Disposition of Comments received during PDTR 18037 ballot (SC22 N3608)

Japan
=====
Comment J-1:
This PDTR does not have the formal cover letter on which the information of the ballot (e.g. the type of the TR) must be described. It should be accompanied with the letter ballot.

Response J-1: Accpeted.

Netherlands
===========
Comment NL-1:
Section 7.18a.3 (page 27), 3rd para: replace "The values given below" by "The integer values given below". Rationale: the current text requires the pre-processor to be able to do fixed-point arithmetic; this was never the intention.

Response NL-1: accepted.

Comment NL-2:
Change the title of 7.18a.6 (page 34) from "The fixed-point intrinsic functions" to "Fixed-point types <stdfix.h>"
In any case: remove the word "intrinsic"

Response NL-2: Accepted; title will be "Fixed-point arithmetic <stdfix.h>".

Comment NL-3:
In the synopsis of 7.18a.6.1 add

```
   #include <stdfix.h>
```

Response NL-3: Accepted.

UK
==
Comment UK-1:
Generally that we should not build on the maths of C99 until the current problems in the standard are fixed.

Note that the majority who were against this PDTR work in the embedded field.

Response UK-1: Within the scope of the project under ballot, this comment cannot be addressed.

USA
===
Comment US-1:
p.27, 2.2/7.18a.2: "If there is no ..." points out a glaring need for test macros so that applications can determine when there will be a problem using such a typedef. HAS_INT_R_T for example, with no requirement to provide a typedef when the corresponding macro is not defined.

Response US-1: Rejected.
In common practice this is not really a problem. If necessary, the required effect can be achieved using the bitsize macro's (ULACCUM_IBIT, ULACCUM_FBIT etc.).

Comment US-2:
7.1.3 (overflow and rounding): Is there any relationship between the rounding done with floating-point numbers and the rounding done with fixed-point numbers? If they are independent, is there a way to determine and/or alter the fixed-point rounding method (similar to FLT_ROUNDS or fesetround())? Is the rounding method for fixed-point static or dynamic?

Response US-2: Rejected.
Because fixed-point and floating-point operations are usually handled by different hardware units, rounding behaviours are treated independently. The rounding mode is explictly left implementation-defined, with no prescribed manner to determine and/or alter the rounding mode (see 2.1.3 last paragraph and 2.2/5.2.4.2.3, penultimate para).

Comment US-3:
7.18a.6.8 (strto*): What is the order between rounding and negating? To match the spirit of IEEE-754, negating should come before rounding. What is "correctly rounded" for conversions from decimal to fixed-point numbers?
Currently, that term is only defined for floating-point numbers. Does it have the same meaning?

Response US-3: Partially accepted.
The order is as described. The usage of the term "correctly rounded" in 7.18a.6.8 is incorrect; sentence will be changed to "... rounded as necessary in an implementation-defined manner".

Comment US-4:
p.5, Introduction para.4: Sentence beginning "In order to allow" should be adjoined, with a comma, to the following sentence.

Response US-4: Accepted.

Comment US-5:
p.6, 1.1 Scope para.2: "standard, necessary" should be "standard necessary".

Response US-5: Accepted.

Comment US-6:
p.7, 1.3 Conformance: "free standing" should be "freestanding".

Response US-6: Accepted.
Also: C.6, paragraph 8: "free-standing" should be "freestanding".

Comment US-7:
pp.9-10, 2.1.2 Spelling ..., the requirement or lack thereof for aliases vs. the keywords is not clear; in particular, "redefined" raises questions and "or to another spelling" suggests that the natural spelling might not be defined.
Suggest "... <stdfix.h>, these formal names are used to define the natural spellings as aliases, and may be used to define other spellings, for instance ..." (While I disagree with the notion that a standard header should define unspecified names not in an implementation-reserved namespace, apparently this was already deliberately decided upon. It is not reflected in the normative wording, however!)

Response US-7: Accepted (without the last parenthetical part).

```
Comment US-8:
p.10, 2.1.3 Overflow ..., last line: insert "the" before "correct".
```

Response US-8: Accepted.

```
Comment US-9:
p.13, 2.1.6.2.1 para.2: There is an error in line filling (the second sentence
should start right after the first).
```

Response US-9: Accepted
This happens in more places (sometimes not visible); see for instance 2.1.3, 1st para. Changed in 2.1.3 and 2.1.6.2.1.

```
Comment US-10:
p.14, 2.1.6.2.1 para.3: Another line filling error; also, "deprecate" is
misspelled.
```

Response US-10: Filling error: not accepted (a Note belonging to the 1st part of the paragraph); typo: accepted.

```
Comment US-11:
p.14, 2.1.6.2.1 para.5: "perform a multiply of" should be "multiply" and the
following "and" would be better as "by".
```

Response US-11: Accepted.

```
Comment US-12:
p.14, 2.1.6.2.2: Change "type int" to "integer type" to match the actual
normative text.
```

Response US-12: Accepted.

```
Comment US-13a:
p.15, 2.1.6.4 Example: The braces { } are unnecessary.
```

Response US-13a: Accepted.

```
Comment US-13b:
p.18, 2.2 Detailed changes ...: "will get in the new document" should be "may
get in the new document".  It is possible that additional parent-section
insertions change "mm", for example. (Also p.47, 3.3.)
```

Response US-13b: Accepted in principle: the text starting with "and hence will get ..." will be removed; this also applies to section 3.3, p47.

```
Comment US-14:
p.18, 2.2/5.2.4.2.3 para.3: The radix "dot" is assumed to be between... what?
Between implies two things, not just the one most-significant-digit.  Perhaps
"between or" should be deleted.
```

Response US-14: Accepted in principle.
Change "between or" to "somewhere between the most significant digit and the least significant digit in the nominator, or"

Comment US-15:
p.19, 2.2/5.2.4.2.3 para.4: Another line filling error, or perhaps this should be two paragraphs.

Response US-15: Accepted: will be two paragraphs.

Comment US-16a:
p.19, 2.2/5.2.4.2.3 para.5: "exact" should be "exactly".

Response US-16a: Accepted.
Sentence will read "... the operation is performed on the operand values according to the operation's usual mathematical definition, ...".

Comment US-16b:
p.19, 2.2/5.2.4.2.3 para.6: "maximal" should be "most" (three occurrences). "maximal negative" is simply wrong.

Response US-16b: Accepted.

Comment US-17:
p.20, 2.2/6.2.5 paras.3-4: Suggest moving "_Sat" just before "_Fract" in each type, for consistency with other uses.

Response US-17: Accepted in principle: the text in p25-26, 2.2/6.7.2 will be changed to be consistent with 2.2/6.2.5.

Comment US-18:
p.21, 2.2/6.2.5 para.4: The last use of "accum" should be italicized.

Response US-18: Accepted.

Comment US-19:
p.21, 2.2/6.2.6.3 paras.1-2: "values of any padding bits" should be "contents of any padding bits". (The term "value" as used in the C standard must carefully exclude padding.)

Response US-19: Rejected.
This is exactly the same as the last sentence of 6.2.6.2 para 1.

Comment US-20:
p.22, 2.2/6.2.6.3 para. 4 first item: "bits or" should be "bits as or" (optional commas around alternative).

Response US-20: Accepted: "bits as, or ..."

Comment US-21:
p.22, 2.2/6.2.6.3 para. 5: Delete "where practical", which adds nothing and raises an unanswered question.

Response US-21: Accepted.

Comment US-22:
p.24, 2.2/6.4.4.2a Syntax: second form for decimal-fixed- constant should have "opt" suffix on exponent-part; otherwise 1k (for example) is not a valid fixed-constant. This also aligns with the strto* functions.

Response US-22: Accepted.

Comment US-23:
p.28, 2.2/7.18a.3 list: Expressions such as (-0.5HR-0.5HR) overflow and thus have undefined behavior. The true required minimum is something like (-0.9921875HR).

Response US-23: Rejected: all the signed fract types have -1 as a 'normal' value.

Comment US-24:
p.28, 2.2/7.18a.3 *FRACT_MAX, *ACCUM_MAX: The second (hex) form contains a spurious "C". Suggest deleting all these (redundant) forms, or correcting them and deleting the first forms.

Response US-24: Rejected.
The 'C's are significant as the hex character defining the value of the 2 least significant bits; take SFRACT_MAX as example: this is a 7-bit fract; the max value thus should have 7 bits set to 1; 0x1.F only describes 5 bits (0x1.FP-1 equals in decimal notation 0.96875, which is less than the required 0.9921875).
The repetition of the constants in a decimal and a hex form also occurs in 5.2.4.2.2 para 13.
Editors note: in 2.2/7.18a.3 on page 32: 2nd occurrence of LACCUM_FBIT should be LACCUM_IBIT; likewise 2nd occurrence of ULACCUM_FBIT should be ULACCUM_IBIT.

Comment US-25:
p.33, 2.2/7.18a.4 Description: "ulp" is not defined in the changes to the standard. Suggest using small caps ("ULP", "ULPs") and defining it here (as in the footnote to 2.1.3).

Response US-25: Accepted.

Comment US-26:
p.33, 2.2/7.18a.4 Description: Change "multiply and divide" to "multiplication and division". Nouns, not verbs.

Response US-26: Accepted.

Comment US-27:
p.33, 2.2/7.18a.5 Description: "according to the set state" is unclear. Should this be "saturating"?

Response US-27: Accepted; it should be "saturation".

Comment US-28:
p.35, 2.2/7.18a.6.1 Returns: "saturated" seems wrong when there's no overflow. Suggest "saturated if it would overflow"?

Response US-28: Accepted ("saturated on overflow").

Comment US-29:
p.35, 2.2/7.18a6.3: Suggest these be called "rounding" functions rather than "round" functions (also in 2.2/7.18a.6.7). Under "Returns", change "The functions" to "The round[ing] functions". If "round" is retained, the latter should be in Courier as used in the countls Returns.

Response US-29: Accepted (cf 2.1.7.2).

Comment US-30:
p.38, 2.2/7.18a.6.6: Change "The bits functions" to "The above functions". (There are other functions whose names contain "bits".)

Response US-30: Assuming this refers to 2.2/7.18a.6.5 (both on p.37 and p.38): accepted.

Comment US-31:
p.38, 2.2/7.18a.6.5 Returns: Change "bitpattern" to "bit pattern".

Response US-31: Accepted.

Comment US-32:
p.41, 2.2/7.19.6.1 'h': Should be insertion before the last semicolon, as specified for 'l'.

Response US-32: Accepted.

Comment US-33:
p.41, 2.2/7.19.6.1 'l': Change "semi-colon" to "semicolon".

Response US-33: Accepted.

Comment US-34:
p.41, 2.2/7.19.6.2 'h': Inserted text should start with "or".

Response US-34: Accepted.

Comment US-35:
p.42, 2.2/7.19.6.2 r,R,k,K: Should be a comma before last "or".

Response US-35: Accepted.

Comment US-36:
p.45, 3.1.3 para.4, last sentence: Change "any" to "a null".

Response US-36: Accepted; change "any pointer into an" to "a null pointer into any".

Comment US-37:
p.46, 3.2.2 para.1: Change "pre-defined" to "predefined".

Response US-37: Accepted (also in 3.1.2, para 1).

Comment US-38:
p.46, 3.2.2 Examples: Make first occurrence of "char" Courier.

Response US-38: Rejected; however, as the notion 'char-sized' could be misleading and is not necessary, it is removed.

Comment US-39:
p.48, 3.3/6.2.4a para.1: Append "Unless otherwise specified, objects are allocated in the generic address space." (Even though this is covered in changes to 6.2.5, it needs to be emphasized here.)

Response US-39: Accepted.

Comment US-40:
p.49, 3.3/6.2.5 para 26, para.1: Last sentence should have appended "and address space qualifiers".

Response US-40: Rejected.
The definition of "additionally access-qualified" is correct as written. If type T is an additionally access-qualified version of type S, then T will have whatever address space qualifier S has (if any), plus some number (possibly zero) of added access qualifiers, "const", "volatile", and "restrict". Where the term "additionally access-qualified" is used, it is important that T and S be in the same address space. If the proposed change were adopted, T could be in an address space different from S.
Added to the end of the sentence: "(if type T is an additionally access-qualified version of type S, then T and S are in the same address space)".

Comment US-41:
p.52, 3.3/6.5.16.1: Change "referenced type of" to "type pointed to by" (four occurrences altogether). "Referenced address space" can remain, but only because there is no better way to say that.

Response US-41: Rejected; the term "referenced type" is introduced in the C standard in 6.2.5 para 20; "referenced address space" is introduced in the new text replacing 6.2.5 para 26.

Comment US-42:
p.52, 3.3/6.7.1 syntax: Also add:

```
    register-name: identifier
```

Response US-42: Accepted.

Comment US-43:
p.53, 3.3/6.7.1.1 para.1: Change "Modifying" to "Accessing". (Read access might also have side effects.)

Response US-43: Accepted.

Comment US-44:
p.53, 3.3/6.7.2.1: Seems wrong; the members get qualified by the address-space qualifier of the aggregate. Suggest deleting this. (Unnecessary due to other requirements.)

Response US-44: Rejected.
However, change text to "The *specifier-qualifier-list* in the declaration of a member of a structure or union shall not include an address space qualifier."

Comment US-45:
p.53, 3.3/6.7.3 syntax: Also add:

```
   address-space-name: identifier
```

Response US-45: Accepted.

Comment US-46:
p.54, 4.1.1 first bullet: Change "market place" to "marketplace".

Response US-46: Accepted.

Comment US-47:

p.55, 4.2: Too many italics. I suggest changing all italics in this section to normal font, then change all bold to italic, which is consistent with usage in the C standard.

Response US-47: Rejected; the intend with the italic in the terminology definitions is to 'keep the words together' in a muli-word term, in order to make it easier for a first time reader to learn the terminology, and to emphasis that these words are terms defined somewhere in 4.2; the text is kept as is for pedagogical reasons (after all this is a TR not a standard).

Comment US-48:
p.55, 4.2: The bullet "Multiple I/O registers may form an I/O group." is redundant with the next bullet and should be removed.

Response US-48: Accepted.

Comment US-49:
p.56, 4.3.1, last line: Change "interleave" to "interleaving".

Response US-49: Accepted.

Comment US-50:
p.57, 4.3.1: Change "<->" to a double-headed arrow glyph.

Response US-50: Accepted.

Comment US-51:
p.57, 4.3.2 figure: Change "users" to "user's" and "vendors" to "vendor's".

Response US-51: Accepted.

Comment US-52:
p.59, 4.4.2: Should use so-called "smart quotes" around "function"; save the "straight double quote" for C code. (Also around "dense" on p.61, 4.4.3, "kind" on p.62, 4.4.5, "acquiring" on p.62, 4.4.6.1, "<iohw.h>" and "function on p.64, 4.5/7.8a, and possibly elsewhere.)
This might be present in the PDF document, but was not evident in the selected font.

Response US-52: rejected in principle: the selected type font Helvetica does not seem to support smart quotes.

Comment US-53:
Annex A? Rationale for 6.2.6.3 should explain why only sign-magnitude representation is allowed. One would think that ones- or twos-complement adders would be faster.

Response US-53: Noted.
To avoid misunderstanding, add a new paragraph at the end of A.1.1.1:

> **Most modern hardware uses two's complement arithmetic for integers. The hardware implementation of fixed-point arithmetic uses the same mechanisms and representations as the integer implementation on that same hardware. Hence, although the C standard allows other types of implementations for integers, this Technical Report requires exclusively two's complement behaviour for fixed-point arithmetic.**

Comment US-54:

Annex A? Rationale for 6.2.6.3 should explain why a signed accum type has to have at least as many integral bits as the corresponding unsigned accum type. One would think that it could reasonably be one fewer.

Response US-54: Accepted.
Change title of A.2 to "Number of data bits in _Fract versus _Accum" and add new paragraph at the end of that section:

> **A signed accum type has to have at least as many integral bits as the corresponding unsigned accum type because the usual arithmetic conversions prescribe that, when signed and unsigned fixed-point types are mixed, the unsigned type is converted to the corresponding signed type, and this should go without loss of magnitude; note also that the notion 'integral bits' does not include the sign bit.**

Comment US-55:
Annex C, general: "Interleave" is a verb, not a noun or adjective. Use "interleaving" for the latter. (Several occurrences.)

Response US-55: Accepted.

Comment US-56:
p.96, D.2.5 title: Change "users" to "user's".

Response US-56: Accepted.

Comment US-57:
p.99, E.4: Change "2's complement" to "twos-complement".

Response US-57: Accepted.

Switzerland
===========
These comments are the result of an attempt to implement the different versions of the chapter 4 "Basic I/O Hardware Addressing" of the TR since the Copenhagen meeting in 2001. This chapter has changed considerably in the course of its development, and major modifications were introduced between the first and the second SC22 ballot.

Generally, these modifications are very positive and make the background and interface usage of I/O HW addressing much clearer.

But unfortunately, some specific changes in interface definitions make the usefulness as well as the efficient implementability of the interface very problematic. Actually, while an efficient implementation of interface from the TR of the first ballot was successfully finished, it turned out as being essentially impossible to create an efficient implementation of the interface as specified in the TR for the second ballot.

This does not say that we object the whole restructuring of chapter 4. But some specific modifications changed the actual substance of the hardware addressing interface without any given plausible technical
rationale.

Comment CH-1:
The proposed 7.8a.1p1 (p.64) states:
  "An I/O register is accessed (read or written) as an unsigned integer."

This can be misleading. There might be no unsigned integer type that can accomodate the value of an I/O register. I.e. though the underlying register is generally indeed treated as an unsigned interger value, the actual C data type to hold it might be something different.

E.g. if the register is 128 bit wide, but the largest unsigned integer type available holds only 64 bit, one needs a struct to hold the value. (Arrays are generally also possible but problematic as they can not be passed by value.)

Or if the register is 24 bit wide, but there is no unsigned integer type of that width, one might also want to hold the value in a struct to avoid unnecessary conversions or paddings.

For this issue, we propose the following resolution:
Write the above sentence as:

  "An I/O register is accessed (read or written) as an unsigned integer
   value[1]."
  "Note 1: This does not necessarily imply that the type used is actually
   one of the set of unsigned integer types provided by the compiler."

Response CH-1: Accepted with modification.
The addition of the word *value* is not strictly necessary, as the C Standard usually (but not always) makes a distinction between "integers" (as mathematical concepts) and "integer types".
There are existing instances in the C Standard where the term *integer* is used to refer to a value that may not be representable in any of the integer types.
To address the spirit of the comment, the sentence will be expanded as follows: "An I/O register is accessed (read or written) as an unsigned integer whose representation (in the register) is implementation-defined and need not conform to any available integer type."

Comment CH-2:
The proposed 7.8a.4 (p.67) defines the register access interface as functions. Though the proposed introduction to 7.8a (p.64) and also 4.4.2 (p.59) allow the implementation as macros, 4.4.2 explicitely requires that a macro-based implementation provides exactly the same effects as the function implementation.

This is a major difference to the previous version that generally defined the interface as macros and allowed the implementation to use (possibly a special kind of) functions. The problematic difference is the definition and handling of parameter types. (Of course, this discussion applies to return types as well.)

Macros allow any type as arguments, even different argument types for the same macro, and take the arguments as given.
But functions are defined to take exactly one argument type for each parameter, and convert any other argument types to the parameter type. Even worse (in this specific case), functions require integer type arguments at least converted to (unsigned) int, i.e. there is no use in defining functions with parameter types of unsigned char or unsigned short. So, an 8-bit value is always converted to an integer at least once for each read and each write, which causes an overhead that is always annoying and sometimes forbidding. Though in theory a good optimizer could remove any unnecessary conversions, in practice this will rarely be the case, as the implementation of these operations typically implies the usage of specific assembler instructions that are generally untouched by optimizers.

Furthermore, the proposed interface functions are provided as int and long versions, so it's not possible to use these functions for I/O register values larger than an unsigned long.

For this issue, we propose the following resolution:

In the proposed 7.8a.4, all the function definitions are replaced by respective macro definitions, e.g. 7.8a.4.1:

replace:
Synopsis
```
   #include <iohw.h>
   unsigned int iord( ioreg_designator );
   unsigned long iordl( ioreg_designator );
```

Description
The functions iord and iordl read the individual I/O register referred to by ioreg_designator and return the value read. The I/O register is read as an unsigned integer of its size; the read value is then converted to the result type, and this converted value is returned.

by:
Synopsis
```
   #include <iohw.h>
   iord( ioreg_designator )
```

Description
The function like makro iord reads the individual I/O register referred to by ioreg_designator and returns the value read. The I/O register is read as an unsigned integer of its size; this value is returned without any conversion.

or 7.8a.4.3:

replace:
Synopsis
```
   #include <iohw.h>
   void iowr( ioreg_designator, unsigned int a );
   void iowrl( ioreg_designator, unsigned long a );
```

Description
The functions iowr and iowrl write the individual I/O register referred to by ioreg_designator. The unsigned integer a is converted to an unsigned integer of the size of the I/O register, and this converted value is written to the I/O register.

by:
Synopsis
```
   #include <iohw.h>
   iowr( ioreg_designator, a )
```

Description
The function like macro iowr writes the individual I/O register referred to by ioreg_designator. If the unsigned integer value 'a' is of the same size as the size of the register, it is written to the I/O register without any conversion. If 'a' is of a different size than the size of the register, it is converted to an unsigned integer value of the size of the I/O register, and this converted value is written to the I/O register. iowr does not return anything.

Response CH-2: Rejected.
An implementation is free to provide specialized macro's or functions for specific register sizes; the proposal precludes the possibility of an implementation using functions. For more discussion on this topic see the discussion on the embedded-c reflector (for instance http://www.dkuug.dk/jtc1/sc22/wg14/embedded-c/150).

Comment CH-3:
All functions in the proposed section 7.8a define a parameter "iogroup_designator" or "ioreg_designator". The types for these parameters are correctly left undefined.

On a specific platform, these designators might have completely different types for different kinds of registers. But it is not possible (in C) to have the same function name for different types of the same positional parameter. Therefore, the designators must be unified to a common type which typically costs additional overhead. As this should be avoided, the function approach turns out to be wrong here as well.

For this issue, we propose the following resolution:

In the proposed 7.8a.3, replace the function definitions by respective macro definitions, e.g. 7.8a.3.1:

```
replace:
Synopsis
  #include <iohw.h>
  void iogroup_acquire( iogroup_designator );
  void iogroup_release( iogroup_designator );

by:
Synopsis
  #include <iohw.h>
  iogroup_acquire( iogroup_designator )
  iogroup_release( iogroup_designator )
```

Response CH-3: Rejected.
The "function approach" should not be judged "wrong" on the grounds that it involves extra overhead. For some implementations, this may be deemed an acceptable option, so such a choice should not be precluded gratuitously. Implementations using macros are already permitted and are not negatively impacted by the existing specification.

Note: An efficient (zero-overhead) implementation of the interface according to the proposed changes was successfully accomplished.

The disapproval will be changed to approval if the proposed changes are accepted.

Individual editorial notes
========================
Comment ED-1:
The word "section" should be replaced throughout the TR by the word "clause". The C standard (ISO 9899:1999) speaks not of "sections" but of "clauses".

Response ED-1: Accepted.

Comment ED-2:
The Introduction (page 5) contains some odd wording and some outright errors. (To be frank, parts of the Introduction give the impression of having been badly translated into English from a different language.) Because it's the first thing in the TR most people will read, it would be nice if it could be cleaned up to make a better first impression. Some minimal corrections and suggestions:

Change word:
"get more complex" --> "become more complex"

Add comma and change words:
"... in assembly language), and processor models for embedded systems have a decreasing lifespan ..."

Drop two "the"s:
"... re-adapting of applications to new instruction sets)."

Drop "the", add "for", change words:
"..., embedded processors are often equipped with special hardware for fixed-point data."

Replace "the" with "for":
"... does not provide support for fixed-point arithmetic operations, ..."

Change words:
"assembler" --> "assembly language"

Replace sentence "In this manner, ..." with:
"Optimizing C compilers can generate highly efficient code for fixed-point data as easily as for integer and floating-point data."

Replace sentence "Typical for the mentioned filtering ..." with:
"Many DSPs have multiple distinct banks of memory and require that data be grouped in different banks to achieve maximum performance."

Response ED-2: Accepted, with minor changes.

Comment ED-3:
1.1, paragraph 2 (page 6) says, "The second subclause of each clause contains the editorial changes to the standard, ...". This is true only for clause 2 (fixed-point), not for clauses 3 and 4.

Response ED-3: Accepted; text is adapted.

Comment ED-4:
1.1, paragraph 2 (page 6): The last sentence should be changed to: "Additional explanation and rationale are provided in the Annexes."

Response ED-4: Accepted.

Comment ED-5:
2.2, page 23, in the new text for 6.3.1.8 to be added after the conversion rule for conversion to "float": The end of the second rule would be clearer as:
"...; if either of the operands has a saturating fixed-point type, the result type shall be the saturating fixed-point type corresponding to the fixed-point type with the higher fixed-point rank."

Similarly, the end of the third rule would be clearer as:
"...; if either of the operands has a saturating fixed-point type, the result type shall be the saturating fixed-point type corresponding to the signed fixed-point type with the higher fixed-point rank."

Response ED-5: Accepted.

Comment ED-6:
4.2, page 56: The terms "static designator" and "dynamic designator" are defined but never used. The sentences that define these terms should be deleted.

Response ED-6: Rejected; these terms can be useful in related documentation.