Release Notes: GCC 4.9.2.202103-GNURL78

31st of August, 2021

CyberThor Studios Ltd. is releasing the GCC 4.9.2.202103-GNURL78, a cross compiler tool for Renesas RL78 micro-controllers.

SALIENT FEATURES

The GCC 4.9.2.202103-GNURL78 toolchain is based on:

- GCC 4.9.2 [released]
- Binutils 2.24 [released]
- Newlib 3.1.0 [released]
- GDB 7.8.2 [released]

The latest patches are applied to GCC, Binutils and Newlib sources.

ABOUT GCC 4.9.2.202103-GNURL78

Release Version:	GCC 4.9.2.202103-GNURL78
Release Date:	31st of August, 2021
Platforms Supported:	64bits Red Hat GNU/Linux v8.0 or later (or compatible distribution) Windows XP or later
Language:	C, C++
Targets:	G1X, I1X, D1X, LIN MCP, F1X, and L1X
Object File Format:	ELF

CHANGES IN GCC 4.9.2.202103-GNURL78

This section describes the fixes made in the GCC 4.9.2.202103-GNURL78 release.

1. **[Bug Fix] Shift operations result in memory access issues.**

In some cases, depending on the optimization level, the shift operations produced incorrect results. This issue is now fixed.

2. **[Bug Fix] Min/Max operations result in memory access issues.**

In some cases, when the optimize for size option (-Os) was used, the min/max operations produced incorrect results. This issue is now fixed.

3. **[Improvement] Jump tables are now disabled by default.**

Jump tables can only be used in the bottom 64 KB/the ROM memory area. So please enable the option only when the codesize doesn't reach the 64 KB limit.

4. **[Improvement] New -munaligned-access option**

New -munaligned-access option which enables optimizations on load/store instructions. Default is -mno-unaligned-access.

INSTALLER and RPM:

1. The GCC 4.9.2.202103-GNURL78 Installer onwards supports the 'Custom Installation' and 'Default Installation' modes. The 'Default Installation' mode is set by default where the tools are installed into the default location at "C:\Program Files\GCC 4.9.2.202103-GNURL78" and the user's username and activation key are silently accepted if cached in the registry.
2. The GNURL78 ABI (Application Binary Interface) is made available on our GNU Tools support website (https://llvm-gcc-renesas.com) and also provided along with Linux RPM and Windows installer.

Notes:

This installer does not provide an option to integrate the GNURL78 toolchain with e2 studio, as the e2 studio IDE will automatically detect the GNURL78 toolchain installation on start-up for integration. Alternatively, you may use the 'Toolchain Management' feature in e2 studio to achieve this.

For details on e2 studio please visit the following link below:
http://www.renesas.com/products/tools/ide/ide_e2studio/index.jsp

There is no support in this installer to integrate toolchain with the HEW IDE.

KNOWN ISSUES IN GCC 4.9.2.202103-GNURL78

This section describes the known issues in the GCC 4.9.2.202103-GNURL78 release.

1. ES is used without being initialized.

   Workaround:
   In order to initialize ES, the address should be stored in a *far* pointer before usage. So instead of:

   ```
   ((volatile reg __far*)0x000FFF).bit._1 = 0;
   ```

   the code will be:

   ```
   volatile reg __far *address0 = 0x000FFF;
   (*address0).bit._1 = 0;
   ```

2. The *__far* keyword is not supported for C++ projects.

3. The following feature is considered deprecated: Optlib library

The OPTLIB library feature is considered deprecated, due to the following reasons:

1. It does not contain all the headers and the defines of the ANSI/ISO standard.
2. Partial implementation of library functions (e.g. standard I/O functions are not all implemented)
3. The math library sacrifices precision for speed/code size (not IEEE754 compliant)

It is now removed from the product.

4. Issue with the *__builtin_signbit* function

The *__builtin_signbit* function doesn't work for 64-bit doubles

5. Issue with side-effect from VLA array parameters

Side-effect from VLA array parameters is lost
__builtin_expect showcase the same side effect issue

6. 16-bit volatile bitfields access issue.

16-bit volatile bitfields accesses are wrongfully optimized to 8-bit accesses.

7. C++11 ICE with const int

Compiling the following code with the options*: -std=c++11 -Wall -Wextra -c* results in ICE.

```
int main() {  const int (&in)[]{1,2,3,4,5};  return 0; }
```

8. ICE generated when the *-frepo* option is used

Compiling the following code with the *-frepo* options generates ICE

```
template <typename H> struct J { J(H); };
template <unsigned long, typename...> struct K;
```

```
template <unsigned long I> struct K<I> {};
template <unsigned long I, typename H, typename... T>
struct K<I, H, T...> : K<I + 1, T...>, J<H> {
  K(const H &p1, const T &... p2) : K<I + 1, T...>(p2...), J<H>(p1) {}
};
template <typename... E> struct C : K<0, E...> {
  C(const E &... p1) : K<0, E...>(p1...) {}
};
template <typename> struct A {
  A() = default;
};
struct M;
template <typename> struct L {
  struct B {
    template <typename> static M *__test(...);
    typedef A<int> _Del;
    typedef decltype(__test<_Del>()) type;
  };
  C<typename B::type, A<M>> _M_t;
  L(typename B::type) : _M_t(0, A<M>()) {}
};
struct M {};
void foo() { L<int>(new M); }
```

9. Not a constant expression issue.

When calling constexpr constructor with pointer-to-member the following code generates an error:

```
struct Foo;
struct Bar
{
    using MemberT = int Foo::*;

    MemberT h_;

    constexpr Bar(MemberT h) : h_{h}
    {
    }
};
struct Foo
{
    int test = 0;
    static constexpr Bar bar {&Foo::test};
};
constexpr Bar Foo::bar;
```

10. -Wunused-variable option warning issue.

The following code should produce the '*unused variable* b' warning:

```
Class Foo
{
};
Foo& getfoo();
void foo()
{
        Foo& f = getfoo();
}
```

```
class Bar
{
        virtual ~Bar() {}
};
Bar& getbar();
void bar()
{
        Bar& b = getbar();
}
```

11. Class reference as template parameter issue

Using a class reference as template parameter causes compilation to fail.

12. Variadic alias template with wrappers issue

Using a variadic alias template with wrappers produces ICE

13. Member class of anonymous class linkage issue

Compiling the following code with the -c option results in '*anonymous type with no linkage used to declare function*' error

```
static struct
{
    void f();
    struct Inner
    {
        void g();
    };
} x;
```

14. Assignment operator issue

Default move assignment does not move array members

15. Const overloads resolution failure

Gcc fails to take address of const overloads (see the following example):

```
namespace X {
struct S
{
    void s (int) const throw ();
    void s (int) throw ();
};
}
extern "C" {
typedef int index_t;
}
namespace X {
using ::index_t;
}
void (X::S::*f) (X::index_t) = &X::S::s;
void (X::S::*g) (X::index_t) const = &X::S::s
```

16. Array subscript issue

The following code generates an '*above array bounds*' error when *-Werror=array-bounds* is added as a compiling option

```
#include <map>
#include <vector>
#include <string>
#include <string.h>
#include <list>
class CPriceLst;
class DPriceLst;
class DDPriceLst;
class ProdBook;
class PriceLst {
public:
    virtual ~PriceLst();
};
class DPriceLst : public PriceLst
{
public:
};
class DDPriceLst : public DPriceLst
{
public:
};
class ProdBook {
public:
    PriceLst    m_bs_pr[2];
    DPriceLst   m_bs_d_pr[2];  // derived order lst
    DDPriceLst  m_bs_dd_pr[2]; // 2nd derived lst

    ProdBook();
};
class OrderBook {
public:
    std::vector<ProdBook> m_prod;
};
int main() {
        void *ptr=NULL;
        ProdBook prod;

        ((OrderBook*)ptr)->m_prod.push_back(prod);

        return 0;
}
```

17. Non-variadic template parameter issue

ICE is generated when non-variadic template parameter is the default argument of the variadic template parameter

18. Constexpr variables issue

Constexpr variables can trigger incorrect compiler warnings

19. String {} initialized array issue

The following code generates ICE:

```
#include <string>
struct C {
  std::string s;
};
int main() {
  C cs[5]{};
}
```

20. Issue with array bound check in for loops

GCC produces a false-positive '*out of bound*' warning in the following testcase

```
class A;
class C {
  void m_fn1();
  A *a_;
};
class A {
public:
  void m_fn2();
  int elements_[8];
  int num_elements_;
};
int a;
void A::m_fn2() {
  int b = 0;
  if (num_elements_)
    ++b;
  for (int i = b + 1; i < num_elements_; ++i) {
    if (elements_[i])
      ++a;
  }
}
void C::m_fn1() {
  a_->m_fn2();
}
```

21. Issue with overriding final function

Overriding final function, defined out of line, does not lead to an error

22. Issue with *Std::experimental*

Std::experimental::bad _optional_access is not by default constructible

23. Issues with -*Warray-bound* option

-*Warray-bounds* option generates additional warnings depending on parameter type/optimization level

24. *Trunc* function issue

The *trunc* function from *math.h* produces wrong results

FREE SUPPORT FOR GCC 4.9.2.202103-GNURL78

For free technical support, please register at
https://llvm-gcc-renesas.com

For your feedback and suggestions, please visit
https://llvm-gcc-renesas.com/help/contact-us/