Document Number: WG14 N731/X3J11

STANDARDIZATION OF BASIC I/O HARDWARE ADDRESSING (Draft 1.5).

Title: STANDARDIZATION OF BASIC I/O HARDWARE ADDRESSING.
Author: Jan Kristoffersen
Author Affiliation: RAMTEX International AS
Postal Address: Box 84, 2850 Naerum, Denmark
E-mail Address: jkristof@pip.dknet.dk
Telephone Number: +45 45505357
Fax Number: +45 45505390
Sponsor: DS
Date: June 27, 1997
Proposal Category:
   __ Editorial change/non-normative contribution
   __ Correction
   XX New feature
   __ Addition to obsolescent feature list
   __ Addition to Future Directions
   __ Other (please specify)
Area of Standard Affected:
   __ Environment
   __ Language
   __ Preprocessor
   XX Library
      __ Macro/typedef/tag name
      XX Function
      XX Header
   XX Other (please specify) New Annex

Prior Art: Widely used
Target Audience: Device driver programmers, I/O chip manufactures

Related Documents (if any):

WG14 N509/X3J11 95-110	Introduction paper
WG14 N558/X3J11 95-110	Proposal
WG14 N656/X3J11	Revised Proposal
WG14 N706/X3J11	Revised Proposal pre London

This draft of the document (1.5) is edited according to the feedback and corrections which came at the London meeting. The wording in this document assumes that bool is an internal type.

Annex X (Draft 1.5)
(informative)
Basic I/O hardware addressing <iohw.h>

X.1 Introduction

[1] The header <iohw.h> declares several functions for making direct access to I/O registers in peripheral hardware. [1]

The purpose of these functions is to make device driver functions for peripheral I/O hardware more portable between different execution environments and implementations. [2]

X.1.1 Conceptual model for I/O registers

[1] A given I/O register is always considered to have a fixed size in number of bits. This register size is independent of the execution environment and the implementation.
The I/O functions in <iohw.h> consider an I/O register to belong to one of the following categories: 1 bit, 8 bit, 16 bit, 32 bit, or 64 bit.

[2] The I/O functions in <iohw.h> deals with four basic operations on I/O registers: read, write, bit-wise AND, and bit-wise OR. [3]

[3] I/O registers, or individual bits in a register, are assumed to belong to one of the categories below. Each category places different restrictions on allowed operations.[4]

---

[1] The term peripheral hardware includes, but is not limited to, I/O chips, I/O cells, and I/O circuitry.
A peripheral hardware device can be considered as a collection of individual I/O registers.

[2] I/O driver portability is defined as follows:
a. If it is possible to reuse I/O hardware with another platform, then it should also be possible to reuse the hardware device drivers for that I/O hardware in the new platform, without any modifications to the driver source code.
b. If there exists two translators for the same platform, it should be possible to compile I/O hardware device driver source code with either translators, and get the same logical behaviors time of execution.

[3] AND operations are used for clearing register bits. OR operations are used for setting register bits.

[4] As to which access category an given I/O register belongs to depends of the physical characteristics of the register itself and its physical location in the execution environment.

Register category	Allowed operation
Read-Only	Read
Write-Only	Write
Read-and-Write [5]	Read, Write
Read-Modify-Write [6]	Read, Write, AND, OR

The result of an operation not allowed by the register category shall be considered as undefined.

X.2 Access type parameter

[1] The I/O functions declared in <iohw.h> shall use a single parameter, the access type, as identification of an I/O register. The access type parameter represents or references a complete description of how a given register is connected in the given hardware platform with the given processor architecture. [7]

[2] Access type parameter definitions shall be included via the header <iohw.h>. [8]

X.2.1 Access type parameter definitions

[1] The definition of access type parameters will be implementation defined. [9]

[2] A typical implementation will operate with the following five access type categories, one for each register size:

```
access1
access8
access16
access32
access64
```

[3] An I/O register definition shall contain a symbolic I/O register name which shall represents a complete specification of the access method for the I/O register in the given hardware platform.

---

5 Behaves like a read-only register and a write-only register on the same physical address.

6 Behaves like a memory cell.

7 That is, how the I/O register should be addressed.

8 This encapsulates the descriptions of processor and platform architecture differences.
The recommended practice is to let <iohw.h> contain I/O function definitions specific to the processor architecture, and let <iohw.h> include a separate header <iohw_ta.h> which contain access type definitions for all I/O registers in the given target platform.

9 An access type definition will typically depend on the given processor architecture, platform hardware, and translator.

Such a specification is implementation defined and can in addition to the symbolic name contain any number of elements. A typical specifications may contain the following elements:

1. Symbolic name.
2. I/O register data size: (i.e., 1, 8, 16, 32, or 64 bits)
3. Address space used (if more than one address space)

4a. Physical address (if data size = register size)
or
4b. Physical address, bit position. (if bit is in a register)

X.2.2 I/O function implementation

[1] A conforming implementation may support an execution environment in which one or more of the functions described below cannot be implemented. [10]

[2] An implementation may chose to implement some or all of the functions described below as macros.[11] Such an implementation will still be conforming.

[3] A conforming implementation may support an execution environment in which the access type parameter adr in one or more of the functions below cannot be used as a variable. [12]

[4] The I/O buffer functions described in X.3.5 to X.3.8 are intended to be used to access register elements as a linear array of I/O registers. An implementation should assume that all I/O register elements in such a buffer have the same width in number of bits.
Given two elements [n] and [n+1] in an I/O register buffer then the address of element [n] compares less than the address of element [n+1].

[5] If the width of the databus is larger than the width of an I/O buffer register element, then an implementation should assume that all register elements in the buffer are using the same part of the databus.

---

[10] A typical implementation is not expected to have I/O functions for register sizes larger than what is supported by the processor architecture.

[12] An implementation can used macros in stead of functions in order to generate efficient in-line code.

[11] Some processor architectures may require that the access type parameter is defined as a constant at compile time.

Example
An I/O chip with a 8 bit databus is connected to the LSB part of a 16 bit databus from a processor which has a byte aligned address range. The I/O chip contains a I/O register buffer with 3 elements of 8 bits.

```
xxxx xxxx <0000 0000>   Register element 0 at address + 0
xxxx xxxx <1111 1111>   Register element 1 at address + 2
xxxx xxxx <2222 2222>   Register element 2 at address + 4
```

[6] An I/O function implementation should not handle limitations which originate from limitations in the physical I/O hardware. [13]

Example
AND and OR operations are not permitted on the register types write-only and read-only. Such limitations, which originate from the physical I/O hardware registers, should not be handled nor masked by an implementation of the I/O functions. The handling of such limitations should be done by the driver function not the I/O functions.

X.3 I/O Functions

X.3.1 I/O register read functions

X.3.1.1 The iord1 function

Synopsis
[1]

```
#include <inttypes.h>
#include <iohw.h>
bool iord1( access1 adr );
```

Description

[2] The iord1 function reads a value from an 1-bit register whose physical location is defined by the argument adr.

Returns

[3] The iord1 function returns the value read from the 1-bit register.

---

[13] The purpose of the I/O functions is to hide processor architecture differences, not the characteristics of the I/O hardware.

X.3.1.2 The iord8 function

Synopsis

[1]

```
#include <inttypes.h>
#include <iohw.h>
uint_least8_t iord8( access8 adr );
```

Description

[2] The iord8 function reads a value from an 8-bit register whose physical location is defined by the argument adr.

Returns

[3] The iord8 function returns the value read from the 8-bit register.

X.3.1.3 The iord16 function

Synopsis

[1]

```
#include <inttypes.h>
#include <iohw.h>
uint_least16_t iord16( access16 adr );
```

Description

[2] The iord16 function reads a value from a 16-bit register whose physical location is defined by the argument adr.

Returns

[3] The iord16 function returns the value read from the 16-bit register.

X.3.1.4 The iord32 function

Synopsis

[1]

```
#include <inttypes.h>
#include <iohw.h>
uint_least32_t iord32( access32 adr );
```

Description

[2] The iord32 function reads a value from a 32-bit register whose physical location is defined by the argument adr.

Returns

[3] The iord32 function returns the value read from the 32-bit register.

X.3.1.5 The iord64 function

Synopsis

[1]

```
#include <inttypes.h>
#include <iohw.h>
uint_least64_t iord64( access64 adr );
```

Description

[2] The iord64 function reads a value from a 64-bit register whose physical location is defined by the argument adr.

Returns

[3] The iord64 function returns the value read from the 64-bit register.

X.3.2 I/O register write functions

X.3.2.1 The iowr1 function

Synopsis

[1]

```
#include <inttypes.h>
#include <iohw.h>
void iowr1( access1 adr, bool value );
```

Description

[2] The iowr1 function writes the argument value to a 1-bit register whose physical location is defined by the argument adr.

X.3.2.2 The iowr8 function

Synopsis
[1]

```
#include <inttypes.h>
#include <iohw.h>
void iowr8( access8 adr, uint_least8_t value );
```

Description

[2] The iowr8 function writes the argument value to an 8-bit register whose physical location is defined by the argument adr.

X.3.2.3 The iowr16 function

Synopsis
[1]

```
#include <inttypes.h>
#include <iohw.h>
void iowr16( access16 adr, uint_least16_t value );
```

Description

[2] The iowr16 function writes the argument value to a 16-bit register whose physical location is defined by the argument adr.

X.3.2.4 The iowr32 function

Synopsis
[1]

```
#include <inttypes.h>
#include <iohw.h>
void iowr32( access32 adr, uint_least32_t value );
```

Description

[2] The iowr32 function writes the argument value to a 32-bit register whose physical location is defined by the argument adr.

X.3.2.5 The iowr64 function

Synopsis
[1]

```
#include <inttypes.h>
#include <iohw.h>
void iowr64( access64 adr, uint_least64_t value );
```

Description

[2] The iowr64 function writes the argument value to a 64-bit register whose physical location is defined by the argument adr.

X.3.3 I/O register AND functions

X.3.3.1 The ioand1 function

Synopsis
[1]

```
#include <inttypes.h>
#include <iohw.h>
void ioand1( access1 adr, bool value );
```

Description

[2] The ioand1 function performs a bit-wise AND of the argument value and a 1-bit register whose physical location is defined by the argument adr. The result of the operation is stored in the register.

X.3.3.2 The ioand8 function

Synopsis
[1]

```
#include <inttypes.h>
#include <iohw.h>
void ioand8( access8 adr, uint_least8_t value );
```

Description

[2] The ioand8 function performs a bit-wise AND of the argument value and an 8-bit register whose physical location is defined by the argument adr. The result of the operation is stored in the register.

X.3.3.3 The ioand16 function

Synopsis
[1]

```
#include <inttypes.h>
#include <iohw.h>
void ioand16( access16 adr, uint_least16_t value );
```

Description

[2] The ioand16 function performs a bit-wise AND of the argument value and a 16-bit register whose physical location is defined by the argument adr. The result of the operation is stored in the register.

X.3.3.4 The ioand32 function

Synopsis

[1]

```
#include <inttypes.h>
#include <iohw.h>
void ioand32( access32 adr, uint_least32_t value );
```

Description

[2] The ioand32 function performs a bit-wise AND of the argument value and a 32-bit register whose physical location is defined by the argument adr. The result of the operation is stored in the register.

X.3.3.5 The ioand64 function

Synopsis

[1]

```
#include <inttypes.h>
#include <iohw.h>
void ioand64( access64 adr, uint_least64_t value );
```

Description

[2] The ioand64 function performs a bit-wise AND of the argument value and a 64-bit register whose physical location is defined by the argument adr. The result of the operation is stored in the register.

X.3.4 I/O register OR functions

X.3.4.1 The ioor1 function

Synopsis

[1]

```
#include <inttypes.h>
#include <iohw.h>
void ioor1( access1 adr, bool value );
```

Description

[2] The ioor1 function performs a bit-wise OR of the argument value and a 1-bit register whose physical location is defined by the argument adr. The result of the operation is stored in the register.

X.3.4.2 The ioor8 function

Synopsis

[1]

```
#include <inttypes.h>
#include <iohw.h>
void ioor8( access8 adr, uint_least8_t value );
```

Description

[2] The ioor8 function performs a bit-wise OR of the argument value and an 8-bit register whose physical location is defined by the argument adr. The result of the operation is stored in the register.

X.3.4.3 The ioor16 function

Synopsis

[1]

```
#include <inttypes.h>
#include <iohw.h>
void ioor16( access16 adr, uint_least16_t value );
```

Description

[2] The ioor16 function performs a bit-wise OR of the argument value and a 16-bit register whose physical location is defined by the argument adr. The result of the operation is stored in the register.

X.3.4.4 The ioor32 function

Synopsis

[1]

```
#include <inttypes.h>
#include <iohw.h>
void ioor32( access32 adr, uint_least32_t value );
```

Description

[2] The ioor32 function performs a bit-wise OR of the argument value and a 32-bit register whose physical location is defined by the argument adr. The result of the operation is stored in the register.

X.3.4.5 The ioor64 function

Synopsis

[1]

```
#include <inttypes.h>
#include <iohw.h>
void ioor64( access64 adr, uint_least64_t value );
```

Description

[2] The ioor64 function performs a bit-wise OR of the argument value and a 64-bit register whose physical location is defined by the argument adr. The result of the operation is stored in the register.

X.3.5 I/O buffer read functions

X.3.5.1 The iordbuf8 function

Synopsis

[1]

```
#include <inttypes.h>
#include <iohw.h>
uint_least8_t iordbuf8( access8 adr, unsigned int index );
```

Description

[2] The iordbuf8 function reads the value of the [index] element in a hardware buffer arranged as a one-dimensional array of 8-bit registers. The physical location of element [0] is defined by the argument adr.

Returns

[3] The iordbuf8 function returns the value read from the 8-bit register.

X.3.5.2 The iordbuf16 function

Synopsis

[1]

```
#include <inttypes.h>
#include <iohw.h>
uint_least16_t iordbuf16( access16 adr, unsigned int index );
```

Description

[2] The iordbuf16 function reads the value of the [index] element in a hardware buffer arranged as a one-dimensional array of 16-bit registers. The physical location of element [0] is defined by the argument adr.

Returns

[3] The iordbuf16 function returns the value read from the 16-bit register.

X.3.5.3 The iordbuf32 function

Synopsis
[1]

```
#include <inttypes.h>
#include <iohw.h>
uint_least32_t iordbuf32( access32 adr, unsigned int index );
```

Description

[2] The iordbuf32 function reads the value of the [index] element in a hardware buffer arranged as a one-dimensional array of 32-bit registers. The physical location of element [0] is defined by the argument adr.

Returns

[3] The iordbuf32 function returns the value read from the 32-bit register.

X.3.5.4 The iordbuf64 function

Synopsis
[1]

```
#include <inttypes.h>
#include <iohw.h>
uint_least64_t iordbuf64( access64 adr, unsigned int index );
```

Description

[2] The iordbuf64 function reads the value of the [index] element in a hardware buffer arranged as a one-dimensional array of 64-bit registers. The physical location of element [0] is defined by the argument adr.

Returns

[3] The iordbuf64 function returns the value read from the 64-bit register.

X.3.6 I/O buffer write functions

X.3.6.1 The iowrbuf8 function

Synopsis
[1]

```
#include <inttypes.h>
#include <iohw.h>
void iowr8( access8 adr, unsigned int index, uint_least8_t value );
```

Description

[2] The iowrbuf8 function writes value to the [index] element in a hardware buffer arranged as a one-dimensional array of 8-bit registers. The physical location of element [0] is defined by the argument adr.

X.3.6.2 The iowrbuf16 function

Synopsis
[1]

```
#include <inttypes.h>
#include <iohw.h>
void iowr16( access16 adr, unsigned int index, uint_least16_t value );
```

Description

[2] The iowrbuf16 function writes value to the [index] element in a hardware buffer arranged as a one-dimensional array of 16-bit registers. The physical location of element [0] is defined by the argument adr.

X.3.6.3 The iowrbuf32 function

Synopsis
[1]

```
#include <inttypes.h>
#include <iohw.h>
void iowr32( access32 adr, unsigned int index, uint_least32_t value );
```

Description

[2] The iowrbuf32 function writes value to the [index] element in a hardware buffer arranged as a one-dimensional array of 32-bit registers. The physical location of element [0] is defined by the argument adr.

X.3.6.4 The iowrbuf64 function

Synopsis

[1]

```
#include <inttypes.h>
#include <iohw.h>
void iowr64( access64 adr, unsigned int index, uint_least64_t value );
```

Description

[2] The iowrbuf64 function writes value to the [index] element in a hardware buffer arranged as a one-dimensional array of 64-bit registers. The physical location of element [0] is defined by the argument adr.

X.3.7 I/O buffer AND functions

X.3.7.1 The ioandbuf8 function

Synopsis

[1]

```
#include <inttypes.h>
#include <iohw.h>
void ioandbuf8( access8 adr, unsigned int index, uint_least8_t value );
```

Description

[2] The ioandbuf8 function performs a bit-wise AND of the argument value and the [index] element in a hardware buffer arranged as a one-dimensional array of 8-bit registers. The result of the operation is stored in the [index] register. The physical location of element [0] is defined by the argument adr.

X.3.7.2 The ioandbuf16 function

Synopsis

[1]

```
#include <inttypes.h>
#include <iohw.h>
void ioandbuf16( access16 adr, unsigned int index, uint_least16_t value );
```

Description

[2] The ioandbuf16 function performs a bit-wise AND of the argument value and the [index] element in a hardware buffer arranged as a one-dimensional array of 16-bit registers. The result of the operation is stored in the [index] register. The physical location of element [0] is defined by the argument adr.

X.3.7.3 The ioandbuf32 function

Synopsis
[1]
```
 #include <inttypes.h>
 #include <iohw.h>
 void ioandbuf32( access32 adr, unsigned int index, uint_least32_t value );
```

Description

[2] The ioandbuf32 function performs a bit-wise AND of the argument value and the [index] element in a hardware buffer arranged as a one-dimensional array of 32-bit registers. The result of the operation is stored in the [index] register. The physical location of element [0] is defined by the argument adr.

X.3.7.4 The ioandbuf64 function

Synopsis
[1]
```
 #include <inttypes.h>
 #include <iohw.h>
 void ioandbuf64( access64 adr, unsigned int index, uint_least64_t value );
```

Description

[2] The ioandbuf64 function performs a bit-wise AND of the argument value and the [index] element in a hardware buffer arranged as a one-dimensional array of 64-bit registers. The result of the operation is stored in the [index] register. The physical location of element [0] is defined by the argument adr.

X.3.8 I/O buffer OR functions

X.3.8.1 The ioorbuf8 function

Synopsis
[1]
```
   #include <inttypes.h>
   #include <iohw.h>
   void ioorbuf8( access8 adr, unsigned int index, uint_least8_t value );
```

Description

[2] The ioorbuf8 function performs a bit-wise OR of the argument value and the [index] element in a hardware buffer arranged as a one-dimensional array of 8-bit registers. The result of the operation is stored in the [index] register. The physical location of element [0] is defined by the argument adr.

X.3.8.2 The ioorbuf16 function

Synopsis
[1]
```
#include <inttypes.h>
#include <iohw.h>
void ioorbuf16( access16 adr, unsigned int index, uint_least16_t value );
```

Description

[2] The ioorbuf16 function performs a bit-wise OR of the argument value and the [index] element in a hardware buffer arranged as a one-dimensional array of 16-bit registers. The result of the operation is stored in the [index] register. The physical location of element [0] is defined by the argument adr.

X.3.8.3 The ioorbuf32 function

Synopsis
[1]
```
#include <inttypes.h>
#include <iohw.h>
void ioorbuf32( access32 adr, unsigned int index, uint_least32_t value );
```

Description

[2] The ioorbuf32 function performs a bit-wise OR of the argument value and the [index] element in a hardware buffer arranged as a one-dimensional array of 32-bit registers. The result of the operation is stored in the [index] register. The physical location of element [0] is defined by the argument adr.

X.3.8.4 The ioorbuf64 function

Synopsis
[1]
```
#include <inttypes.h>
#include <iohw.h>
void ioorbuf64( access64 adr, unsigned int index, uint_least64_t value );
```

Description

[2] The ioorbuf64 function performs a bit-wise OR of the argument value and the [index] element in a hardware buffer arranged as a one-dimensional array of 64-bit registers. The result of the operation is stored in the [index] register. The physical location of element [0] is defined by the argument adr.

X.4 Constraints

X.4.1 Atomic operations

[1] The I/O functions do not guarantee that read-modify-write operations will be atomic.

Example:
[2] For some processor types the statement:

```
ioand8( PORT3, 0xf3 );
```

may be implemented as separate machine operations; for example:

```
in temp,PORT3;
and temp,0xf3;
out PORT3,temp;
```

X.4.2 Compiler optimization

[1] The I/O functions do not guarantee two ioxxxx operations will always be two separate operations. Two operations in a statement are allowed to be combined to a single read-modify-write operation by the translator.

Example:
[2] The statement:

```
iowr8(PORT1, iord8(PORT1) + 0x10);
```

can be optimized by the translator to a single machine instruction; for example:

```
ADD   PORT1,0x10
```