TriCore Application Note

The saturate Type Qualifier and Integral Types

Document ID:	AN060-02
Date:	2002-03-25

A L T I U M B V

1 Introduction

When an integral type is qualified with the _sat type qualifier, all operations executed with variables of this type will be performed saturated.
When an operation is performed on a plain variable and a _sat variable, the _sat takes precedence, and the operation is done using saturating arithmetic. The type of the result of such an operation also includes the _sat qualifier, so that another operation on the result will also be saturated. In this respect, the behavior of the _sat type qualifier is comparable to the unsigned keyword.

In the remainder of this application note we give attention to the following aspects:

2. Operations not in 16 or 8 bits
3. No saturation from unsigned to signed and vice versa
4. Assignment
5. Casting
6. Integral promotions
7. Usual arithmetic conversions
8. Examples
9. Known problems

Considering the peculiar proporties of using saturation on shorter integral types and the fact that no saturation occurs from signed to unsigned and vice versa, **we advise only to use saturation on the signed int** (and on _(s)fract of course). It is possible to use saturation on unsigned int, but care should be taken not to combine this with signed saturation.
The known problems (as in § 9) still apply, even if saturation is only used for the signed int.

The following variables are used througout the examples:

```
  char  c;   unsigned char  uc;   _sat char  sc;   _sat unsigned char  suc;
  short s;   unsigned short us;   _sat short ss;   _sat unsigned short sus;
  int   i;   unsigned int   ui;   _sat int   si;   _sat unsigned int   sui;
```

There are no examples with longs, since on the Tricore an int and a long are the same.

Examples:

```
si      = 0x7FFFFFFF;
i       = 0x12345;
ui      = 0xFFFFFFFF;

si + i   // a saturated addition is performed, yielding a saturated int
si + ui  // a saturated unsigned addition is performed, yielding a saturated
         // unsigned int
i + ui   // a normal unsigned addition is performed, yielding an unsigned int
```

2 Operations not in 16 or 8 bits

The Tricore only supports 32-bits operations for integral types, conform the ANSI-C 1989 specification. Since the _sat is a type qualifier that can be used with the integral types, the saturated operations are only performed for 32-bits integers. So if either a short or a character variable is used in a saturated operation, it is first promoted and/or converted conform to the integral promotions and the usual arithmetic conversions.

Example:

```
ss = sc + ss;   // first the operands of the additon are both promoted to 32-
                // bits integers then the 32-bits saturated addition is done,
                // yielding a 32-bits integer
                // finally the 32-bits result of the addition is moved
                // saturated as a halfword into the ss variable
```

3 No saturation from unsigned to signed and vice versa

Under no circumstance a saturation from unsigned to signed or vice versa is performed. So if a cast or assignment suggests an implicit saturation from signed to unsigned or vice versa, no such thing will happen. The value is just reinterpreted as unsigned or signed. This way unexpected results may occur when the values being cast are in the region where signed and unsigned differ, the high numbers for an unsigned type and the negative numbers for an signed type.

Examples:

```
  i   = -5;
  sus = i;                  // sus == 65535
  ui  = 0xFFFFFFFF;
  s   = (_sat short)ui;     // s = -1
```

4 Assignment

In an assignment the right side is implicitly cast to the type of the left side of the assignment.
Example:

```
ss = si; // the 32-bits integer si is moved saturated as a halfword into the
         // ss variable
ss = i;  // identical, since the values of a saturated type are identical to
         // those of the unsaturated type
si = ss; // the 16-bits integer is simply moved into the si variable, no
        // saturation required
```

5 Casting

Casting between integral types of the same type, causes the bits to be interpreted differently. Next to this reinterpretation the value may also have influence on operations it is involved in. When casting to a larger integral type the current value is copied into the lower byte(s) of the target value, sign-extend if the target is signed, zero-extended if the target is unsigned. When casting to a smaller not saturated integral type the lower byte(s) are extracted from the current value and copied in the target value. When casting to a smaller saturated integral type the current value is moved saturated to the the target value, using the specific instructions the Tricore has for this purpose.
If you are performing more than one cast on the same value, the casts will be executed from right to left according to the casting rules as given. The resulting value will have the left most type.
When casting from a signed integral type to a saturated smaller unsigned integral type, an implicit cast to an unsigned integral type of the larger integral type is first applied. So if you want to apply the cast (_sat unsigned short)i implicitly the following is executed: (_sat unsigned short)(unsigned int)i.
When casting from a unsigned integral type to a saturated smaller signed integral type, an implicit cast to a signed integral type of the larger integral type is first applied. So if you want to apply the cast (_sat short)ui implicitly the following is executed: (_sat short)(int)ui.

Examples:

```
s   = -5;
us  = (unsigned short) s;                      // us == 65531
us  = ( _sat unsigned short) s;                // us == 65531
us  = (_sat unsigned int) s;                   // us == 65531
sus = (_sat unsigned short) s;                 // sus == 65531
sus = (_sat unsigned int) s;                   // sus == 65535
suc = s;                                       // suc == 255
uc  = (_sat) s;                                // uc == 255
uc  = (_sat unsigned char)(unsigned char) s;   // uc == 251
uc  = (unsigned char)(_sat unsigned char) s;   // uc == 255
ui  = 0xFFFFFFFF;
s   = (_sat short)ui;                          // s = -1
```

6 Integral promotions

When integral promotions are applied on an operand from a operation, the type of the operand may change. Saturation is always preserved when applying integral promotions, unsigned however is lost if the basic type is either char or short. All values represented by unsigned char and unsigned short are also represented in signed integer, so the represented value is always preserved when applying integral promotions. This results in the following type changes when applying integral promotions:

char	=>	int
unsigned char	=>	int
_sat char	=>	_sat int
_sat unsigned char	=>	_sat int
short	=>	int
unsigned short	=>	int
_sat short	=>	_sat int
_sat unsigned short	=>	_sat int
int	=>	int
unsigned int	=>	unsigned int
_sat int	=>	_sat int
_sat unsigned int	=>	_sat unsigned int

7 Usual arithmetic conversions

When usual arithmetic conversions are applied on the operands of an operation, both operands will be converted to the same type before executing the operation, which will yield a value of the same type as both operands have been converted to. After applying integral promotions on each of the operands, the target type will be unsigned if one of the operands is unsigned and the target type will be saturated if one of the operands is saturated.

Examples:

```
ui = ui + c     // left operand is unsigned int, right operand becomes int
                // operation is performed on, and yields, unsigned int
i = us + uc     // left operand becomes int, right operand becomes int
                // operation is performed on, and yields, int
si = sus + uc   // left operand becomes _sat int, right operand becomes int
                // operation is performed on, and yields, _sat int
ui = us + ui    // left operand becomes int, right operand is unsigned int
                // operation is performed on, and yields, unsigned int
sui = si + ui   // left operand is _sat int, right operand is unsigned int
                // operation is performed on, and yields, _sat unsigned int
```

8 Examples

Here are some a litle bit more complicated examples, with a short explanation.

Example 1:

```
s   = -5;
sus = s;        // sui == 65531
                // there is an implicit cast from short to _sat unsigned
                // short, due to the assignment, since both types are the
                // same size only a reinterpretation of the value is done,
                // no saturation is performed
i   = -5;
sus = i;        // sus == 0
                // this time both types are different, the target being the
                // smaller type, so saturation is performed
```

Example 2:

```
s  = ss + sc;   // left and right operand will both become _sat int the add
                // operation will be performed for saturated 32-bits integers
                // the result will be cast to a short again (not saturated,
                // since the left side of the assignement detemines the type)
ss = ss + sc;   // now the result will be moved saturated, since the left
                // side is now saturated.
```

9 Known problems

- assigning a too large constant in a static initialisation is truncated instead of saturated workaround: e.g. first assign constant to a large enough integer, then assign this value to the required saturated variable.

  Example:
  ```
  _sat unsigned short us = 0x12345;    // this triggers a warning W195, the
                                       // value will be truncated, not
                                       // saturated

  Workaround:
    unsigned int        ui = 0x12345;
    _sat unsigned short us = ui;       // stored using a sat.hu instruction, so
                                       // now it is saturated
  ```

- saturated operation with 2 constants, the compiler performs the operation unsaturated. workaround: switch off the constant/copy propagation optimization for the function this occurs in.

  Example:
  ```
  int test ( void )
   {
    _sat int si_1 = 0x0FFFFFFF;
    _sat int si_2 = 0x7FFFFFFF;

    return ( si_1 + si_2 ); // calculation done buildtime, unsaturated  !
   }
  ```

  Workaround:
  ```
  #pragma optimize P
  int test ( void )
   {
    _sat int si_1 = 0x0FFFFFFF;
    _sat int si_2 = 0x7FFFFFFF;

    return ( si_1 + si_2 ); // calculation done runtime, saturated
   }
  #pragma optimize restore
  ```