Fixed Point Math Library

Module user’s Guide
C24x Foundation Software
IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI’s standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer’s applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such products or services might be or are used. TI’s publication of information regarding any third party’s products or services does not constitute TI’s approval, license, warranty or endorsement thereof.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations and notices. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible or liable for any such use.

Resale of TI’s products or services with statements different from or beyond the parameters stated by TI for that products or service voids all express and any implied warranties for the associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products, www.ti.com/sc/docs/stdterms.htm

Mailing Address:
Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Copyright ©2002, Texas Instruments Incorporated
Trademarks
TMS320 is the trademark of Texas Instruments Incorporated.
All other trademark mentioned herein are property of their respective companies

Acronyms
xDAIS  : eXpress DSP Algorithm Interface Standard
IALG   : Algorithm interface defines a framework independent interface for the creation of
         algorithm instance objects
STB    : Software Test Bench
QMATH : Fixed Point Mathematical computation
CcA    : C-Callable Assembly
FIR    : Finite Impulse Response Filter
IIR    : Infinite Impulse Response Filter
FFT    : Fast Fourier Transform
Contents

1. Math Modules .............................................................................................................................................. 1
   1.1. QSIN: Fixed point SIN (Taylor series implementation) ................................................................. 1
   1.2. QSINLT: SIN (Table look-up and linear interpolation) .................................................................... 7
   1.3. QCOS: Fixed point COS (Taylor series implementation) .............................................................. 13
   1.4. QCOSLT: COS (Table look-up and linear interpolation) .............................................................. 19
   1.5. QATAN: Fixed point ATAN (Taylor series implementation) ......................................................... 25
   1.6. QSQR: Fixed point Square-root (Taylor series implementation) .................................................. 31
   1.7. QLOG10: Fixed point LOG10 (Taylor series implementation) ...................................................... 37
   1.8. QLOGN: Fixed point LOGN (Taylor series implementation) .......................................................... 45
   1.9. QINV1: Reciprocal (32-bit Precision) ............................................................................................. 53
   1.10. QINV2: Reciprocal (16-bit Precision) ........................................................................................... 57
**QSIN**

*Fixed Point SIN (Taylor Series Implementation)*

**Description**

This module computes the sine value of the input using Taylor series approximation method.

**Availability**

This module is available in two interface formats:

1) The direct-mode assembly-only interface (Direct ASM)
2) The C-callable interface version

**Module Properties**

**Type:** Target Independent, Application Independent

**Target Devices:** x24x/x24xx

**Direct ASM Version File Names:** q1sin.asm

**C-Callable Version File Names:** qsin.asm, qmath.h

<table>
<thead>
<tr>
<th>Item</th>
<th>ASM Only</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>46 words</td>
<td>48 words</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>3 words</td>
<td>0 words</td>
<td></td>
</tr>
<tr>
<td>xDAIS ready</td>
<td>No</td>
<td>Yes</td>
<td>IALG layer not implemented</td>
</tr>
<tr>
<td>xDAIS component</td>
<td>No</td>
<td>No</td>
<td></td>
</tr>
<tr>
<td>Multiple instances</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>-</td>
<td>4 words</td>
<td>Stock grows by 4 words</td>
</tr>
</tbody>
</table>
Module Terminal Variables/Functions

<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Description</th>
<th>Format</th>
<th>Range(Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>sin_input</td>
<td>Angle in radians between $[-\pi, +\pi]$ normalized to $[-1, 1]$</td>
<td>Q15</td>
<td>8000 - 7FFF</td>
</tr>
<tr>
<td>Output</td>
<td>sin_output</td>
<td>Sine value of the input</td>
<td>Q15</td>
<td>8000 – 7FFF</td>
</tr>
</tbody>
</table>

Variable Declaration:
In the system file include the following statements:

```
.ref     qsin       ; function call
.ref     sin_input, sin_output   ; input & output
```

Memory map:
All variables are mapped to an uninitialized named section “qmath”

Example: The following sample code obtains the $\sin(0.25 \times \pi) = 0.707$

**Input**
Sample input $0.25 \times \pi$ in normalized Q15 format

$$= \left( \frac{0.25 \times \pi}{\pi} \right) \times 2^{15} = 2000h$$

**Output**
The Output is $0.707$ in Q15 format $= 0.707 \times 2^{15} = 23170$ (5A82h)

Sample Code
```
.ref     qsin       ; function call reference
.ref     sin_input, sin_output   ; input & output reference
.bss    input_var1,1   ; Reserve space for input
.bss    output_var1,1   ; Reserve space for output
LDP     #input_var1    ; Store $0.25 \times \pi$ in normalized Q15
SPLK    #02000h,input_var1
LDP     #sin_input     ; Set DP for the module input
BLDD    #input_var1, sin_input   ; Pass input value to module input
CALL    qsin
LDP     #output_var1    ; Set DP for output variables
BLDD    #sin_output, output_var1   ; Pass module output to output variable
```
C/C-Callable ASM Interface

Declaration        signed short int qsin(signed short int x)

Input Format      The argument 'x' is a fixed-point number in Q15 format that contains the angle in radians between $[-\pi, +\pi]$ normalized between $[-1, +1]$

Output Format     This function returns the sine of the input argument as fixed-point number in Q15 format.

Example           The following sample code obtains the $\sin(0.25 \times \pi) = 0.707$

Input
The sample input $0.25 \times \pi$ in normalized Q15 format
\[
\left( \frac{0.25 \times \pi}{\pi} \right) \times 2^{15} = 2000h
\]

Output
The output is 0.707 in Q15 format = $0.707 \times 2^{15} = 23170$ (5A82h)

Sample Code
```c
#include<qmath.h> /* Header file for fixed point math routine */

void main(void )
{
    signed short int x,y;
    x=0x2000; /* 0.25 \times \pi \text{ in normalized Q15 format} = \left( \frac{0.25 \times \pi}{\pi} \right) \times 2^{15} = 2000h */
    y=qsin(x); /* 'y' will have $\sin\left(0.25 \times \pi\right) = 0.707$ in Q15 */
}
/* \text{0.707 in Q15 format = 0.707 \times 2^{15} = 23170 (5A82h) } */
```
Fixed Point SIN Function vs C Float SIN ($\pi < x < \pi$)

SIN(x) Obtained by C Float (Q15 format)

SIN(x) Obtained by C2000 Fixed Point Routine(Q15 Format)

Error of Fixed Point SIN(x) Routine with respect to C Float

©Texas Instruments Inc., May 2002
Background Information

Input
The range of input to the sin function is limited to \(-\pi\) to \(+\pi\) radians. The Fixed-point sine computation routine assumes the input to be in Q15 format. Hence, the input range of \([-\pi, +\pi]\) are normalized to \([-1, +1]\). Therefore, the input value of 0x7FFFh which corresponds to \(-1\) in Q15 format, represents the \(+\pi\) radians or 180 degree and similarly the input value of 0x8000 which corresponds to \(-1\) in Q15 notation, represents the \(-\pi\) radians or -180 degree.

Output
Since the output range of sine function is \(-1\) to \(+1\), the Q15 representation is the obvious choice.

Computation
The sin function computation is performed by means of the following fourth order Taylor Series approximation polynomial. The value of \(x\) in the below equation is in radians.

\[
\sin(x) = 0.0013 + 0.9826x + 0.0544x^2 - 0.2338x^3 + 0.0372x^4 \quad x \in [0, +1]
\]

Accuracy
The plot in the previous page shows the error of the fixed-point implementation of sin function with respect to the IEEE floating point (single precision) computation. The maximum error is 83 counts, resulting in precision of app. 10 bits over the full input range.

Normalization
Normalizing the input range of \(-\pi\) to \(+\pi\) radians to \([-1, +1]\) is nothing but a one to one mapping as shown below.

Let us say, the \(x\) is the radians within the range \(-\pi\) to \(+\pi\), then the normalized value of \(x\) in Q15 representation can be obtained by the following simple equation

\[
= \frac{x}{\pi} \times 2^{15}
\]
Description
This module computes the sine value of the input using table look-up and linear interpolation method.

Availability
This module is available in two interface formats:
1) The direct-mode assembly-only interface (Direct ASM)
2) The C-callable interface version

Module Properties
Type: Target Independent, Application Independent

Target Devices: x24x/x24xx

Direct ASM Version File Names: q1sinlt.asm, sintb360.asm

C-Callable Version File Names: qsinlt.asm, sintb360.asm, qmath.h

<table>
<thead>
<tr>
<th>Item</th>
<th>ASM Only</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>30 words</td>
<td>35 words</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>4 words</td>
<td>0 words</td>
<td></td>
</tr>
<tr>
<td>xDAIS ready</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>xDAIS component</td>
<td>No</td>
<td>No</td>
<td>IALG layer not implemented</td>
</tr>
<tr>
<td>Multiple instances</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>-</td>
<td>5 words</td>
<td>Stock grows by 5 words</td>
</tr>
</tbody>
</table>
Direct ASM Interface

Module Terminal Variables/Functions

<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Description</th>
<th>Format</th>
<th>Range(Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>sinlt_input</td>
<td>Angle in radians between ([-\pi, +\pi]) normalized to ([-1, +1])</td>
<td>Q15</td>
<td>8000 - 7FFF</td>
</tr>
<tr>
<td>Output</td>
<td>sinlt_output</td>
<td>cosine value of the input</td>
<td>Q15</td>
<td>8000 - 7FFF</td>
</tr>
</tbody>
</table>

Variable Declaration:
In the system file include the following statements:

```
 .ref     qsinlt    ; function call
 .ref     sinlt_input, sinlt_output   ; input & output
```

Memory map:
- All variables are mapped to an uninitialized named section “qmath”
- Look-up table entries are assembled in “SINTBL” section that has to be loaded in program memory

Example: The following sample code obtains the \( \sin(0.25 \times \pi) = 0.707 \)

Input
Sample input \( 0.25 \times \pi \) in normalized Q15 format

\[
\left( \frac{0.25 \times \pi}{\pi} \right) \times 2^{15} = 2000h
\]

Output
The Output is 0.707 in Q15 format = 0.707 \( \times 2^{15} = 23170 \) (5A82h)

Sample Code
```
 .ref     qsinlt    ; function call reference
 .ref     sinlt_input, sinlt_output   ; input & output reference
 .bss  input_var1,1   ; Reserve space for input
 .bss  output_var1,1    ; Reserve space for output
 LDP #input_var1    ; Store \( 0.25 \times \pi \) in normalized Q15
 SPLK #02000h,input_var1
 LDP #sinlt_input   ; Set DP for the module input
 BLDD #input_var1, sinlt_input   ; Pass input value to module input
 CALL     qsinlt
 LDP #output_var1    ; Set DP for output variables
 BLDD #sinlt_output, output_var1    ; Pass module output to output variable
```

Linker Command file:
Edit linker command file to place “SINTBL” section in program memory and also place the “qmath” section in data memory

```
 qmath : { }      > DATA      PAGE 1
 SINTBL    > PROG      PAGE 0
```

©Texas Instruments Inc., May 2002
C/C-Callable ASM Interface

Declaration
signed short int qsinlt(signed short int x)

Input Format
The argument ‘x’ is a fixed-point number in Q15 format that contains the angle in radians between \([-\pi, +\pi]\) normalized between \([-1, 1]\).

Output Format
This function returns the sine of the input argument as fixed-point number in Q15 format.

Example
The following sample code obtains the \(\sin(0.25 \times \pi) = 0.707\).

Input
The sample input \(0.25 \times \pi\) in normalized Q15 format
\[
\left( \frac{0.25 \times \pi}{\pi} \right) \times 2^{15} = 2000h
\]

Output
The output is 0.707 in Q15 format = \(0.707 \times 2^{15} = 23170 (5A82h)\).

Sample Code
```c
#include<qmath.h> /* Header file for fixed point math routine */
void main(void )
{
    signed short int x, y;
    x=0x2000; /* 0.25 \times \pi in normalized Q15 format = \left( \frac{0.25 \times \pi}{\pi} \right) \times 2^{15} = 2000h */
    y=qsinlt(x); /* ‘y’ will have \(\sin(0.25 \times \pi) = 0.707\) in Q15 */
}
```

Linker Command file:
Edit linker command file to place “SINTBL” section in program memory

```
SINTBL   > PROG   PAGE 0
```
**Table Look-up with linear interpolation vs Floating Point Analysis**

**SIN Function: Table look-up with linear interpolation vs C Float SIN \(-\pi < x < \pi\)**

- **SIN(x) Obtained by C-float (Q15 format)**
  - **Input 'x' (in radians)**
  - **SIN(x)**

- **SIN(x) Obtained by table look-up with linear interpolation (in Q15 format)**
  - **Input 'x' (in radians)**
  - **SIN(x)**

- **Error with respect to Floating point**
  - **Input 'x' (in radians)**
  - **Error (in Counts)**

---

©Texas Instruments Inc., May 2002
Background Information

Input
The range of input to the SIN function is limited to $-\pi$ to $+\pi$ radians. This function needs the input in Q15 format. Hence, the input range of $[-\pi, +\pi]$ are normalized to $[-1, +1]$. Therefore, the input value of 0x7FFFh which corresponds to $-1$ in Q15 format, represents the $+\pi$ radians or 180 degree and similarly the input value of 0x8000 which corresponds to $-1$ in Q15 notation, represents the $-\pi$ radians or -180 degree.

Output
Since the output range of SIN function is $-1$ to $+1$, the Q15 format is the obvious choice.

Computation
The SIN computation is performed using table interpolation technique that is comprised of two steps viz.,

1. Direct look-up: Looking through the table to find the interval $[x_i, x_{i+1}]$ at which the considered angle $x$ is located, with $x_i < x < x_{i+1}$.
2. Interpolation: Solving the equation given below to obtain $y$

\[ y = y_i + \frac{x-x_i}{x_{i+1}-x_i} \times (y_{i+1} - y_i) \] (1)

The SIN value is computed with a fixed step table of 256 SIN entries uniformly spaced from 0 to $2\pi$ for easier look-up operation. With fixed step table, it is possible to have a correspondence between the address of a point in the table and its abscissa. In this way the table look-up is instantaneous and constant in execution and the table size is reduced, only ordinates are stored and abscissa are memory addressed.

The 16-bit input scaled down by 256 to get the position of ordinate in the table and the remainder is used for interpolation.

Accuracy
The plot in the previous page shows the error of the SIN function implemented using table interpolation with respect to the IEEE floating point (single precision) results. The maximum error is 4 counts, resulting in precision of app. 14 bits over the full input range.

Normalization
Normalizing the input range of $-\pi$ to $+\pi$ radians to $[1, +1]$ is nothing but a one to one mapping as shown below.

Let us say, the $x$ is the radians within the range $-\pi$ to $+\pi$, then the normalized value of ‘$x$’ in Q15 representation can be obtained by the following simple equation

\[ = \frac{x}{\pi} \times 2^{15} \]
Description
This module computes the cosine value of the input using Taylor series approximation method

Availability
This module is available in two interface formats:
1) The direct-mode assembly-only interface (Direct ASM)
2) The C-callable interface version

Module Properties
Type: Target Independent, Application Independent
Target Devices: x24x/x24xx

Direct ASM Version File Names: q1cos.asm
C-Callable Version File Names: qcos.asm, qmath.h

<table>
<thead>
<tr>
<th>Item</th>
<th>ASM Only</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>60 words</td>
<td>63 words</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>4 words</td>
<td>0 words</td>
<td></td>
</tr>
<tr>
<td>xDAIS ready</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>XDAIS component</td>
<td>No</td>
<td>No</td>
<td>IALG layer not implemented</td>
</tr>
<tr>
<td>Multiple instances</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>-</td>
<td>5 words</td>
<td>Stock grows by 5 words</td>
</tr>
</tbody>
</table>
Direct ASM Interface

Module Terminal Variables/Functions

<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Description</th>
<th>Format</th>
<th>Range (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>cos_input</td>
<td>Angle in radians between $[-\pi, +\pi]$ normalized to $[-1, +1]$</td>
<td>Q15</td>
<td>8000 - 7FFF</td>
</tr>
<tr>
<td>Output</td>
<td>cos_output</td>
<td>cosine value of the input</td>
<td>Q15</td>
<td>8000 – 7FFF</td>
</tr>
</tbody>
</table>

Variable Declaration:
In the system file include the following statements:

```
 références qcos ; function call
 références cos_input, cos_output ; input & output
```

Memory map:
All variables are mapped to an uninitialized named section “qmath”

Example: The following sample code obtains the $\cos(0.25 \times \pi) = 0.707$

Input
The sample input $0.25 \times \pi$ in normalized Q15 format

$$= \left( \frac{0.25 \times \pi}{\pi} \right) \times 2^{15} = 0x2000$$

Output
The output is 0.707 in Q15 format = $0.707 \times 2^{15} = 23170.475$ (5A82h)

Sample Code
```assembly
 références qcos ; function call reference
 références cos_input, cos_output ; input & output reference

.bss input_var1, 1 ; Reserve space for input
.bss output_var1, 1 ; Reserve space for output

LDP #input_var1 ; Store $0.25 \times \pi$ in normalized Q15
SPLK #02000h, input_var1

LDP #cos_input ; Set DP for the module input
BLDD #input_var1, cos_input ; Pass input value to module input

CALL qcos
LDP #output_var1 ; Set DP for output variables
BLDD #cos_output, output_var1 ; Pass module output to output variable
```
C/C-Callable ASM Interface

Declaration
signed short int qcos(signed short int x)

Input Format
The argument ‘x’ is a fixed-point number in signed 1.15 format that contains the angle in radians between \([-\pi, +\pi]\) normalized between \([-1, +1]\).

Output Format
This function returns the cosine of the input argument as fixed-point number in signed 1.15 format.

Example
The following sample code obtains the \(\sin(0.25 \times \pi) = 0.707\)

Input
The sample input \(0.25 \times \pi\) in normalized Q15 format
\[= \left(\frac{0.25 \times \pi}{\pi}\right) \times 2^{15} = 2000h\]

Output
The output is 0.707 in Q15 format = 0.707 \(\times 2^{15} = 23170\) (5A82h)

Sample Code
```c
#include<qmath.h> /* Header file for fixed point math routine */

void main(void)
{
    signed short int x, y;
    x = 0x2000; /* 0.25 \times \pi \text{ normalized to Q15 format} = \left(\frac{0.25 \times \pi}{\pi}\right) \times 2^{15} = 2000h */
    y = qcos(x); /* 'y' will have \(\cos(0.25 \times \pi) = 0.707\) in Q15 */
}
/* 0.707 \text{ in Q15 format} = 0.707 \times 2^{15} = 23170 = 5A82h */
```
Fixed Point COS Function vs C Float COS ($\pm \pi \leq x \leq \pi$)

COS(x) Obtained by C Float (Q15 format)

COS(x) Obtained by C2000 Fixed Point Routine (Q15 Format)

Error of Fixed Point COS(x) Routine with respect to C Float
Background Information

Input
The range of input to the cosine function is limited to \(-\pi\) to \(+\pi\) radians. The Fixed-point sine computation routine assumes the input to be in Q15 format. Hence, the input range of \([-\pi, +\pi]\) are normalized to \([-1, +1\]). Therefore, the input value of 0x7FFFh which corresponds to \(\sim 1\) in Q15 format, represents the \(+\pi\) radians or 180 degree and similarly the input value of 0x8000h which corresponds to \(-1\) in Q15 notation, represents the \(-\pi\) radians or -180 degree.

Output
Since the output range of cosine function is \(-1\) to \(+1\), the Q15 representation is the obvious choice.

Computation
The cosine function computation is performed by means of the following fifth order Taylor Series approximation polynomial. The value of ‘\(x\)’ in the below equation is in radians.

\[
\cos(x) = 1.0 - 0.0028x - 0.4879x^2 - 0.0211x^3 + 0.0595x^4 - 0.0076x^5 \quad x \in [0, +0.5]
\]

The computation of cosine in other quadrant is obtained by using symmetries, which maps the angle ‘\(x\)’ into first quadrant.

Accuracy
The plot in the previous page shows the error of the fixed-point implementation of cosine function with respect to the IEEE floating point (single precision) computation. The maximum error is 21 counts, resulting in precision of about 12 bits over the full input range.

Normalization
Normalizing the input range of \(-\pi\) to \(+\pi\) radians to \([-1, +1]\) is nothing but a one to one mapping as shown below.

Let us say, the \(x\) is the radians within the range \(-\pi\) to \(+\pi\), then the normalized value of ‘\(x\)’ in Q15 representation can be obtained by the following simple equation

\[
\frac{x}{\pi} \times 2^{15}
\]
**Description**

This module computes the cosine value of the input using Table look-up and linear interpolation method.

**Availability**

This module is available in two interface formats:

1) The direct-mode assembly-only interface (Direct ASM)
2) The C-callable interface version

**Module Properties**

**Type:** Target Independent, Application Independent

**Target Devices:** x24x/x24xx

**Direct ASM Version File Names:** q1coslt.asm, sintb360.asm

**C-Callable Version File Names:** qcoslt.asm, sintb360.asm, qmath.h

<table>
<thead>
<tr>
<th>Item</th>
<th>ASM Only</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>34 words</td>
<td>39 words</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>4 words</td>
<td>0 words</td>
<td></td>
</tr>
<tr>
<td>xDAIS ready</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>xDAIS component</td>
<td>No</td>
<td>No</td>
<td>IALG layer not implemented</td>
</tr>
<tr>
<td>Multiple instances</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>-</td>
<td>5 words</td>
<td>Stock grows by 5 words</td>
</tr>
</tbody>
</table>
Module Terminal Variables/Functions

<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Description</th>
<th>Format</th>
<th>Range(Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>coslt_input</td>
<td>Angle in radians between $[-\pi, +\pi]$ normalized to $[-1, +1]$</td>
<td>Q15</td>
<td>8000 - 7FFF</td>
</tr>
<tr>
<td>Output</td>
<td>coslt_output</td>
<td>cosine value of the input</td>
<td>Q15</td>
<td>8000 - 7FFF</td>
</tr>
</tbody>
</table>

Variable Declaration:
In the system file include the following statements:

```
.ref     qcoslt ; function call
.ref     coslt_input, coslt_output ; input & output
```

Memory map:
- All variables are mapped to an uninitialized named section “qmath”
- Look-up table entries are assembled in “SINTBL” section that has to be loaded in program memory

Example: The following sample code obtains the $\cos(0.25\times\pi)=0.707$

Input
The sample input $0.25\times\pi$ in normalized Q15 format

$$
\left(\frac{0.25\times\pi}{\pi}\right)\times2^{15}=0x2000
$$

Output
The output is 0.707 in Q15 format $= 0.707 \times 2^{15} = 23170.475$ (5A82h)

Sample Code
```
.ref     q1coslt ; function call reference
.ref     coslt_input, coslt_output ; input & output reference
.bss  input_var1,1 ; Reserve space for input
.bss  output_var1,1 ; Reserve space for output

LDP #input_var1 ; Store $0.25\times\pi$ in normalized Q15
SPLK #02000h,input_var1

LDP #coslt_input ; Set DP for the module input
BLDD #input_var1, coslt_input ; Pass input value to module input

CALL qcoslt
LDP #output_var1 ; Set DP for output variables
BLDD #coslt_output, output_var1 ; Pass module output to output variable
```

Linker Command file:
Edit linker command file to place “SINTBL” section in program memory and also place the “qmath” section in data memory

```
qmath : { } > DATA PAGE 1
SINTBL > PROG PAGE 0
```
C/C-Callable ASM Interface

Declaration

signed short int qcoslt(signed short int x)

Input Format

The argument ‘x’ is a fixed-point number in signed 1.15 format that contains the angle in radians between \([-\pi, +\pi]\) normalized between \([-\pi, +\pi]\).

Output Format

This function returns the cosine of the input argument as fixed-point number in signed 1.15 format.

Example

The following sample code obtains the \(\cos(0.25\times \pi) = 0.707\).

Input

The sample input \(0.25\times \pi\) in normalized Q15 format

\[
\left(\frac{0.25\times \pi}{\pi}\right) \times 2^{15} = 2000h
\]

Output

The output is 0.707 in Q15 format = 0.707 \(\times 2^{15} = 23170 = 5A82h\)

Sample Code

```c
#include<qmath.h> /* Header file for fixed point math routine */
void main(void )
{
  signed short int x,y;
  x=0x2000; /* 0.25\times \pi normalized to Q15 format = \(\left(\frac{0.25\times \pi}{\pi}\right) \times 2^{15} = 2000h\) */
  y=qcoslt(x); /* 'y' will have \(\cos(0.25\times \pi) = 0.707\) in Q15 */
}
```

Linker Command file:

Edit linker command file to place “SINTBL” section in program memory

```
SINTBL > PROG PAGE 0
```
Table Look-up with linear interpolation vs Floating Point Analysis

COS Function: Table look-up with linear interpolation vs C Float COS (-π<x<π)

COS(x) Obtained by C-float (Q15 format)

COS(x) Obtained by table look-up with linear interpolation (in Q15 format)

Error with respect to Floating point

©Texas Instruments Inc., May 2002
**Input**
The range of input to the COS function is limited to $-\pi$ to $+\pi$ radians. This function needs the input in Q15 format. Hence, the input range of $[-\pi, +\pi]$ are normalized to $[-1, +1]$. Therefore, the input value of 0x7FFFh which corresponds to ~1 in Q15 format, represents the $+\pi$ radians or 180 degree and similarly the input value of 0x8000 which corresponds to $-\pi$ radians or -180 degree.

**Output**
Since the output range of COS function is $-1$ to $+1$, the Q15 format is the obvious choice.

**Computation**
The COS computation is performed using table interpolation technique that is comprised of three steps viz.,
1. Adding $\pi/2$ to the input ‘$x$’ and computing the SIN using the sin look-up table
2. Direct look-up: Looking through the table to find the interval $[x_i, x_{i+1}]$ at which the considered angle ‘$x$’ is located, with $x_i < x < x_{i+1}$.
3. Interpolation: Solving the equation given below to obtain ‘$y$’

$$y = y_i + \frac{x - x_i}{x_{i+1} - x_i} \times (y_{i+1} - y_i) \quad (1)$$

The SIN value is computed with a fixed step table of 256 SIN entries, uniformly spaced from 0 to $2\pi$ for easier look-up operation. With fixed step table, it is possible to have a correspondence between the address of a point in the table and its abscissa. In this way the table look-up is instantaneous and constant in execution and the table size is reduced, only ordinates are stored and abscissa are memory addressed.

The 16-bit input scaled down by 256 to get the position of ordinate in the table and the remainder is used for interpolation.

The resulting output will be cosine of the input due the following symmetry.

$$\sin(x + \pi/2) = \cos(x) \quad (2)$$

**Accuracy**
The plot in the previous page shows the error of the COS function implemented using table interpolation with respect to the IEEE floating point (single precision) results. The maximum error is 4 counts, resulting in precision of app. 14 bits over the full input range.
Background Information

**Normalization**

Normalizing the input range of $-\pi$ to $+\pi$ radians to $[-1, +1]$ is nothing but a one to one mapping as shown below.

Let us say, the $x$ is the radians within the range $-\pi$ to $+\pi$, then the normalized value of ‘$x$’ in Q15 representation can be obtained by the following simple equation

$$\frac{x}{\pi} \times 2^{15}$$
**QATAN**  

**Fixed point ATAN (Taylor Series Implementation)**

**Description**  
This module computes the Inverse Tangent of the input using Taylor series approximation method

![Diagram of QATAN module with inputs atan_input and outputs atan_output]

**Availability**  
This module is available in two interface formats:
1) The direct-mode assembly-only interface (Direct ASM)  
2) The C-callable interface version

**Module Properties**  
**Type:** Target Independent, Application Independent  
**Target Devices:** x24x/x24xx  
**Direct ASM Version File Names:** q1atan.asm  
**C-Callable Version File Names:** qatan.asm, qmath.h

<table>
<thead>
<tr>
<th>Item</th>
<th>ASM Only</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>87 words</td>
<td>63 words</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>5 words</td>
<td>0 words</td>
<td></td>
</tr>
<tr>
<td>xDAIS ready</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>xDAIS component</td>
<td>No</td>
<td>No</td>
<td>IALG layer not implemented</td>
</tr>
<tr>
<td>Multiple instances</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>-</td>
<td>6 words</td>
<td>Stock grows by 6 words</td>
</tr>
</tbody>
</table>
Module Terminal Variables/Functions

<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Description</th>
<th>Format</th>
<th>Range(Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>atan_input</td>
<td>Input argument is represented in signed 16.16 format</td>
<td>16.16</td>
<td>80000000h to 7FFFFFFFh</td>
</tr>
<tr>
<td>Output</td>
<td>atan_output</td>
<td>Output Angle in radians between $[-\pi/2, +\pi/2]$ normalized to $[5.0, 5.0]$</td>
<td>Q15</td>
<td>C000 – 4000</td>
</tr>
</tbody>
</table>

Variable Declaration:
In the system file include the following statements:

```
.ref     qatan    ; function call
.ref     atan_input, atan_output   ; input & output
```

Memory map:
All variables are mapped to an uninitialized named section "qmath"

Example: The following sample code obtains the $\tan^{-1}(50.8) = 1.551$ radians

Input
The sample input value 50.8, in 16.16 format
$= 50.8 \times 2^{16} = 3329228$ (0032 CCCCh)

Output
The output, $1.551$ radiance in normalized Q15 format
$= \frac{1.551}{\pi} \times 2^{15} = 16178$ (3F32h)

Sample Code
```
.ref     qatan    ; function call reference
.ref     atan_input, atan_output   ; input & output reference
.bss     input_var1,2,1    ; Reserve 2 words for Long input
.bss     output_var1,1     ; Reserve 1 word for output
LDP     #input_var1
SPLK    #0CCCh, input_var1
SPLK    #0032h,input_var1+1
LDP     #atan_input      ; Set DP for the module input
BLDD    #input_var1, atan_input ; Pass the LSW of input to the module
BLDD    #input_var1+1,atan_input+1 ; Pass the MSW of input to the module
CALL    qatan
LDP     #output_var1     ; Set DP for output variables
BLDD    #atan_output, output_var1 ; Pass module output to output variable
```
C/C-Callable ASM Interface

Declaration

signed short int atan(signed long int x)

Input Format

The argument ‘x’ is a fixed-point number in signed 16.16 format. This signed 16.16 number representation has a range of from 
\(-2^{15} \cdot 1.5\) to \(2^{15} \cdot 1/2^{16}\) \((7FFF.FFFFh)\). Therefore, the maximum and minimum value for which the inverse tangent can be obtained with this function is \(-32768\) and \(-32768\) respectively.

Output Format

This function returns the inverse tangent of the input argument as fixed-point number in signed 1.15 format. The output contains the angle in radians between \(-\pi/2, \pi/2\) normalized between \([-0.5, +0.5]\) in Q15 format.

Example

The following sample code obtains the \(\tan^{-1}(50.8) = 1.551\) radians.

Input

The sample input value 50.8, in 16.16 format

\[= 50.8 \times 2^{16} = 3329228 (0032 CCCCh)\]

Output

The output, 1.551 radian in normalized Q15 format

\[= \frac{1.551}{\pi} \times 2^{15} = 16178 (3F32h)\]

Sample Code

```c
#include<qmath.h> /* Header file for fixed point math routine */
void main(void ) {
    signed long int x; 
    signed short int y;
    x=0x32cccc; /* 50.8 in Q16 format = 50.8 \times 2^{16} = 3329228 = 32CCCCh */
    y=qatan(x); /* ‘y’ will have normalized value of \(\tan^{-1}(50.8) = 0.49373\) in Q15 */
} /* 0.49373 in Q15 format = 0.49373 \times 2^{15} = 16178 = 3F32h */
```
Fixed Point ARCTAN Function vs C Float ARCTAN (-32768<x<32768)

ATAN(x) Obtained by C Float (Q15 format)

ATAN(x) Obtained by C2000 Fixed Point Routine (Q15 Format)

Error of Fixed Point ATAN(x) Routine with respect to C Float
Fixed Point ARCTAN Function vs C Float ARCTAN (-10<x<10)

ATAN(x) Obtained by C Float (Q15 format)

ATAN(x) Obtained by C2000 Fixed Point Routine (Q15 Format)

Error of Fixed Point ATAN(x) Routine with respect to C Float
**Background Information**

**Input**
Since the argument of Inverse Tangent varies from \(-\infty\) to \(+\infty\), the scaling is no longer possible. Hence the argument is represented as a signed 32-bit number in the 16.16 format to cater to the range as well as resolution. The upper 16 bit of the 32-bit word is used for the integer part and the lower 16 bits are used for the fractional part of the input real number.

The range of the 16.16 number: \(-2^{15} (8000.0000h)\) to \(2^{15} \cdot \frac{1}{2^{16}} (7FFF.FFFFh)\).

To find out the Inverse tangent of any number within the above range, convert the number to 16.16 format and then input it to the QTAN module. The process of converting a real number ‘x’ to 16.16 format is very simple and it is given below.

Formula to convert the real number to 16.16 format = \(x \times 2^{16}\)

**Output**
The output value is an angle in the range from \([-\frac{\pi}{2}, +\frac{\pi}{2}\]) corresponding to \([-0.5, +0.5]\) in the normalized Q15 notation

**Computation**
The inverse tangent computation is performed by means of the following fifth order Taylor Series approximation polynomial

\[
\tan^{-1}(x) = 0.318253x + 0.003314x^2 - 0.130908x^3 + 0.068542x^4 - 0.009159x^5 \quad x \in [0, 1]
\]

\[
= 0.5 \cdot \tan^{-1}\left(\frac{1}{x}\right) \quad x \geq 1.0
\]

**Accuracy**
The plot in the previous two pages shows the error of the fixed-point implementation of the inverse tangent function with respect to the IEEE floating point (single precision) computation. The maximum error is 3 counts, resulting in precision of 14 bits over the full input range.
Description
This module computes the square root of the input using the Taylor series approximation method.

Availability
This module is available in two interface formats:
1) The direct-mode assembly-only interface (Direct ASM)
2) The C-callable interface version

Module Properties
Type: Target Independent, Application Independent

Target Devices: x24x/x24xx

Direct ASM Version File Names: q1sqrt.asm

C-Callable Version File Names: qsqrt.asm, qmath.h

<table>
<thead>
<tr>
<th>Item</th>
<th>ASM Only</th>
<th>C-Callable</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>113 words</td>
<td>115 words</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>5 words</td>
<td>0 words</td>
<td></td>
</tr>
<tr>
<td>xDAIS ready</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>XDAIS component</td>
<td>No</td>
<td>No</td>
<td>IALG layer not implemented</td>
</tr>
<tr>
<td>Multiple instances</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>-</td>
<td>6 words</td>
<td>Stock grows by 6 words</td>
</tr>
</tbody>
</table>
Direct ASM Interface

Module Terminal Variables/Functions

<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Description</th>
<th>Format</th>
<th>Range(Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>sqrt_input</td>
<td>Input argument represented in unsigned 16.16 format</td>
<td>unsigned 16.16</td>
<td>00000000h to                  FFFFFFFFh</td>
</tr>
<tr>
<td>Output</td>
<td>sqrt_output</td>
<td>Output is represented in unsigned 8.8 format</td>
<td>unsigned 8.8</td>
<td>0000 – FFFFh</td>
</tr>
</tbody>
</table>

Variable Declaration:

In the system file include the following statements:

```
.ref     qsqrt    ; function call
.ref     sqrt_input, sqrt_output   ; input & output
```

Memory map:

All variables are mapped to an uninitialized named section “qmath”

Example: The following sample code obtains the $\sqrt{50.8} = 7.127411$

Input

The sample input value 50.8, in unsigned 16.16 format

$= 50.8 \times 2^{16} = 3329228$ (0032 CCCCh)

Output

The output is 7.127411 in unsigned 8.8 format

$= 7.127411 \times 2^8 = 1824$ (0720h)

Sample Code

```
.ref     qsqrt    ; function call reference
.ref     sqrt_input, sqrt_output   ; input & output reference
.bss input_var1,2,1    ; Reserve 2 words for Long input
.bss output_var1,1    ; Reserve 1 word for output
LDP #input_var1
SPLK #0CCCCh, input_var1   ; Store the input 50.8 in 16.16 format
SPLK #00032h,input_var1+1
LDP #sqrt_input       ; Set DP for the module input
BLDD #input_var1, sqrt_input ; Pass the LSW of input to the module
BLDD #input_var1+1,sqrt_input+1  ; Pass the MSW of input to the module
CALL qsqrt
LDP #output_var1    ; Set DP for output variables
BLDD #sqrt_output, output_var1 ; Pass module output to output variable
```
C/C-Callable ASM Interface

Declaration
unsigned short int qsqrt(unsigned long int x)

Input Format
The argument ‘x’ is a fixed-point number in unsigned 16.16 format. This unsigned 16.16 number representation has a range of from 0.0 to 2^{16} - \frac{1}{2^{16}} (FFFF.FFFFh). Therefore, the maximum value for which the square root can be obtained with this function is ~65536.

Output Format
This function returns the square root of the input argument as fixed-point number in unsigned 8.8 format

Example
The following sample code obtains the $\sqrt{50.8} = 7.127411$

Input
The sample input value 50.8, in unsigned 16.16 format
= 50.8 \times 2^{16} = 3329228 (0032 CCCCh)

Output
The output is 7.127411 in unsigned 8.8 format
= 7.127411 \times 2^{8} = 1824 (0720h)

Sample Code
```
#include<qmath.h>  /* Header file for fixed point math routine */
void main(void ) {
    unsigned long int x;
    unsigned short int y;
    x=0x32cccc;     /* 50.8 in Q16 format = 50.8 \times 2^{16} = 3329228(32CCCh) */
    y=qsqrt(x);     /* 'y' will have $\sqrt{50.8} = 7.127411$ in 8.8 format */
}              /* 7.127411 in 8.8 format=7.127411 \times 2^{8} = 1824(0720h) */
```
Fixed Point vs Floating Point Analysis

Fixed Point SQRT Function vs C Float SQRT (0<x<65535)

SQRT(x) Obtained by C Float (Represented in unsigned 8.8 format)

SQRT(x) Obtained by C2000 Fixed Point Routine (unsigned 8.8 Format)

Error of Fixed Point SQRT(x) Routine with respect to C Float
Background Information

Input
As the square root exist only for the positive number, the unsigned number is used to get the better range when compared to the signed number. The square root routine requires 32-bit input in unsigned 16.16 format to cater to the range as well as the resolution. The upper 16 bit of the 32-bit word is used for the integer part and the lower 16 bits are used for the fractional part of the positive real number input.

The range of the unsigned 16.16 format is 0 to ~65536 (2^{16} \cdot 1/2^{16})

To find out the square root of any number within the above range, convert the number to 16.16 format and then input it to the QSQRT module. The process of converting a real number ‘x’ to 16.16 format (unsigned) is very simple and it is given below.

Formula to convert the real number to unsigned 16.16 format = \( x \times 2^{16} \)

Output
The maximum value of the output of the square root routine is ~\( \sqrt{65536} \approx 256 \). The unsigned 8.8 format can cater to this range with the maximum possible resolution. Hence the unsigned 8.8 format is selected for the output.

Computation
The square root computation is performed by means of the following fifth order Taylor series approximation polynomial

\[
0.5\sqrt{x} = 0.7274475x - 0.672455x^2 + 0.553406x^3 - 0.2682495x^4 + 0.0560605x^5 + 0.1037903, \quad x \in [0.5, 1.0]
\]

The square root of input values outside this range is obtained by scaling the number to within the range [0.5, 1.0]. Multiplying by powers of 2 if the input is less than 0.5 or dividing by powers of 2 if the input is greater than 1 will bring the input to lie with in [0.5, 1.0] range. Scaling by this way helps to obtain the square root of the scaling value easily due to the fact that the scaling will be either positive or negative powers of 2.

After computing the square root of the scaled number, it has to be divided by the square root of the scaling value to get the square root of the original number

\[
y = \sqrt{x}
\]

\[
z = \sqrt{s \times x} = \sqrt{s} \times \sqrt{x}
\]

\[
y = \frac{z}{\sqrt{s}} = \sqrt{x}
\]
Accuracy
The plot in the previous page shows the error of the fixed-point implementation of the square root function with respect to the IEEE floating point (single precision) computation. The maximum error is 19 counts, resulting in precision of about 12 bits over the full input range
QLOG10

**Fixed point LOG10 (Taylor Series Implementation)**

**Description**
This module computes the common log of the input using the Taylor series approximation method.

![Diagram]

**Availability**
This module is available in two interface formats:
1) The direct-mode assembly-only interface (Direct ASM)
2) The C-callable interface version

**Module Properties**
**Type:** Target Independent, Application Independent

**Target Devices:** x24x/x24xx

**Direct ASM Version File Names:** q1log10.asm

**C-Callable Version File Names:** qlog10.asm, qmath.h

<table>
<thead>
<tr>
<th>Item</th>
<th>ASM Only</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>66 words</td>
<td>68 words</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>5 words</td>
<td>0 words</td>
<td></td>
</tr>
<tr>
<td>xDAIS ready</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>XDAIS component</td>
<td>No</td>
<td>No</td>
<td>IALG layer not implemented</td>
</tr>
<tr>
<td>Multiple instances</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>-</td>
<td>6 words</td>
<td>Stock grows by 6 words</td>
</tr>
</tbody>
</table>
Module Terminal Variables/Functions

<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Description</th>
<th>Format</th>
<th>Range(Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>log10_input</td>
<td>Input argument represented in unsigned 16.16 format</td>
<td>unsigned 16.16</td>
<td>00000000h to FFFFFFFFh</td>
</tr>
<tr>
<td>Output</td>
<td>log10_output</td>
<td>Output is represented in signed 4.12 format</td>
<td>signed 4.12</td>
<td>B2F0h – 4D10h</td>
</tr>
</tbody>
</table>

Variable Declaration:
In the system file include the following statements:

```
.ref qlog10        ; function call
.ref log10_input, log10_output   ; input & output
```

Memory map:
All variables are mapped to an uninitialized named section “qmath”

Example: The following sample code obtains the \( \log_{10}(50.8) = 1.705863 \)

**Input**
The sample input value 50.8, in unsigned 16.16 format
\[ = 50.8 \times 2^{16} = 3329228 \ (0032 \text{ CCCCh}) \]

**Output**
The output is 1.705863 in signed 4.12 format
\[ = 1.705863 \times 2^{12} = 6987 \ (1B4Bh) \]

Sample Code
```
.ref qlog10        ; function call reference
.ref log10_input, log10_output   ; input & output reference
.bss input_var1,2,1    ; Reserve 2 words for Long input
.bss output_var1,1    ; Reserve 1 word for output
LDP #input_var1
SPLK #0CCCCh, input_var1    ; Store the input 50.8 in 16.16 format
SPLK #00032h,input_var1+1
LDP #log10_input        ; Set DP for the module input
BLDD #input_var1, log10_input    ; Pass the LSW of input to the module
BLDD #input_var1+1,log10_input+1    ; Pass the MSW of input to the module
CALL qlog10
LDP #output_var1        ; Set DP for output variables
BLDD #log10_output, output_var1    ; Pass module output to output variable
```
C/C-Callable ASM Interface

Declaration
signed short int qlog10(unsigned long int x)

Input Format
The argument ‘x’ is a fixed-point number in unsigned 16.16 format. This unsigned 16.16 number representation has a range of from 0.0 to $2^{16} - 1/2^{16}$ (FFFF.FFFFh). Therefore, the maximum value for which the common logarithm can be obtained with this routine is ~65536.

Output Format
This function returns the common log of the input argument as fixed-point number in signed 4.12 format

Example
The following sample code obtains the $\log_{10}(50.8) = 1.705863$

Input
The sample input value 50.8, in unsigned 16.16 format
$= 50.8 \times 2^{16} = 3329228$ (0032 CCCCh)

Output
The output is 1.705863 in signed 4.12 format
$= 1.705863 \times 2^{12} = 6987$ (1B4Bh)

Sample Code
#include<qmath.h> /* Header file for fixed point math routine */
void main(void )
{
    unsigned long int x;
    signed short int y;

    x=0x32cccc; /* 50.8 in Q16 format = 50.8 \times 2^{16} = 3329228(032CCCCh */
    y=qlog10(x); /* 'y' will have $\log_{10}(50.8) = 1.705863$ in 4.12 format */
}
/* 1.705863 in 4.12 format = 1.705863 \times 2^{12} = 6987(1B4B) */
Fixed Point vs Floating Point Analysis

Fixed Point LOG10 Function vs C Float LOG10 (0<x<65535)

LOG10(x) Obtained by C Float (Represented in signed 4.12 format)

LOG10(x) Obtained by C2000 Fixed Point Routine(signed 4.12 Format)

Error of Fixed Point LOG10(x) Routine with respect to C Float
Fixed Point LOG10 Function vs C Float LOG10 (0<x<5)

LOG10(x) Obtained by C Float (Represented in signed 4.12 format)

LOG10(x) Obtained by C2000 Fixed Point Routine (signed 4.12 Format)

Error of Fixed Point LOG10(x) Routine with respect to C Float

©Texas Instruments Inc., May 2002
Background Information

Input
As the common log exist only for the positive number, the unsigned number is used to get the better range when compared to the signed number. The common log routine requires 32-bit input in unsigned 16.16 format to cater to the range as well as the resolution. The upper 16 bit of the 32-bit word is used for the integer part and the lower 16 bits are used for the fractional part of the positive real number input.

The range of the unsigned 16.16 format is 0 to $\sim 65536 \left(2^{16} \cdot \frac{1}{2^{16}}\right)$

To find out the common log of any number within the above range, convert the number to 16.16 format and then input it to the QLOG10 module. The process of converting a real number ‘x’ to 16.16 format (unsigned) is very simple and it is given below.

Formula to convert the real number to unsigned 16.16 format = $x \times 2^{16}$

Output
Maximum output of the QLOG10 routine
$\sim \log_{10}(65536) = \sim 4.8165$
Minimum output of the QLOG10 routine
$= \log_{10}(1/65536) = -4.8165$

The signed 4.12 format can cater to the above range with the maximum resolution. Hence the signed 4.12 format is selected for the output

Maximum value in 4.12 format $\sim \log_{10}(65536) \times 2^{12} = \sim 19728 \ (4D10h)$
Minimum value in 4.12 format $= \log_{10}(65536) \times 2^{12} = -19728 \ (B2F0h)$

Computation
The common log computation is performed by means of the following fifth order Taylor series approximation polynomial

$$
2\log_{10}(x) = 0.8678284(x-1) - 0.4255677(x-1)^2 + 0.2481384(x-1)^3 - 0.1155701(x-1)^4 + 0.0272522(x-1)^5 \quad x \in [1.0, 2.0]
$$

The common log of input values outside this range is obtained by scaling the number to within the range $[1.0, 2.0]$. Multiplying by powers of 2 if the input is less then 1.0 or dividing by powers of 2 if the input is greater then 2.0 will bring the input to lie with in $[1.0, 2.0]$ range. Scaling by this way helps to obtain the common logarithm of the scaling value easily due to the fact that the scaling will be either positive or negative powers of 2.
The common logarithm of unscaled input is equal to the logarithm of the scaled input less the logarithm of the scaling factor ‘s’.

\[
y = \log_{10}(x)
\]

\[
z = \log_{10}(s \times x) = \log_{10}(s) + \log_{10}(x)
\]

\[
y = z - \log_{10}(s)
\]

**Accuracy**

The plot in the previous two pages shows the error of the fixed-point implementation of the common log function with respect to the IEEE floating point (single precision) computation. The maximum error is 2 counts, resulting in precision of 14 odd bits over the full input range.
**QLOGN**

*Fixed point LOGN (Taylor Series Implementation)*

**Description**

This module computes the natural log of the input using the Taylor series approximation method.

![Diagram](qlogn.png)

**Availability**

This module is available in two interface formats:

1) The direct-mode assembly-only interface (Direct ASM)
2) The C-callable interface version

**Module Properties**

*Type:* Target Independent, Application Independent

*Target Devices:* x24x/x24xx

*Direct ASM Version File Names:* q1logn.asm

*C-Callable Version File Names:* qlogn.asm, qmath.h

<table>
<thead>
<tr>
<th>Item</th>
<th>ASM Only</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>66 words</td>
<td>68 words</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>5 words</td>
<td>0 words</td>
<td></td>
</tr>
<tr>
<td>xDAIS ready</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>XDAIS component</td>
<td>No</td>
<td>No</td>
<td>IALG layer not implemented</td>
</tr>
<tr>
<td>Multiple instances</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>-</td>
<td>6 words</td>
<td>Stock grows by 6 words</td>
</tr>
</tbody>
</table>
Module Terminal Variables/Functions

<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Description</th>
<th>Format</th>
<th>Range (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>logn_input</td>
<td>Input argument represented in unsigned 16.16 format</td>
<td>unsigned</td>
<td>00000000h to FFFFFFFFh</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>16.16</td>
<td></td>
</tr>
<tr>
<td>Output</td>
<td>logn_output</td>
<td>Output is represented in signed 5.11 format</td>
<td>signed</td>
<td>A747h – 58B9h</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>5.11</td>
<td></td>
</tr>
</tbody>
</table>

Variable Declaration:
In the system file include the following statements:

```
/ref     qlogn    ; function call
/ref     logn_input, logn_output   ; input & output
```

Memory map:
All variables are mapped to an uninitialized named section “qmath”

Example: The following sample code obtains the \(\log_e (50.8) = 3.92789\)

Input
The sample input value 50.8, in unsigned 16.16 format
\[= 50.8 \times 2^{16} = 3329228 \text{ (0032 CCCCh)}\]

Output
The output is 3.92789 in signed 5.11 format
\[= 3.92789 \times 2^{11} = 8044 \text{ (1F6Ch)}\]

Sample Code
```
/ref     qlogn    ; function call reference
/ref     logn_input, logn_output   ; input & output reference
.bss     input_var1,2,1    ; Reserve 2 words for Long input
.bss     output_var1,1    ; Reserve 1 word for output
LDP     #input_var1
SPLK    #0CCCh, input_var1   ; Store the input 50.8 in 16.16 format
SPLK    #00032h,input_var1+1
LDP     #logn_input     ; Set DP for the module input
BLDD    #input_var1, logn_input    ; Pass the LSW of input to the module
BLDD    #input_var1+1,logn_input+1  ; Pass the MSW of input to the module
CALL    qlogn
LDP     #output_var1     ; Set DP for output variables
BLDD    #logn_output, output_var1  ; Pass module output to output variable
```
C/C-Callable ASM Interface

Declaration
signed short int qlogn(unsigned long int x)

Input Format
The argument ‘x’ is a fixed-point number in unsigned 16.16 format. This unsigned 16.16 number representation has a range of from 0.0 to $2^{16} - 1/2^{16}$ (FFFF.FFFFh) Therefore, the maximum value for which the natural logarithm can be obtained with this function is ~65536.

Output Format
This function returns the natural log of the input argument as fixed-point number in signed 5.11 format

Example
The following sample code obtains the $\log_e (50.8) = 3.92789$

Input
The sample input value 50.8, in unsigned 16.16 format
$= 50.8 \times 2^{16} = 3329228$ (0032 CCCCh)

Output
The output is 3.92789 in signed 5.11 format
$= 3.92789 \times 2^{11} = 8044$ (1F6Ch)

Sample Code
```
#include<qmath.h>  /* Header file for fixed point math routine               */
void main(void )
{
    unsigned long int x;
    signed short int y;

    x=0x32cccc;    /* 50.8 in Q16 format = 50.8*2^{16} = 3329228(032CCCCh) */
    y=qlogn(x);    /* 'y' will have $\log_e (50.8) = 3.92789$ in 5.11 format */
}   /* 3.92789 in 5.11 format = 3.92789*2^{11} = 8044 (1F6Ch) */
```
Fixed Point LOGN Function vs C Float LOGN (0<x<65535)

LOGN(x) Obtained by C Float (Represented in signed 5.11 format)

LOGN(x) Obtained by C2000 Fixed Point Routine (signed 5.11 Format)

Error of Fixed Point LOGN(x) Routine with respect to C Float
Fixed Point VS Floating Point Analysis

Fixed Point LOGN Function vs C Float LOGN (0<x<5)

LOGN(x) Obtained by C Float (Represented in signed 5.11 format)

LOGN(x) Obtained by C2000 Fixed Point Routine (signed 5.11 Format)

Error of Fixed Point LOGN(x) Routine with respect to C Float
**Background Information**

**Input**
As the natural log exist only for the positive number, the unsigned number is used to get the better range when compared to the signed number. The natural log routine requires 32-bit input in unsigned 16.16 format to cater to the range as well as the resolution. The upper 16 bit of the 32-bit word is used for the integer part and the lower 16 bits are used for the fractional part of the positive real number input.

The range of the unsigned 16.16 format is 0 to \(~65536\) (\(2^{16}/2^{16}\))

To find out the common log of any number within the above range, convert the number to 16.16 format and then input it to the QLOGN module. The process of converting a real number ‘x’ to 16.16 format (unsigned) is very simple and it is given below.

Formula to convert the real number to unsigned 16.16 format = \(x \times 2^{16}\)

**Output**
Maximum output of the QLOGN routine
\(~\log_e (65536)\) = ~11.0903

Minimum output of the QLOGN routine
\(=\log_e (1/65536)\) = -11.0903

The signed 5.11 format can cater to the above range with the maximum resolution. Hence the signed 5.11 format is selected for the output

Maximum value in 5.11 format \(~\log_e (65536) \times 2^{11}\) = ~22713 (58B9h)

Minimum value in 5.11 format = \(\log_e (65536) \times 2^{11}\) = -22713 (A747h)

**Computation**
The natural log computation is performed by means of the following fifth order Taylor series approximation polynomial

\[ \log_e(x) = 0.999115(x-1) - 0.4899597(x-1)^2 + 0.2856751(x-1)^3 - 0.1330566(x-1)^4 + 0.03137207(x-1)^5 \quad x \in [1.0, 2.0] \]

The natural log of input values outside this range is obtained by scaling the number to within the range \([1.0, 2.0]\). Multiplying by powers of \(2\) if the input is less than \(1.0\) or dividing by powers of \(2\) if the input is greater than \(2.0\) will bring the input to lie with in \([1.0, 2.0]\) range. Scaling by this way helps to obtain the natural logarithm of the scaling value easily due to the fact that the scaling will be either positive or negative powers of \(2\).
The natural logarithm of unscaled input is equal to the logarithm of the scaled input less the logarithm of the scaling factor ‘s’.

\[ y = \log_e (x) \]
\[ z = \log_e (s \times x) = \log_e (s) + \log_e (x) \]
\[ y = z - \log_e (s) \]

Accuracy

The plot in the previous two pages shows the error of the fixed-point implementation of the natural log function with respect to the IEEE floating point (single precision) computation. The maximum error is 2 counts, resulting in precision of 14 odd bits over the full input range.
**QINV1**

**Reciprocal (32-bit Precision)**

**Description**
This module computes the Reciprocal of input and provides the 32-bit quotient

![Diagram](https://via.placeholder.com/150)

**Availability**
This module is available in two interface formats:

1) The direct-mode assembly-only interface (Direct ASM)
2) The C-callable interface version

**Module Properties**

**Type:** Target Independent, Application Independent

**Target Devices:** x24x/x24xx

**Direct ASM Version File Names:** q1inv1.asm

**C-Callable Version File Names:** qinv1.asm, qmath.h

<table>
<thead>
<tr>
<th>Item</th>
<th>ASM Only</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>47 words</td>
<td>48 words</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>5 words</td>
<td>0 words</td>
<td></td>
</tr>
<tr>
<td>xDAIS ready</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>XDAIS component</td>
<td>No</td>
<td>No</td>
<td>IALG layer not implemented</td>
</tr>
<tr>
<td>Multiple instances</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>-</td>
<td>5 words</td>
<td>Stock grows by 5 words</td>
</tr>
</tbody>
</table>
Module Terminal Variables/Functions

<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Description</th>
<th>Format</th>
<th>Range(Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>inv1_input</td>
<td>Input argument represented in signed m.n format (m+n=16)</td>
<td>Signed m.n</td>
<td>8000h to 7FFFh</td>
</tr>
<tr>
<td>Output</td>
<td>inv1_output</td>
<td>Output is represented in signed 1+n.31-n format</td>
<td>Signed 1+n.31-n</td>
<td>80000000h to 7FFFFFFFh</td>
</tr>
</tbody>
</table>

Variable Declaration:
In the system file include the following statements:

```
.reff qinv1 ; function call
.ref inv1_input, inv1_output ; input & output
```

Memory map:
All variables are mapped to an uninitialized named section "qmath"

Example: The following sample code obtains the \( \text{inv}(8.125) = 0.123077 \)

**Input**

*Input format is assumed to be in signed 8.8 format*

The sample input value 8.125, in signed 8.8 format = \( 8.125 \times 2^8 = 2080 \) (0820h)

**Output**

*Output will be in signed 9.23 format*

The output 0.1230769, in Q23 format = \( 0.123077 \times 2^{23} = \text{FC0FCh} \)

Sample Code

```
.reff qinv1 ; function call reference
.ref inv1_input, inv1_output ; input & output reference
.bss input_var1, 1 ; Reserve 1 word for input
.bss output_var1, 2, 1 ; Reserve 2 word for long output
LDP #input_var1
SPLK #0820h, input_var1 ; Store the input 8.125 in 8.8 format
LDP #inv1_input ; Set DP for the module input
BLDD #input_var1, inv1_input ; Pass the value of input to the module
CALL qinv1
LDP #output_var1 ; Set DP for output variables
BLDD #inv1_output, output_var1 ; LSW of output
BLDD #inv1_output+1, output_var1+1 ; MSW of output
```
C/C-Callable ASM Interface

Declaration
signed long int qinv1(signed short int x)

Input Format
The argument ‘x’ is a fixed-point number in signed \(m.n\) format. This signed \(m.n\) number representation has a range from \(-2^{m-1}(0x8000)\) to \(2^{m-1} - \frac{1}{2^n}(0x7fff)\)

Example: The range of fixed-point number in signed 16.0 format 
\([-32768,32767]\)

Output Format
This function returns the reciprocal of the input argument as fixed-point number in signed \((1+n)\).(31-n) format.

Example: If the input is in 16.0 format, then the output will be in 1.31 format

Example
The following sample code obtains the \(qinv1(8.125) = 0.123077\)

Input
Input format is assumed to be in signed 8.8 format
The sample input value 8.125, in signed 8.8 format = \(8.125 \times 2^8 = 2080\) (0820h)

Output
Output will be in signed 9.23 format
The output 0.1230769, in Q23 format = \(0.1230769 \times 2^{23} = \text{FC0FCh}\)

Sample Code
```c
#include<qmath.h> /* Header file for fixed point math routine */

void main(void )
{
    signed long int y;
    signed short int x;

    x=0x820; /* 8.125 in 8.8 format = 8.125 \times 2^8 = 2080 \text{ (0820h)} */
    y=qinv1(x); /* 'y' will have \(qinv1(8.125) = 0.1230769\) in 9.23 */
}
```

/* 0.1230769 in 9.23 format =0.1230769 \times 2^{23} = \text{FC0FCh} */
32 bit Quotient Computation

Fixed-point division is implemented by repeated subtractions executed with SUBC, a special conditional subtract instruction.

Given a 16 bit positive dividend and divisor, the repetition of the SUBC command 16 times produces a 16-bit quotient in the low accumulator and a 16-bit remainder in the high accumulator. For the reciprocal computation dividend is 1, as we wish to find the \( \frac{1}{x} \) of the input ‘x’. Both the dividend and the divisor must be positive when using the SUBC command. The absolute value of the input is obtained before performing the division and the result is negated if the input is negative.

1. Keep the Absolute value of Divisor Input in the data memory (ASM) or stack (CcA). The input to the routine is in signed m.n format, which means the input ‘x’ is scaled up by \( 2^n \) to represent it in m.n format.

\[
\text{Divisor} = x \times 2^n
\]

2. Scale up the dividend ‘1’ by \( 2^{15} \) and store it in ACCL.

\[
\text{Dividend} = 1 \times 2^{15}
\]

3. Obtain the first 16 quotient bits by executing SUBC 16 times. The format of the quotient which is available in ACCL is given below, it will be in Q(15-n) format:

\[
\frac{1 \times 2^{15}}{x \times 2^n} = \frac{1}{x} \times 2^{15-n}
\]

4. The Remainder of the division will be in ACCH, keep it as such and clear the ACCL so that the remainder is in scaled \( 2^{16} \) format in the ACC.

5. Again execute the SUBC 16 times to obtain another 16 bits of the quotient.

6. Combine the quotient bits, the combined quotient is now in Q(15+16-n) format.

7. Negate the quotient if the input were negative.
QINV2

**Reciprocal (16-bit Precision)**

**Description**
This module computes the Reciprocal of input and provides the 16-bit quotient

**Availability**
This module is available in two interface formats:
1) The direct-mode assembly-only interface (Direct ASM)
2) The C-callable interface version

**Module Properties**

<table>
<thead>
<tr>
<th>Item</th>
<th>ASM Only</th>
<th>C-Callable ASM</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>Code Size</td>
<td>41 words</td>
<td>43 words</td>
<td></td>
</tr>
<tr>
<td>Data RAM</td>
<td>3 words</td>
<td>0 words</td>
<td></td>
</tr>
<tr>
<td>xDAIS ready</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>XDAIS component</td>
<td>No</td>
<td>No</td>
<td>IALG layer not implemented</td>
</tr>
<tr>
<td>Multiple instances</td>
<td>-</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>Reentrancy</td>
<td>No</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Multiple Invocation</td>
<td>Yes</td>
<td>Yes</td>
<td></td>
</tr>
<tr>
<td>Stack usage</td>
<td>-</td>
<td>5 words</td>
<td>Stock grows by 5 words</td>
</tr>
</tbody>
</table>

©Texas Instruments Inc., May 2002
Module Terminal Variables/Functions

<table>
<thead>
<tr>
<th>Item</th>
<th>Name</th>
<th>Description</th>
<th>Format</th>
<th>Range (Hex)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>Inv2_input</td>
<td>Input argument represented in signed m.n format (m+n=16)</td>
<td>Signed</td>
<td>8000h to 7FFFh</td>
</tr>
<tr>
<td>Output</td>
<td>Inv2_output</td>
<td>Output is represented in signed 1+n.15-n format</td>
<td>signed</td>
<td>8000h – 7FFFh</td>
</tr>
</tbody>
</table>

Variable Declaration:
In the system file include the following statements:
```
.ref qinv2 ; function call
.ref inv2_input, inv2_output ; input & output
```

Memory map:
All variables are mapped to an uninitialized named section “qmath”

Example: The following sample code obtains the \( \frac{1}{2} \times 1.25 = 0.123077 \)

Input
Input format is assumed to be in signed 8.8 format
The sample input value 8.125, in signed 8.8 format = \( 8.125 \times 2^8 = 2080 \) (0820h)

Output
Output will be in signed 9.7 format
The output 0.1230769, in 9.7 format = \( 0.123077 \times 2^7 = 000Fh \)

Sample Code
```
.ref qinv2 ; function call reference
.ref inv2_input, inv2_output ; input & output reference
.bss input_var1, 1 ; Reserve 1 word for input
.bss output_var1, 1 ; Reserve 1 word for output
LDP #input_var1
SPLK #0820h, input_var1 ; Store the input 8.125 in 8.8 format
SPM #0 ; Set PM shift bits to “00”
LDP #inv2_input ; Set DP for the module input
BLDD #input_var1, inv2_input ; Pass the value of input to the module
CALL qinv2
LDP #output_var1 ; Set DP for output variables
BLDD #inv2_output, output_var1 ; Output
```
C/C-Callable ASM Interface

Declaration: signed short int qinv2(signed short int x)

Input Format: The argument ‘x’ is a fixed-point number in signed \( m.n \) format. This signed \( m.n \) number representation has a range from \(-2^{m-1} \text{ } (0x8000) \) to \(2^{m-1} - \frac{1}{2^n} \text{ } (0x7fff)\)

Example: The range of fixed-point number in signed 16.0 format \([-32768,32767]\)

Output Format: This function returns the reciprocal of the input argument as fixed-point number in signed \((1+n).(15-n)\) format.

Example: If the input is in 16.0 format, then the output will be in 1.15 format

Example: The following sample code obtains the \( qinv2(8.125) = 0.123077 \)

Input
Input format is assumed to be in signed 8.8 format
The sample input value 8.125, in signed 8.8 format = \( 8.125 \times 2^8 = 2080 \text{ (0820h)} \)

Output
Output will be in signed 9.7 format
The output 0.1230769, in 9.7 format = \( 0.1230769 \times 2^7 = 000Fh \)

Sample Code
```c
#include<qmath.h> /* Header file for fixed point math routine */

void main(void )
{
    signed short int x,y;

    x=0x820; /* 8.125 in 8.8 format = 8.125 \times 2^8 = 2080 (0820h) */
    y=qinv2(x); /* 'y' will have \( qinv2(8.125) = 0.1230769 \text{ in } 9.7 \text{ format } \) */
}

/* 0.1230769 in 9.7 format = 0.1230769 \times 2^7 = 000Fh */
```

©Texas Instruments Inc., May 2002
16 bit Quotient Computation

Fixed-point division is implemented by repeated subtractions executed with SUBC, a special conditional subtract instruction.

Given a 16 bit positive dividend and divisor, the repetition of the SUBC command 16 times produces a 16-bit quotient in the low accumulator and a 16-bit remainder in the high accumulator. For the reciprocal computation dividend is 1, as we wish to find the 1/x of the input ‘x’. Both the dividend and the divisor must be positive when using the SUBC command. The absolute value of the input is obtained before performing the division and the result is negated if the input is negative.

1. Keep the Absolute value of Divisor Input in the data memory (ASM) or stack (CcA). The input to the routine is in signed m.n format, which means the input ‘x’ is scaled up by 2^n to represent it in m.n format
   
   \[ \text{Divisor} = x \times 2^n \]

2. Scale up the dividend ‘1’ by 2^{15} and store it in ACCL
   
   \[ \text{Dividend} = 1 \times 2^{15} \]

3. Obtain the first 16 quotient bits by executing SUBC 16 times. The format of the quotient which is available in ACCL is given below, it will be in Q(15-n) format

   \[ \frac{1 \times 2^{15}}{x \times 2^n} = \frac{1}{x} \times 2^{15-n} \]

4. Negate the quotient if the input were negative