time and Host Memory Usage with Complex Templates

The compiler may require substantial host memory and additional time to compile files containing deeply recursive templates. This is particularly true for files containing recursive instances of class templates.

Toolchain File Size Limitation

The toolchain is not guaranteed to work correctly with files that are 2 gigabytes or larger.

Target Connection Issues

Flash Driver Library libflash.a Limited to Four CFI Drivers

The flash driver library distributed with the MULTI IDE offers support for Common Flash Interface (CFI)-compatible flash devices. Drivers for these devices are configured at run time. Previous versions of **libflash.a** supported only targets with a single CFI flash device. MULTI 5 now includes support for up to four CFI flash devices. Calls to `FLASH_loadDriver()` on a fifth CFI flash device will return an error code. Multiple calls to `FLASH_loadDriver()` on a single flash base address will return the same CFI driver, even if the hardware is changed or remapped in memory. These limitations do not affect non-CFI drivers. You can force a non-CFI driver by setting the `disableCFI` flag in `FLASH_loadDriver_2()`.

Debugging Issues

Target Agent May Cause Failures When Programming Flash

The MULTI **Fast Flash Programmer** dialog contains an input field for the RAM base address. During a program operation, the flash programmer downloads target agents to this address. If the memory used by the target agent conflicts with reserved memory sections, programming may fail. Error messages printed by the flash programmer may indicate that either a hardware exception was hit or the target agent stalled. Check the link map of your executable or the memory map of the board to confirm that the RAM base address entered in the flash dialog is located after the end of any reserved sections.

References Not Available for Inlined Functions

Issuing the **browseref** or **xref** command or right-clicking a symbol (a function, type, member, variable, etc.) and selecting **Browse References** allows you to view

all references of a symbol. However, references are not stored, and consequently not displayed, for inlined functions.

Workaround: Disable inlining by compiling your program with **--no_inlining**. Note that functions defined in header files may be implicitly inlined by the compiler if this option is not used.

Cannot Debug Executables in Eclipse Projects with a Space in the Name

You cannot debug executables in Eclipse projects with a space in the name. Do not put a space in the name of an Eclipse project.

TimeMachine Does Not Track Floating-Point/SIMD Registers

TimeMachine does not attempt to reconstruct the values of floating-point/SIMD registers and displays them as `Unknown`.

C11 Debugging Issues

Some C11 features are not currently well supported by the MULTI Debugger. Below is a non-comprehensive list of issues:

- The C11 `alignof` and `alignas` operators are not supported in the Debugger. If you highlight an `alignof` or `alignas` expression, the Debugger will issue an error: "`Unknown name 'alignof' in expression`" or "`Unknown name 'alignas' in expression`".

C++11 Debugging Issues

Some C++11 features are not currently well supported by the MULTI Debugger. Below is a non-comprehensive list of issues:

- The C++11 `alignof` operator is not supported in the Debugger. If you highlight an `alignof` expression, the Debugger will issue an error: "`Unknown name 'alignof' in expression`."

Memory Leak Detection May Result in False Positives/Negatives

Memory leak detection may fail to identify some memory leaks while incorrectly reporting others. In the later case, this is because the detection algorithms do not traverse through pointers that are represented in an encoded form or stored in a misaligned memory location. Memory that is only reachable through these means will be susceptible to false positives.

NEON Vectors Incorrectly Displayed in MULTI

When debugging a MULTI project built for a big endian target, any NEON vectors allocated to registers will be incorrectly displayed as "optimized away". This bug does not affect the correctness of the generated code.

Stack Trace Causes Incorrect Disassembly

Source-level debugging information (**-G** or **-g**) is required in order for the debugger to correctly disassemble code that contains literal pools. (Note that the Green Hills C and C++ run-time is built with **-gs**.) Additionally, you should avoid setting breakpoints in literal pools, as doing so may change the underlying data on the target and lead to errant program execution.

File System Issues

Two Dots (..) Not Supported After Symbolic Links in Paths

MULTI and other Green Hills tools do not support paths that use two dots (`..`) to indicate the parent directory of a symbolic link. For example, if `symlink` is a symbolic link, the following paths are not supported:

```
/projects/symlink/..
/projects/symlink/foo/../../bar
```

NFS May Cause Poor Performance

Linux only

MULTI tools may hang if they are accessing files over an NFS server that is slow or is not responding. In certain cases, it may not be possible to terminate them, even when using `kill -9`. These problems are caused by a fundamental limitation in the design and implementation of NFS.

Workaround: To help prevent these problems, eliminate references to broken file system mount points in your user configuration directory (**~/.ghs/***) — in particular in the **integrity.dist** and/or **uvelosity.dist** files located there.

Miscellaneous

Links to Connecting Book

The MULTI documentation set contains links to the *MULTI: Configuring Connections* book. These links are disabled for MULTI and compiler versions designated for native targets.