Release Notes: GCC 8.3.0.202204-GNURX

30th of November, 2022

CyberThor Studios Ltd. is releasing the GCC 8.3.0.202204-GNURX, a cross compiler tool for Renesas RX micro-controllers.

SALIENT FEATURES

The GCC 8.3.0.202204-GNURX toolchain is based on:

- GCC 8.3.0 [released]
- Binutils 2.36.1 [released]
- Newlib 4.1.0 [released]
- GDB 7.8.2 [released]

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

ABOUT GCC 8.3.0.202204-GNURX

Release Version:	GCC 8.3.0.202204-GNURX
Release Date:	30th of November, 2022
Platforms Supported:	64bit RedHat GNU/Linux v8.0 or later (or compatible distribution) Windows XP, Windows 7, Windows 8, Windows 10
Language:	C, C++
Targets:	RX100 RX200 RX600 RX64M RX700
Object File Format:	ELF

CHANGES IN THE GCC 8.3.0.202204-GNURX

This section describes the fixes made in the GCC 8.3.0.202204-GNURX release.

GCC:

1. **[Improvement] DCMR register not getting saved/restored on function entry/exit**

In the current release, the DCMR register will be saved on function entry and restored on function exit.

2. **[Improvement] Negated floats getting promoted to double**

In previous releases, negation of floats resulted in their promotion to double, generating sub-optimal code if doubles were represented on 8 bytes.
In the current release of the compiler, promotion will no longer happen when negating floats.

3. **[Deprecation] Automatic interrupt vector entry generation**

Declaring interrupt handlers with a numeric argument resulted in the compiler creating additional symbols, which could be referenced in the linker script to initialize the interrupt vector.
This feature is being deprecated and removed in a future release, since the current implementation does not work when paired with the -gc-section command line option.

INSTALLER:

The GNURX ABI (Application Binary Interface) is made available on our GNU Tools support website (https://llvm-gcc-renesas.com) and also provided along with Linux and Windows installer.

This installer does not provide an option to integrate the GNURX toolchain with e2 studio, as the e2 studio IDE will automatically detect the GNURX 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

Integration with the HEW IDE is not supported by the installer.

KNOWN ISSUES IN GCC 8.3.0.202204-GNURX

This section describes all known issues for this particular release:

1. -Wreturn-type is enabled by default
G++ now assumes that control never reaches the end of a non-void function (i.e. without reaching a return statement). This means that you should always pay attention to -Wreturn-type warnings, as they indicate code that can misbehave when optimized.

To tell the compiler that control can never reach the end of a function (e.g. because all callers enforce its preconditions) you can suppress -Wreturn-type warnings by adding __builtin_unreachable:

```
char signchar(int i) // precondition: i != 0
{
  if (i > 0)
    return '+';
  else if (i < 0)
    return '-';
  __builtin_unreachable();
}
```

Because -Wreturn-type is now enabled by default, G++ will warn if main is declared with an implicit int return type (which is non-standard but allowed by GCC). To avoid the warning simply add a return type to main, which makes the code more portable anyway.

2. Stricter rules when using templates
G++ now diagnoses even more cases of ill-formed templates which can never be instantiated (in addition to the stricter rules in GCC 7). The following example will now be diagnosed by G++ because the type of B<T>::a does not depend on T and so the function B<T>::f is ill-formed for every possible instantiation of the template:

```
class A { };
template <typename T> struct B {
  bool f() const { return a; }
  A a;
};
```

```
In member function 'bool B<T>::f() const':
error: cannot convert 'const A' to 'bool' in return
   bool f() const { return a; }
                          ^
```

Ill-formed template code that has never been tested and can never be instantiated should be fixed or removed.

3. Changes to `alignof` results
The `alignof` operator has been changed to return the minimum alignment required by the target ABI, instead of the preferred alignment (consistent with `_Alignof` in C).

Previously the following assertions could fail on 32-bit x86 but will now pass. GCC's preferred alignment for standalone variables of type double or long long is 8 bytes, but the minimum alignment required by the ABI (and so used for non-static data members) is 4 bytes:

```
struct D { double val; };
static_assert(alignof(D) == alignof(double), "...");
struct L { long long val; };
static_assert(alignof(L) == alignof(long long), "...");
```

Code which uses alignof to obtain the preferred alignment can use __alignof__ instead.

4. Associative containers check the comparison function
The associative containers (std::map, std::multimap, std::set, and std::multiset) now use static assertions to check that their comparison functions support the necessary operations. In C++17 mode this includes enforcing that the function can be called when const-qualified:

```
      struct Cmp {
        bool operator()(int l, int r) /* not const */ { return l < r; }
      };
      std::set<int, Cmp> s;

In member function 'bool B<T>::f() const':
error: static assertion failed: comparison object must be invocable as const
       static_assert(is_invocable_v<const _Compare&, const _Key&, const _Key&>,
                     ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   bool f() const { return a; }
                          ^
```

This can be fixed by adding const to the call operator:

```
       struct Cmp {
         bool operator()(int l, int r) const { return l < r; }
       };
```

5. The following feature has been removed: Optlib library

The OPTLIB library feature is now removed, 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)

6. Section to segment mapping issue

In some rare cases there's an issue in the section to segment mapping which can cause load problems in the debugger.
This issue is being investigated and will be fixed in the next release.

FREE SUPPORT FOR GCC 8.3.0.202204-GNURX

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/