WG 14, N1848

Integer Precision Bits

David Svoboda

svoboda@cert.org

Date: 2014-07-25

The Problem

Each integer type in C takes a fixed number of bits of memory. Unsigned integers partition these bits into padding bits and value bits. Signed integers are similar, with one value bit reserved as the sign bit, and the remaining value bits represent the precision, or magnitude, of the number. The number of bits used in an integer can be obtained easily with the following expression, where **`uint_t`** represents an unsigned integer type.

```
size_t bits = sizeof(uint_t) * CHAR_BIT
```

Many programs use the number of bits in a manner that assumes that every bit is used for precision (except the sign bit). For example:

```
uint_t half_max = ((uint_t) 1) << (bits – 1);
/* 0b1000... */
```

On platforms with no padding bits, this code works correctly. But if a platform has any padding bits, then the number of value or precision bits cannot be determined from the size. On such a platform, the code example above will most likely set **`half_max`** to 0.

Workarounds

There have been many workarounds for producing the correct number of precision bits for any unsigned integer type. The simplest (but least portable) is to hardcode integer sizes per platform:

```
#ifdef IA32
#define UINT_PRECISION 32
#elif IA64
#define UINT_PRECISION 64
/* … */
#end
```

One common option makes use of a **`popcount()`** function, which takes an integer and counts the number of set bits. Some platforms provide an assembly-code instruction to accomplish this; here is a sample C implementation of this function:

```
size_t popcount(uintmax_t num) {
  size_t precision = 0;
  while (num != 0) {
    if (num % 2 == 1) {
      precision++;
    }
    num >>= 1;
  }
  return precision;
}
```

Applying this function to the maximum unsigned integer yields the number of precision bits of that integer type:

```
#define UINT_PRECISION popcount((unsigned int) -1)
```

This solution is portable. However the precision bits are not available at compile time. Consequently, useful static assertions are impossible:

```
static_assert( UINT_PRECISION >= 64)
```

Solution

We propose amending the standard with macros that indicate the number of precision bits for the standard unsigned integer types. Precision bits for the signed integer types can be derived by subtracting 1 from the precision bits for the corresponding unsigned integer type.

Proposed Wording Changes

Insert a new section before 5.2.4.2.2 with the following text:

5.2.4.2.2 Unsigned Integer Precisions **`<limits.h>`**

The values given below shall be replaced by constant expressions suitable for use in **`#if`** preprocessing directives. Moreover, the following shall be replaced by expressions of type **`size_t`**. Their implementation-defined values shall be equal or greater in magnitude (absolute value) to those shown.

- Number of precision bits for an object of type **`unsigned char`**
  `UCHAR_PRECISION                  // 8`

- Number of precision bits for an object of type **unsigned short**
  USHRT_PRECISION // 16
- Number of precision bits for an object of type **unsigned int**
  UINT_PRECISION // 16
- Number of precision bits for an object of type **unsigned long**
  ULONG_PRECISION // 32
- Number of precision bits for an object of type **unsigned long long**
  ULLONG_PRECISION // 64

Modify section 7.20.2 and 7.20.3 as follows: Text to be added is displayed in red.

7.20.2 Limits and Precisions of specified-width integer types

1 The following object-like macros specify the minimum and maximum limits and precisions of the types declared in **<stdint.h>**. Each macro name corresponds to a similar type name in 7.20.1.

2 Each instance of any defined macro shall be replaced by a constant expression suitable for use in **#if** preprocessing directives, and this expression shall have the same type as would an expression that is an object of the corresponding type converted according to the integer promotions. Its implementation-defined value shall be equal to or greater in magnitude (absolute value) than the corresponding value given below, with the same sign, except where stated to be exactly the given value.

7.20.2.1 Limits and precision of exact-width integer types

— minimum values of exact-width signed integer types

INT*N*_MIN exactly $-(2^{N-1})$

— maximum values of exact-width signed integer types

INT*N*_MAX exactly $2^{N-1}-1$

— maximum values of exact-width unsigned integer types

UINT*N*_MAX exactly $2^{N}-1$

--- number of precision bits of exact-width unsigned integer types

UINT*N*_PRECISION <= N

7.20.2.2 Limits and precision of minimum-width integer types

— minimum values of minimum-width signed integer types

INT_LEAST*N*_MIN $-(2^{N-1}-1)$

— maximum values of minimum-width signed integer types

INT_LEAST*N*_MAX $2^{N-1}-1$

— maximum values of minimum-width unsigned integer types

UINT_LEAST*N*_MAX $2^{N}-1$

--- number of precision bits of minimum-width unsigned integer types
UINT_LEAST*N*_PRECISION N

7.20.2.3 Limits and precision of fastest minimum-width integer types

— minimum values of fastest minimum-width signed integer types
INT_FAST*N*_MIN $-(2^{N-1} - 1)$
— maximum values of fastest minimum-width signed integer types
INT_FAST*N*_MAX $2^{N-1} - 1$
— maximum values of fastest minimum-width unsigned integer types
UINT_FAST*N*_MAX $2^{N} - 1$
--- number of precision bits of fastest minimum-width unsigned integer types
UINT_FAST*N*_PRECISION N

7.20.2.4 Limits and precision of integer types capable of holding object pointers

— minimum value of pointer-holding signed integer type
INTPTR_MIN $-(2^{15} - 1)$
— maximum value of pointer-holding signed integer type
INTPTR_MAX $2^{15} - 1$
— maximum value of pointer-holding unsigned integer type
UINTPTR_MAX $2^{16} - 1$
--- number of precision bits of pointer-holding unsigned integer type
UINTPTR_PRECISION 16

7.20.2.5 Limits and precision of greatest-width integer types

— minimum value of greatest-width signed integer type
INTMAX_MIN $-(2^{63} - 1)$
— maximum value of greatest-width signed integer type
INTMAX_MAX $2^{63} - 1$
— maximum value of greatest-width unsigned integer type
UINTMAX_MAX $2^{64} - 1$
--- number of precision bits of greatest-width signed integer type
INTMAX_PRECISION 63
--- number of precision bits of greatest-width unsigned integer type
UINTMAX_PRECISION 64

7.20.3 Limits and precision of other integer types

1 The following object-like macros specify the minimum and maximum limits and precision of integer
types corresponding to types defined in other standard headers.
2 Each instance of these macros shall be replaced by a constant expression suitable for use in **#if** preprocessing directives, and this expression shall have the same type as would an expression that is an object of the corresponding type converted according to the integer

promotions. Its implementation-defined value shall be equal to or greater in magnitude (absolute value) than the corresponding value given below, with the same sign. An implementation shall define only the macros corresponding to those typedef names it actually provides.263)

— limits and precision of **ptrdiff_t**
PTRDIFF_MIN −65535
PTRDIFF_MAX +65535
PTRDIFF_PRECISION 16
— limits and precision of **sig_atomic_t**
SIG_ATOMIC_MIN *see below*
SIG_ATOMIC_MAX *see below*
SIG_ATOMIC_PRECISION *see below*
— limit and precision of **size_t**
SIZE_MAX 65535
SIZE_PRECISION 16
— limits and precision of **wchar_t**
WCHAR_MIN *see below*
WCHAR_MAX *see below*
WCHAR_PRECISION *see below*
— limits and precision of **wint_t**
WINT_MIN *see below*
WINT_MAX *see below*
WINT_PRECISION *see below*

3 If **sig_atomic_t** (see 7.14) is defined as a signed integer type, the value of **SIG_ATOMIC_MIN** shall be no greater than −127, the value of **SIG_ATOMIC_MAX** shall be no less than 127, and **SIG_ATOMIC_PRECISION** shall be no less than 7; otherwise, **sig_atomic_t** is defined as an unsigned integer type, the value of **SIG_ATOMIC_MIN** shall be 0, the value of **SIG_ATOMIC_MAX** shall be no less than 255, and the value of **SIG_ATOMIC_PRECISION** shall be no less than 8.
4 If**wchar_t** (see 7.19) is defined as a signed integer type, the value of **WCHAR_MIN** shall be no greater than −127, the value of **WCHAR_MAX** shall be no less than 127, and the value of **WCHAR_PRECISION** shall be no less than 7;
otherwise, **wchar_t** is defined as an unsigned integer type, the value of **WCHAR_MIN** shall be 0, the value of **WCHAR_MAX** shall be no less than 255, and the value of **WCHAR_PRECISION** shall be no less than 8.264)
5 If **wint_t** (see 7.29) is defined as a signed integer type, the value of **WINT_MIN** shall be no greater than −32767, the value of **WINT_MAX** shall be no less than 32767, and the value of **WINT_PRECISION** shall be no less than 15;
otherwise, **wint_t** is defined as an unsigned integer type, the value of **WINT_MIN**

shall be 0, the value of **`WINT_MAX`** shall be no less than 65535, and the value of **`WINT_PRECISION`** shall be no less than 16.

Acknowledgements

This proposal was inspired by CERT Secure Coding rule INT35-C [INT35-C]. This rule itself was inspired by an email conversation between David Keaton (CERT) and Masaki Kubo (JPCERT).

References

[C99] ISO/IEC 9899:2011, C Standard

[INT35-C] INT35-C. Use correct integer precisions, CERT C Coding Standard