Adding Fundamental Type for N-bit integers

Committee: ISO/IEC JTC1 SC22 WG14

Document Number: N2534

Date: 2020-06-09

Revises document number: N2501

Authors: Melanie Blower, Tommy Hoffner, Erich Keane

Reply to:

Melanie.Blower@intel.com

Tommy.Hoffner@intel.com

Erich.Keane@intel.com

Contents

Summary of Changes

N2501

- High level introduction to the topic without proposed wording, with corrections to the original document.

N2472

- Original report.

Introduction and Rationale

We propose adding a set of special integer types spelled as _ExtInt(N), where N is an integral constant expression representing the number of bits to be used to represent the type. The goal is to provide a language spelling for all the supported extended integer types.

In most hardware programmed with C compilers, the usual 8-, 16-, 32-, 64-bit width provides satisfactory expressiveness. However, in the case of FPGA hardware, using normal integer types where the full bit-width isn't used is extremely wasteful and creates severe performance/space concerns. At the other extreme, FPGA's can support really wide integers, essentially providing arbitrary precision. For example, the clang implementation of _ExtInt provides support for bit widths to 10^{24} .

Proposed solution

A set of special extended integer types using the syntax _ExtInt(N) where N is an integer that specifies the number of bits that are used to represent the type, including the sign bit. The keyword _ExtInt is a type specifier, thus it can be used in any place a type can, including as the type of a bitfield.

An _ExtInt can be declared either signed, or unsigned by using the signed/unsigned keywords. If no sign specifier is used or if the signed keyword is used, the _ExtInt type is a signed integer and can represent negative values.

The N expression is an integer constant expression, which specifies the number of bits used to represent the type, following normal integer representations for both signed and unsigned types. Both a signed and unsigned _ExtInt of the same N value will have the same number of bits in its representation. Many architectures don't have a way of representing non power-of-2 integers, so these architectures emulate these types using larger integers. In these cases, they are expected to follow the 'as-if' rule and do math 'as-if' they were done at the specified number of bits.

In order to be consistent with the C language and make the _ExtInt types useful for their intended purpose, _ExtInt types follow the usual C standard integer conversion ranks. An _ExtInt type has a greater rank than any integer type with less precision, or any implementation defined integer type with the same precision. However, they have lower rank than any of the standard integer types with the same precision. (cf 6.3.1.1 “The rank of any standard integer type shall be greater than the rank of any extended integer type with the same width.”) Usual arithmetic conversions also work the same, where the smaller ranked integer is converted to the larger.

There is one crucial exception to the C rules for Integer Promotion: _ExtInt types are excepted from the integer promotions. Unary and binary operators typically will promote operands to int. Doing these promotions would inflate the size of required hardware on some platforms, so _ExtInt types aren't subject to the integer promotion rules. For example, if a binary expression involves operands which are both _ExtInt, rather than promoting both operands to int, the narrower operand will be promoted to match the size of the wider operand, and the result of the binary operation is the wider type.

Proposed Wording

The wording proposed is a diff from WG14 N2478. Green text is new text, while ~~red~~ text is deleted text.

Modify 5.2.4.2.1 Immediately preceding p2 add this macro definition with colored text box showing the value 64 (ULLONG_WIDTH minimum value) and description:

> The macro **EXTINT_MAXBITS** represents the maximum bitwidth N supported in the declaration of special extended integer types in the type specifier _ExtInt(N). The value EXTINT_MAXBITS shall be greater than or equal to ULLONG_WIDTH.

Modify 6.2.5 p4

- There are five *standard signed integer types*, designated as **signed char**, **short int**, **int**, **long int**, and **long long int**. (These and other types may be designated in several additional ways, as described in 6.7.2.) There are also *special extended signed integer types* designated as **_ExtInt**(N) where N is an integer constant expression that specifies the number of bits that are used to represent the type, including the sign bit. [Footnote: Many architectures don't have a way of representing non power-of-2 integers, so these architectures may emulate these types using larger integers. In these cases, they are expected to follow the 'as-if' rule and do math 'as-if' they were done at the specified number of bits.] The keyword **_ExtInt** is a type specifier, thus it can be used in any place a type can, including as the type of a bitfield. There may also be implementation-defined *extended signed integer types*.[1)] The standard, special extended signed integer types, and extended signed integer types are collectively called *signed integer types*.[2)]

Modify 6.2.5 p6

For each of the signed integer types, there is a corresponding (but different) unsigned integer type (designated with the keyword unsigned) that uses the same amount of storage (including sign information) and has the same alignment requirements. The type _Bool and the unsigned integer types that correspond to the standard signed integer types are the standard unsigned integer types. The unsigned integer types that correspond to the special extended signed integer types and the extended signed integer types are the extended unsigned integer types. The standard and extended unsigned integer types are collectively called unsigned integer types.

Modify 6.3.1.1 p1 Add 2 items following the rank of enumerated type, and modify the list item which formerly followed the rank of enumerated type

-- A special extended integer type (**_ExtInt**) has a greater rank than any integer type with less precision, and a greater rank than any implementation defined extended integer types with the same precision.

-- A special extended integer type (**_ExtInt**) has a lower rank than any of the standard integer types.

— The rank of any implementation defined extended signed integer type relative to another implementation defined extended signed integer type with the same precision is implementation-defined, but still subject to the other rules for determining the integer conversion rank.

Modify 6.3.1.1 p2 Add a 3rd element to the list:

- An object or expression with special extended integer type(**_ExtInt)** whose integer conversion rank is less than the rank of **int** and **unsigned int** is excepted from the integer promotion to **int** or **unsigned int**. The consequence of this is that the _ExtInt operand type of a unary operator is not promoted to int or unsigned int, it remains unchanged, and the result of the

[1])Implementation-defined keywords have the form of an identifier reserved for any use as described in 7.1.3.

[2])Therefore, any statement in this document about signed integer types also applies to the extended signed integer types.

unary operation is the operand type. Likewise, if a binary expression involves operands which are both **_ExtInt**, rather than promoting both operands to **int** or **unsigned int** the narrower operand will be promoted to match the size of the wider operand, and the result of the binary operation is the wider type.

Example

```
_ExtInt(2) a2; _ExtInt(3) a3; char c;

a2 * a3 /* Before the multiplication, a2 is promoted to _ExtInt(3)
and the result type is _ExtInt(3) */

a2 * c /* Before the multiplication c is promoted to int, a2 is
promoted to int and the result type is int. */
```

Modify 6.4.1 p1 Add a new keyword

_ExtInt

Modify 6.4.4.1 p 1 change integer-suffix: to include a new alternative

Integer-suffix : special-extended-int-suffix

special-extended-int-suffix: one of **xi XI**

Modify 6.4.4.1 p 5 Add 2 rows to the bottom of the table:

xi or XI	**_ExtInt**	**_ExtInt [Footnote: the width N will be calculated for the type as the smallest N which can accommodate the integer constant)**
Both u or U and xi or XI	**unsigned _ExtInt**	**unsigned _ExtInt [Footnote as above]**

Modify 6.5.3.4 changing the section name

The sizeof, _AlignOf and _Bitwidthof operators

Modify 6.5.3.4 p1

The sizeof and _Bitwidthof operator shall not be applied to an expression that has function type or an incomplete type, to the parenthesized name of such a type, or to an expression that designates a bit-field member. The _Alignof operator shall not be applied to a function type or an incomplete type.

Modify 6.5.3.4 Add new paragraph after p3. The purpose of adding _Bitwidth operator is to provide the programmer a convenient and reliable way to access the width of a _ExtInt typed expression. The value could be used to declare other objects, or it could be used as the argument to the formatted i/o macros. It would also be useful in compiler self-tests, for example static_assert(_Bitwidthof(7xi) == 4).

> The **_Bitwidth** operator yields the bitwidth of its operand type. The operand is not evaluated, and the result is an integer constant.

Modify 6.5.3.4 p5

> The value of the result of ~~both~~ these three operators is implementation-defined, and its type (an unsigned integer type) is **size_t**, defined in <stddef.h> (and other headers).

Modify 6.7.2 p1

Add **_ExtInt** to the list of type-specifier alternatives.

Modify 6.7.2 p2

Add 2 items to the list below **unsigned long long**

- _ExtInt(N), or signed _ExtInt(N)
- unsigned _ExtInt(N)

Modify 6.7.2 adding paragraph 3.1 concerning constraints,

> The expression N in _ExtInt(N) shall be an integer constant expression, that specifies the number of bits used to represent the type. The implementation defined maximum value of N, EXTINT_MAXBITS , is defined in the file <limits.h>. The value of N for an unsigned _ExtInt shall be greater than or equal to 1. The value of N for signed _ExtInt shall be greater than or equal to 2.

Modify 6.10.8.1, adding a new mandatory macro name to the list. The purpose of adding what is essentially a language builtin is to bypass the intmax_t limits on the value of integer literals cf 6.10.1p4 thus allowing programmers to express very large integer values without resorting to cumbersome and error-prone calculations.

> **__STR_TO_EXTINT__**("string literal") The unsigned _ExtInt(N) literal constant value where N is the smallest bitwidth that can contain the integer. For example, __STR_TO_EXTINT("15") is of type unsigned _ExtInt(4) and has the integer value 15. The string literal should contain only numeric characters, specifically it should not contain any unary operators such as + or -.

Modify 7.8 p 2

It declares functions for manipulating greatest-width standard integer types and converting numeric character strings to greatest-width standard integer types…

Modify 7.8 p 2 – add this sentence to the end of paragraph 2:

For the special extended integer types, it defines corresponding macros for conversion specifiers for use with the formatted input/output functions.

Modify 7.8.1 p 1

Each of the following object-like macros expands to a character string literal containing a conversion specifier, possibly modified by a length modifier, suitable for use within the format argument of a formatted input/output function when converting the corresponding integer type. These macro names have the general form of PRI (character string literals for the fprintf and fwprintf family) or SCN (character string literals for the fscanf and fwscanf family),231) followed by the conversion specifier, followed by a name corresponding to a similar type name in 7.20.1. In these names, N represents the width of the type as described in 7.20.1. For example, PRIdFAST32 can be used in a format string to print the value of an integer of type int_fast32_t. For special extended integer types, the bitwidth of the type is provided as a macro argument.

Modify 7.8.1 p2 Add an alternative to the signed list

PRIxi(N)

Modify 7.8.1 p3 Add an alternative to the unsigned list

PRIuxi(N)

Modify 7.8.1 p4 Add an alternative to the signed list

SCNxi(N)

Modify 7.8.1 p4 Add an alternative to the unsigned list

SCNuxi(N)

Modify 7.8.2 title: Functions for greatest-width standard integer types

Modify Annex E p 3, Adding a new line below CHAR_WIDTH

```
#define  EXTINT_MAXWIDTH                         16    // UINT_WIDTH
```

Acknowledgements

The authors would like to recognize the following people for their help with this work: Aaron Ballman, Jens Gustedt, Joseph S. Myers, Richard Smith

References

1. The HLS compiler: https://www.intel.com/content/www/us/en/software/programmable/quartus-prime/hls-compiler.html) refer to "Arbitrary Precision Integer"
2. The FPGA compiler: https://www.intel.com/content/www/us/en/software/programmable/sdk-for-opencl/overview.html
3. The current clang review: https://reviews.llvm.org/D73967
4. https://reviews.llvm.org/D59105 An earlier version of this feature was proposed for acceptance into clang/llvm, the code review is here.
5. An earlier version of this feature is available in Intel's oneAPI product, currently in beta test: https://software.intel.com/en-us/oneapi
6. The XiLinux HLS compiler arbitrary precision data types https://www.xilinx.com/support/documentation/sw_manuals/ug998-vivado-intro-fpga-design-hls.pdf