

Quad Integer Library Reference

Copyright ©2001-2003, Astra Network Inc. All Rights Reserved

January 3, 2003

This is documentation for a BETA version of HAFTA. Please be advised that you may encounter problems; please report them so that they may be fixed. The latest information about HAFTA is always available at <http://www.astranetwork.com/hafta/>.

Contents

1	Introduction	2
2	Issues	2
2.1	Basic Integer Operations	2
2.2	Comparisons	3
2.3	Type Conversions	3
2.4	Rules	4
3	Portability Macros	4
3.1	Arithmetic Operations	4
3.2	Bitwise Operations	5
3.3	Comparisons	5
3.4	Type Conversions	5
4	Library Calls	6
5	Acknowledgements	7
5.1	BSD License	7
5.2	Astra license	7

1 Introduction

The Quad Integer Library is full set of functions to perform operations (arithmetic, logical, etc) on integer that are twice the size of the compiler's native long integer. The size of a quad integer is generally 64-bits, but this library does not depend on it.

The library contains a set of functions and a set of macros that wrap those functions. The macros should be used when created a software product that is to remain portable. If the macros are used on a compiler that has a **long long** integer type then the operations built into the compiler are used, if not the macros calls the respective library call to complete the operation.

It is generally suggested that you always call the macros instead of the calling the library calls directly.

2 Issues

In this section I will attempt to explain the issues involved in using this library to write a more portable (with respect to quad_t operations) program.

2.1 Basic Integer Operations

The most obvious thing to keep in mind is simple arithmetic operations. On a compiler that does not have a builtin quad_t, the Quad Integer Library will create a quad_t type that is a structure composed of two long integers. The compiler does not understand that we wish to use this type as a long long integer, it only knows that it is a structure. To write code that adds two structures means nothing to the compiler and it will readily complain about it. The solution is to wrap all arithmetic (and bitwise) operations with macros supplied by this library. This is simple enough to do. Take for example this code.

```
quad_t x, a, b;  
x = a + b;
```

This code is wrong, a compiler with no native quad_t has no clue what to do with this. The addition should be wrapped like this:

```
quad_t x, a, b;  
x = QADD( a, b );
```

This snippet will compile and will work optimally on both a compiler that have a native quad_t and one that doesn't.

All of the arithmetic and bitwise operators that are available in C for standard integer types have corresponding macros that work in this manner.

2.2 Comparisons

Comparing two `quad_t`'s is also something to look out for. The compiler with no `quad_t` also has no idea how to compare two `quad_t`'s, so you must also wrap these operations. Take the following snippet:

```
void test( quad_t a, quad_t b )
{
    if( a == b )
        printf("a equals b\n");
    else if( a > b )
        printf("a greater than b\n");
    else if( a < b )
        printf("a less than b\n");
}
```

The above should be rewritten like this, to allow it to be compiled on all compilers.

```
void test( quad_t a, quad_t b )
{
    if( QEQ(a, b) )
        printf("a equals b\n");
    else if( QGT(a, b) )
        printf("a greater than b\n");
    else if( QLT(a, b) )
        printf("a less than b\n");
}
```

2.3 Type Conversions

The automatic conversion of types could be viewed as a wonderfully flexible feature of C, but when it comes to introducing abstract integer-like types this feature makes programmers expect a lot. Unfortunately there is not much we can do other than supply a set of macros to convert between types.

In C, it is very common to see the following code:

```
int a = 5;
long b;
b = a + 99;
```

This is perfectly valid code when using `int`'s and `long`'s but not when using `quad_t`'s. Similar code using `quad_t`'s would look like this:

```
int a = 5;
long b;
quad_t q;
q = QADD( LTOQ(a), LTOQ(99) ); // This will promote a and
                               // 99 to quad_t and then
```

```
                                // perform the addition
                                // and assignment.
    b = QTOL(q); // This will truncate q to fit into a long.
```

2.4 Rules

If you follow these rules, and understand fully that `quad_t` is not just another integer type, you will be able to create perfectly portable (with respect to `quad_t`, anyway) programs.

- Never assign anything to a `quad_t` except a `quad_t`.
- Always pass/return integers of the proper type to and from functions.
- Do not use C's built-in arithmetic, comparison or bitwise operators for operations involving `quad_t`'s.
- Do not assume a `quad_t` has the same boolean properties as standard integer types. Always convert using `QTOBOOL`.
- Never use `[s]printf()`'s `%lld` option for reading/writing `quad_t`'s. It will not exist on compilers with no `quad_t`.
- Never assume a integer type will be automatically promoted or truncated to or from a `quad_t`.

3 Portability Macros

3.1 Arithmetic Operations

```
quad_t QADD(quad_t x, quad_t y)
    Return (x + y)
quad_t QSUB(quad_t x, quad_t y)
    Return (x - y)
quad_t QMUL(quad_t x, quad_t y)
    Return (x * y)
quad_t QDIV(quad_t x, quad_t y)
    Return (x / y)
quad_t QMOD(quad_t x, quad_t y)
    Return (x % y)
quad_t QNEG(quad_t x)
    Return (0 - x)
```

3.2 Bitwise Operations

quad_t QAND(quad_t x, quad_t y)
Return (x & y)

quad_t QOR(quad_t x, quad_t y)
Return (x | y)

quad_t QXOR(quad_t x, quad_t y)
Return (x XOR y)

quad_t QNOT(quad_t x)
Return (~ x), ie one's complement of x.

3.3 Comparisons

bool QGT(quad_t x, quad_t y)
Return (x > y)

bool QLT(quad_t x, quad_t y)
Return (x < y)

bool QEQ(quad_t x, quad_t y)
Return (x == y)

3.4 Type Conversions

quad_t LTOQ(long x)
Returns a quad_t with the lower bits filled with x.
Effectively converting a long to a quad_t.

long QTOL(quad_t x)
Returns the lower bits of x. Truncating a quad_t
into a long.

bool QTOBOOL(quad_t x)
Return a bool containing non-zero if x is non-zero
and zero otherwise. Effectively convert a quad_t
to a boolean.

char * QTOSTR(quad_t x)
Return a string of the base 10 representation of x.
This call will allocate space on the heap to store
the string, so this memory must be free()'d when
no longer in use.

quad_t STRTOQ(char *str)
Return the quad_t representation of the base 10
number contained in str.

4 Library Calls

```
quad_t quad_add ( quad_t a, quad_t b )
quad_t quad_and ( quad_t a, quad_t b )
quad_t quad_ashl ( quad_t a, qshift_t shift )
quad_t quad_ashr ( quad_t a, qshift_t shift )
int quad_cmp ( quad_t a, quad_t b )
quad_t quad_div ( quad_t a, quad_t b )
quad_t quad_ior ( quad_t a, quad_t b )
quad_t quad_lshl ( quad_t a, qshift_t shift )
quad_t quad_lshr ( quad_t a, qshift_t shift )
quad_t quad_ltoq ( long a )
long quad_qtol ( quad_t a )
int quad_qtobool ( quad_t a )
quad_t quad_mod ( quad_t a, quad_t b )
quad_t quad_mul ( quad_t a, quad_t b )
quad_t quad_neg ( quad_t a )
quad_t quad_not ( quad_t a )
u_quad_t quad_divrem ( u_quad_t uq, u_quad_t vq, u_quad_t *arq )
quad_t quad_sub ( quad_t a, quad_t b )
quad_t quad_swap ( quad_t a )
int quad_ucmp ( u_quad_t a, u_quad_t b )
u_quad_t quad_udiv ( u_quad_t a, u_quad_t b )
u_quad_t quad_umod ( u_quad_t a, u_quad_t b )
quad_t quad_xor ( quad_t a, quad_t b )
```

5 Acknowledgements

The bulk of this code was taken from FreeBSD's libc. It was adopted for more general use by Astra Network Inc.

5.1 BSD License

Portions of code under the BSD license:

Copyright (c) 1992, 1993
The Regents of the University of California. All rights reserved.

This software was developed by the Computer Systems Engineering group at Lawrence Berkeley Laboratory under DARPA contract BG 91-66 and contributed to Berkeley.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. All advertising materials mentioning features or use of this software must display the following acknowledgement:
This product includes software developed by the University of California, Berkeley and its contributors.
4. Neither the name of the University nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE REGENTS AND CONTRIBUTORS ‘‘AS IS’’ AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

5.2 Astra license

Other portions of code are released under Astra's license:

Copyright, 2001, Astra Network Inc. All Rights Reserved

This source code has been published by Astra Network Inc. However, any use, reproduction, modification, distribution or transfer of this software, or any software which includes or is based upon any of this code, is only permitted if expressly authorized by a written license agreement from Astra. Contact your Astra representative directly for more information.