# Notes on Using GCC IWMMXT Vector Class Header File

## Table of Contents



## List of Figures and Tables

# 1. Overview

The IWMMXT vector class header file contains functions abstracted from the IWMXMT instruction set. By providing a convenient interface to access the underlying IWMMXT instructions through intrinsics, the classes enable Single-Instruction, Multiple-Data (SIMD) operations. The objective is to improve application performance of complex and computation-intensive audio, video, and graphical data bit streams.

IWMMXT SIMD instructions can help improve efficiency of vector operations. These instructions can be implemented using inline asm, intrinsics or the C++ vector classes. The C++ vector class reuse the standard notation in C++, makes it much easier to implement SIMD operations over other methods.

This documentation is intended for programmers writing code for the IWMMXT coprocessor, particularly code that would benefit from the use of SIMD instructions. You should be familiar with C++ and the use of C++ classes.

# 2. Using GCC IWMMXT Vector Class in C++ sources

## 2.1. Include files and naming conventions

To use IWMMXT vector class, the ***<mmclass.h>*** file must be included.

The name of each class denotes the data type, signedness, bit size, number of elements using the following generic format:

| <type> | <signedness> | <bits> | vec | <elements> |
|---|---|---|---|---|
| { I } | {s \| u } | {64 \| 32 \| 16 \| 8 } | | {1 \| 2 \| 4 \| 8} |

For example:

Is32vec2 denotes the class supporting operations on **two signed 32-bit integer** vector.

Note that not all the combinations of signedness, bits and elements are supported. Please refer to Section 3 for more details.

## 2.2. Rules for Operators

To use operators with the Ivec classes you must use one of the following three syntax conventions:

[ Ivec_Class ] R = [ Ivec_Class ] A [ operator ][ Ivec_Class ] B

**Example 1**: I64vec1 R = I64vec1 A & I64vec1 B;

[ Ivec_Class ] R = [ operator ] ([ Ivec_Class ] A,[ Ivec_Class ] B)

**Example 2**: I64vec1 R = andnot(I64vec1 A, I64vec1 B);

[ Ivec_Class ] R [ operator ]= [ Ivec_Class ] A

**Example 3**: I64vec1 R &= I64vec1 A;

[ operator ] an operator (for example, &, |, or ^ )
[ Ivec_Class ] an Ivec class
R, A, B variables declared using the corresponding Ivec classes

## 2.3. Example

A simple example usage comparison is shown in **Table 1**. In the example, we add four 8-bit integer values twice, one with signed saturation the other with unsigned saturation. You can see how much easier it is to code with the vector class. Besides using fewer keystrokes and fewer lines of code, you can use standard C++ operators and don't have to remember the different intrinsic names or complicated inline asm syntax.

**Table 1 Comparison among Inline ASM, Intrinsics, and Vector Class**

| | |
|---|---|
| Inline ASM | `__m64 op1, op2, op3, op4;`<br>`__asm volatile(`<br>`"waddbss %0, %1, %2"\`<br>`: "=y" (chks), "y" (op1), "y" (op2));`<br>`"waddbus %0, %1, %2"\`<br>`: "=y" (chku), "y" (op3), "y" (op4));`<br>`)` |
| Intrinsics | `#include <mmintrin.h>`<br>`....`<br>`__m64 op1, op2, op3, op4;`<br>`chks = _mm_adds_pi8( op1, op2 );`<br>`chku = _mm_adds_pu8( op1, op2 );`<br>`....` |
| Vector Class | `#include <mmclass.h>`<br>`....`<br>`Is8vec8 op1, op2;`<br>`Iu8vec8 op3, op4;`<br>`chks = sat_add(op1,op2);`<br>`chku = sat_add(op3,op4);`<br>`....` |

# 3. Vector Classes details

## 3.1. Overview

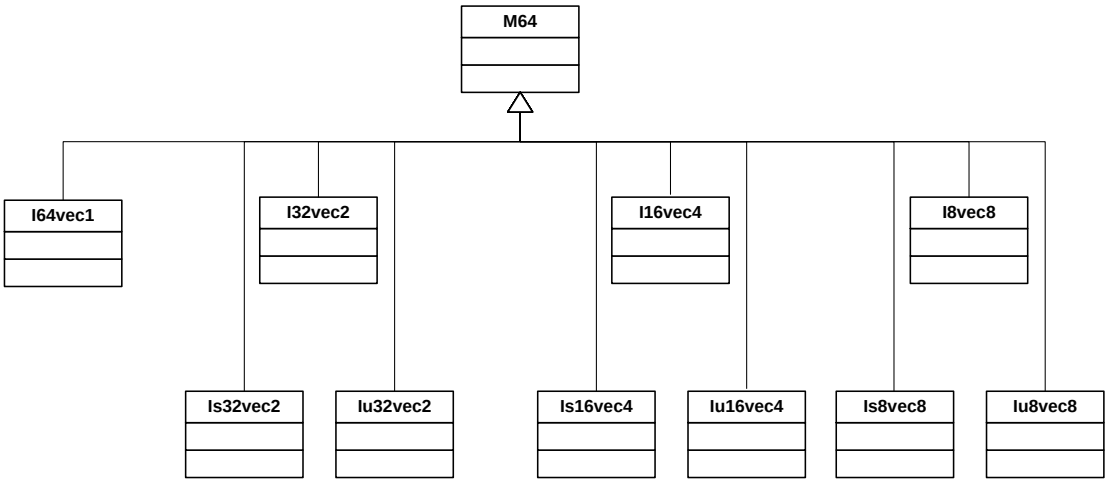


**Figure 1 Vector Class Hierarchy**

All supported vector classes are shown in **Figure 1**. The M64 class defines the __m64 data types from which the rest of the vector classes are derived. The first four child classes are derived based solely on bit sizes of 64, 32, 16 and 8 respectively for the I64vec1, I32vec2, I16vec4 and I8vec8 classes. The latter six of the classes require specification of signedness and saturation. Detail class names and their data type are listed in **Table 2**.

**Table 2 Vector Classes Data Types**

| Class | Signedness | Data Type | Size | Elements |
|---|---|---|---|---|
| I64vec1 | unspecified | __m64 | 64 | 1 |
| I32vec2 | unspecified | int | 32 | 2 |
| Is32vec2 | signed | int | 32 | 2 |
| Iu32vec2 | unsigned | int | 32 | 2 |
| I16vec4 | unspecified | short | 16 | 4 |
| Is16vec4 | signed | short | 16 | 4 |
| Iu16vec4 | unsigned | short | 16 | 4 |
| I8vec8 | unspecified | char | 8 | 8 |
| Is8vec8 | signed | char | 8 | 8 |
| Iu8vec8 | unsigned | char | 8 | 8 |

The vector class supports almost all the regular operators, but it varies a little for different class. The supported operators are: Assignment, Logical, Addition and Subtraction, Multiplication, Shift, Comparison. Details about the supported operators could be found in **Table 3**.

**Table 3 Operators Support Matrix**

| | | I8vec8 | Iu8vec8 | Is8vec8 | I16vec4 | Iu16vec4 | Is16vec4 | I32vec2 | Iu32vec2 | Is32vec2 | I64vec1 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| Assign | = | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| Logical | & | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| | &= | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| | \| | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| | \|= | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| | ^ | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| | ^= | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| | andnot | Y | Y | Y | Y | Y | Y | Y | Y | Y | Y |
| Arith | + | Y | Y | Y | Y | Y | Y | Y | Y | Y | - |
| | - | Y | Y | Y | Y | Y | Y | Y | Y | Y | - |
| | += | Y | Y | Y | Y | Y | Y | Y | Y | Y | - |
| | -= | Y | Y | Y | Y | Y | Y | Y | Y | Y | - |
| | sat_add | - | Y | Y | - | Y | Y | - | Y | Y | - |
| | sat_sub | - | Y | Y | - | Y | Y | - | Y | Y | - |
| Multiply | * | - | - | - | Y | Y | Y | Y | Y | Y | - |
| | *= | - | - | - | Y | Y | Y | Y | Y | Y | - |
| | mul_high | - | - | - | - | Y | Y | - | Y | Y | - |
| | mul_add | - | - | - | - | Y | Y | - | - | - | - |
| Shift | << | - | - | - | Y | Y | Y | Y | Y | Y | Y |
| | <<= | - | - | - | Y | Y | Y | Y | Y | Y | Y |
| | >> | - | - | - | - | Y | Y | - | Y | Y | - |
| | >>= | - | - | - | - | Y | Y | - | Y | Y | - |
| Compare | == | Y | Y | Y | Y | Y | Y | Y | Y | Y | - |
| | != | Y | Y | Y | Y | Y | Y | Y | Y | Y | - |
| | > | - | Y | Y | - | Y | Y | - | Y | Y | - |
| | >= | - | Y | Y | - | Y | Y | - | Y | Y | - |
| | < | - | Y | Y | - | Y | Y | - | Y | Y | - |
| | <= | - | Y | Y | - | Y | Y | - | Y | Y | - |
| Pack | unpack_low | Y | Y | Y | Y | Y | Y | Y | Y | Y | - |
| | unpack_high | Y | Y | Y | Y | Y | Y | Y | Y | Y | - |
| | pack_sat | - | - | - | - | - | Y | - | - | Y | - |
| | packu_sat | - | - | - | - | Y | - | - | Y | - | - |

**Note:** All operators are restricted within same class only, except the assign operator.

## 3.2. Initialization and Data access

All vector classes could be initialized by three different values in examples below. All values are initialized with the most significant element on the left and the least significant to the right.

**Example 1**: __m64 initialization

I64vec1 A(__m64 m); Iu8vec8 A(__m64 m);

**Example 2**: __int64 initialization

I64vec1 A = __int64 m; Iu8vec8 A = __int64 m;

**Example 2**: Corresponding data type initialization

I32vec2 A (int A1, int A0);

Is32vec2 A (signed int A1, signed int A0);

Iu32vec2 A (unsigned int A1, unsigned int A0);

I16vec4 A (short A3, short A2, short A1, short A0);

The value of the class could be accessed by operator __m64().

For example, __m64 (I8vec8 A) would return the value of class A (of the type __m64).

## 3.3. Assign Operators

Any Ivec object can be assigned to any other Ivec object; conversion on assignment from one Ivec object to another is automatic. **Table 4** shows the corresponding intrinsics used to implement the assign operator.

**Table 4 Corresponding Intrinsics for Assign Operators**

| **Assign** | |
|---|---|
| | **=** |
| I8vec8 | _mm_set_pi32, _mm_cvtsi32_si64 |
| Iu8vec8 | |
| Is8vec8 | |
| I16vec4 | |
| Iu16vec4 | |
| Is16vec4 | |
| I32vec2 | |
| Iu32vec2 | |
| Is32vec2 | |
| I64vec1 | |

## 3.4. Logical Operators

The logical operators use the intrinsics listed in **Table 5**.

**Table 5 Corresponding Intrinsics for Logical Operators**

| **Logicals** | | | | | | | |
|---|---|---|---|---|---|---|---|
| | **&** | **&=** | **\|** | **\|=** | **^** | **^=** | **andnot** |
| I8vec8 | _mm_and_si64 | | _mm_or_si64 | | _mm_xor_si64 | | _mm_andnot_si64 |
| Iu8vec8 | | | | | | | |
| Is8vec8 | | | | | | | |
| I16vec4 | | | | | | | |
| Iu16vec4 | | | | | | | |
| Is16vec4 | | | | | | | |
| I32vec2 | | | | | | | |
| Iu32vec2 | | | | | | | |
| Is32vec2 | | | | | | | |
| I64vec1 | | | | | | | |

## 3.5. Arithmetic Operators

Default behavior is no saturation operation. The arithmetic operators use the intrinsics listed in **Table 6**.

**Table 6 Corresponding Intrinsics for Arithmetic Operators**

| Arithmetic | | | | | | |
|---|---|---|---|---|---|---|
| | **+** | **+=** | **sat_add** | **-** | **-=** | **sat_sub** |
| I8vec8 | _mm_add_pi8 | | - | _mm_sub_pi8 | | - |
| Iu8vec8 | | | _mm_adds_pu8 | | | _mm_subs_pu8 |
| Is8vec8 | | | _mm_adds_pi8 | | | _mm_subs_pi8 |
| I16vec4 | _mm_add_pi16 | | - | _mm_sub_pi16 | | - |
| Iu16vec4 | | | _mm_adds_pu16 | | | _mm_subs_pu16 |
| Is16vec4 | | | _mm_adds_pi16 | | | _mm_subs_pi16 |
| I32vec2 | _mm_add_pi32 | | - | _mm_sub_pi32 | | - |
| Iu32vec2 | | | _mm_adds_pu32 | | | _mm_subs_pu32 |
| Is32vec2 | | | _mm_adds_pi32 | | | _mm_subs_pi32 |
| I64vec1 | - | | | - | | |

## 3.6. Multiplication Operators

Default is to return the lower 32-bits of result. The multiplication operators use the intrinsics listed in **Table 7**.

**Table 7 Corresponding Intrinsics for Multiplication Operators**

| Multiply | | | | |
|---|---|---|---|---|
| | ***** | ***=** | **mul_high** | **mul_add** |
| I8vec8 | - | | | |
| Iu8vec8 | | | | |
| Is8vec8 | | | | |
| I16vec4 | _mm_mullo_pi16 | | - | - |
| Iu16vec4 | | | _mm_mulhi_pu16 | _mm_madd_pu16 |
| Is16vec4 | | | _mm_mulhi_pi16 | _mm_madd_pi16 |
| I32vec2 | _mm_mullo_pi32 | | - | - |
| Iu32vec2 | | | _mm_mulhi_pu32 | - |
| Is32vec2 | | | _mm_mulhi_pi32 | |
| I64vec1 | - | | | |

## 3.7. Shift Operators

The right shift argument can be any integer or Ivec value. The shift operators use the intrinsics listed in **Table 8**.

**Table 8 Corresponding Intrinsics for Shift Operators**

| Shift | | | | |
|---|---|---|---|---|
| | << | <<= | >> | >>= |
| I8vec8 | - | - | - | - |
| Iu8vec8 | - | - | - | - |
| Is8vec8 | - | - | - | - |
| I16vec4 | _mm_sll_pi16, _mm_slli_pi16 | | - | |
| Iu16vec4 | | | _mm_srl_pi16,_mm_srli_pi16 | |
| Is16vec4 | | | _mm_sra_pi16,_mm_srai_pi16 | |
| I32vec2 | _mm_sll_pi32, _mm_slli_pi32 | | - | |
| Iu32vec2 | | | _mm_srl_pi32,_mm_srli_pi32 | |
| Is32vec2 | | | _mm_sra_pi32,_mm_srai_pi32 | |
| I64vec1 | _mm_sll_pi64, _mm_slli_pi64 | | - | |

## 3.8. Pack/Unpack Operators

The pack/unpack operators use the intrinsics listed in **Table 9**.

**Table 9 Corresponding Intrinsics for Pack/Unpack Operators**

| Pack/Unpack | | | | |
|---|---|---|---|---|
| | **unpack_low** | **unpack_high** | **pack_sat** | **packu_sat** |
| I8vec8 | _mm_unpacklo_pi8 | _mm_unpackhi_pi8 | - | - |
| Iu8vec8 | | | - | - |
| Is8vec8 | | | - | - |
| I16vec4 | _mm_unpacklo_pi16 | _mm_unpackhi_pi16 | - | - |
| Iu16vec4 | | | - | _mm_packs_pu16 |
| Is16vec4 | | | _mm_packs_pi16 | - |
| I32vec2 | _mm_unpacklo_pi32 | _mm_unpackhi_pi32 | - | - |
| Iu32vec2 | | | - | _mm_packs_pu32 |
| Is32vec2 | | | _mm_packs_pi32 | - |
| I64vec1 | - | | | |

## 3.9. Comparison Operators

The equality and inequality comparison operands can have mixed signedness, but they must be of the same size. The comparison operators for “less-than” and “greater-than” must be of the same sign and size. The comparison operators use the intrinsics listed in **Table 10**.

**Table 10 Corresponding Intrinsics for Comparison Operators**

| Compare | | | | | | |
|---|---|---|---|---|---|---|
| | == | != | > | < | >= | <= |
| I8vec8 | _mm_cmpeq_pi8 | _mm_andnot_si64, _mm_cmpeq_pi8 | - | | - | |
| Iu8vec8 | | | _mm_cmpgt_pu8 | | _mm_andnot_si64, _mm_cmpgt_pu8 | |
| Is8vec8 | | | _mm_cmpgt_pi8 | | _mm_andnot_si64, _mm_cmpgt_pi8 | |
| I16vec4 | _mm_cmpeq_pi16 | _mm_andnot_si64, _mm_cmpeq_pi16 | - | | - | |
| Iu16vec4 | | | _mm_cmpgt_pu16 | | _mm_andnot_si64, _mm_cmpgt_pu16 | |
| Is16vec4 | | | _mm_cmpgt_pi16 | | _mm_andnot_si64, _mm_cmpgt_pi16 | |
| I32vec2 | _mm_cmpeq_pi32 | _mm_andnot_si64, _mm_cmpeq_pi16 | - | | - | |
| Iu32vec2 | | | _mm_cmpgt_pu32 | | _mm_andnot_si64, _mm_cmpgt_pu32 | |
| Is32vec2 | | | _mm_cmpgt_pi32 | | _mm_andnot_si64, _mm_cmpgt_pi32 | |
| I64vec1 | - | | | | | |

# References

[1] *Intel® C++ Compiler for Linux Compiler Reference* (Document Number: 307777 - 002US), Intel

[2] Intel Wireless MMX Technology Intrinsic Support (Chapter 20 of *Marvell C++ Compiler Users' Manual* ), Marvell, December, 2008