Application Report
SPRA516

TMS320C67xx Divide and Square Root Floating-Point Functions

Syd Poland *Technical Training Organization*

Abstract

The purpose of these Floating Point (FP) functions is to provide the Texas Instruments (TI™) TMS320C67xx Digital Signal Processor (DSP) with fast divide and square root assembly language subroutines that are C callable. The functions include both single-precision (SP) 32-bit and double-precision (DP) 64-bit formats. Subjects covered include:

- ❒ Descriptions
- ❒ Run-time library functions
- ❒ Program timing estimates and performance
- ❒ Assembly (ASM) listings
- ❒ Linear assembly (SA) listings
- ❒ C listings
- ❒ Quadratic formula example

Contents

Description

This section provides information in the following areas:

- Formats
- Special cases
- Rounding
- Conventions for passing input arguments
- Convention for returning answers
- Temporary registers
- Validation

Format for 32 Bits

The single-precision floating-point format used is the 32-bit IEEE-754 standard.

- Bit 31 = sign bit (0 = positive, 1 = negative numbers)
- Bits 30-23 = exponent field as a power of 2 with a bias of 127 (exponent of 1.0)
- Bits 22-0 = fractional absolute mantissa that has an implied 1(bit 23) so that the full mantissa is ***1.mantissa*** when the exponent is non-zero and represents about 7 decimal digits (16,777,215 is maximum)
- Number = $(–1)^{sign}$ * (1.mantissa) * $2^{(exp\ –127)}$ for non-zero exponents
- Smallest FP number = `0x0080_0000`
  with exponent of 1 is 1.175494 x $10^{–38}$ (1.0 x $2^{–126}$)
- Largest FP number = `0x7f7f_ffff`
  with exponent of 254 is 3.402823 x 10^{+38} ($2.0^{–}$ x 2^{+127})

NOTE:
The prefix " **`0x`** " indicates the number is hexadecimal.

Format for 64 Bits

The double-precision floating-point format used is the 64-bit IEEE-754 standard.

- Bit 63 = sign bit (0 = positive, 1 = negative numbers)
- Bits 62-52 = exponent field as a power of 2 with a bias of 1023 (exponent of 1.0)
- Bits 51-0 = fractional absolute mantissa that has an implied 1(bit 52) so that the full mantissa is ***1.mantissa*** when the exponent is non-zero and represents about 16 decimal digits (4,503,599,627,370,495 is maximum)
- Number = $(–1)^{sign}$ * (1.mantissa) * $2^{(exp\ –1023)}$ for non-zero exponents

- Smallest FP number with exponent of 1 is about 2.2×10^{-308} (1.0×2^{-1022})
- Largest FP number with exponent of 2046 is about $1.8 \times 10^{+308}$ ($2.0^{-} \times 2^{+1023}$)

Special Cases

The floating-point representation has unique cases for the following:

- Zero–the entire 32-bits or 64-bits are 0 (this software never returns –0).
- Underflow–occurs when the exponent < 1; the answer returned is 0.
- 32-bit overflow–this occurs when the exponent > 254; one of two possible answers is returned:
  - Positive infinity = `0x7fff_ffff` (exponent and mantissa = 1s)
  - Negative infinity = `0xffff_ffff` (all fields = 1s)
- 64-bit overflow–this occurs when the exponent > 2046; one of two possible answers is returned:
  - Positive infinity = `0x7fff_ffff_ffff_ffff` (exponent and mantissa = 1s)
  - Negative infinity = `0xffff_ffff_ffff_ffff` (all fields = 1s)
- Denormal–zero exponent, non-zero mantissa; denormal FP numbers may be generated by hardware. If the exponent = 0, then the sign and mantissa are returned as zero by software.
- Signaling Not-a-Number (SNaN) is treated as infinity. SNaN has:
  - Sign–0 or 1
  - Exponent–255 for SP; 2047 for DP
  - Mantissa–bit 22 = 0 for SP or bit 51 = 0 for DP; remaining mantissa bits are arbitrary
- Quiet Not-a-Number (QNaN) is treated as infinity. QNaN has:
  - Sign–0 or 1
  - Exponent–255 for SP; 2047 for DP
  - Mantissa–bit 22 = 1 for SP or bit 51 = 1 for DP; remaining mantissa bits are arbitrary
- Infinity–the 'C67xx uses the following format for infinity:
  - Sign–0 or 1
  - Exponent–255 for SP or 2047 for DP
  - Mantissa–all 0s

NOTE:
The infinite value returned by these ‘C67xx functions is an exponent and mantissa of all 1s (see 32-bit overflow and 64-bit overflow bullets above).

Rounding

The rounding mode used here is the rounding mode selected in the **FADCR** (ADD/SUB) and **FMCR** (MPY) control registers as set by the system or user. Options include:

- Round-to-nearest (default)
- Round-to-zero
- Round-to-positive-infinity
- Round-to-negative-infinity

Conventions for Passing Input Arguments

Conventions for passing arguments are described next.

- 32-bit single-precision floating-point input numbers are passed as follows:
  - Argument 1 is in register **A4**
  - Argument 2 is in register **B4** (if needed)
- 64-bit double-precision floating-point input numbers are passed as follows:
  - Argument 1 is in registers **A5:A4**
  - Argument 2 is in registers **B5:B4** (if needed)

Convention for Returning Answers

Conventions for returning answers are described next.

- 32-bit single-precision floating-point output numbers are returned in register **A4**
- 64-bit double-precision floating-point output numbers are returned in registers **A5:A4**

Temporary Registers

In accordance with the C software convention, registers **A0-A9** and **B0-B9** (except **B3** with the return address) are considered to be temporary registers and thus are not saved or restored. Any registers in the range **A10-A15** and **B10-B15** that are used will be saved and restored before returning from the subroutine.

Validation

These routines were tested using the ***rts6701.lib*** version 2.00.

Run-Time Library Functions

The 'C67xx ***rts6701.lib*** functions (from *TMS320C6x Optimizing C Compiler User's Guide*, TI literature number SPRU187C) that have been coded here include:

NOTE:
'C67xx hardware instructions are shown in [square brackets].

- _divd = double-precision (DP) floating-point (FP) division

'C67xx [RCPDP] performs an 8-bit table-look-up of the divisor's reciprocal. Three software Newton-Rhapson iterations are performed to obtain the DP FP quotient.

```
X0 = RCPDP(arg2)              ; get initial TLU value to 8-bits
X1 = X0 * ( 2.0 - arg2 * X0 ) ; 1st iteration to 16-bits
X2 = X1 * ( 2.0 - arg2 * X1 ) ; 2nd iteration to 32-bits
X3 = X2 * ( 2.0 - arg2 * X2 ) ; 3rd iteration to 52-bits
Ans = arg1 * X3               ; generate quotient
```

- _divf = single-precision (SP) floating-point division

'C67xx [RCPSP] performs an 8-bit table-look-up of the divisor's reciprocal. Two software Newton-Rhapson iterations are performed to obtain the SP FP quotient.

```
X0 = RCPSP(arg2)              ; get initial TLU value to 8-bits
X1 = X0 * ( 2.0 - arg2 * X0 ) ; 1st iteration to 16-bits
X2 = X1 * ( 2.0 - arg2 * X1 ) ; 2nd iteration to 24-bits
Ans = arg1 * X2               ; generate quotient
```

- sqrt = double-precision (DP) floating-point square root

'C67xx [RSQRDP] performs an 8-bit table-look-up of the reciprocal of the square root of the input argument. Three software Newton-Rhapson iterations are performed to obtain the DP FP square root.

```
X0 = RSQRDP(arg1)           ; get initial TLU value to 8-bits
V = 0.5 * arg1              ; generate term just once
X1 = X0 * ( 1.5 - V * X0^2 ) ; 1st iteration to 16-bits
X2 = X1 * ( 1.5 - V * X1^2 ) ; 2nd iteration to 32-bits
X3 = X2 * ( 1.5 - V * X2^2 ) ; 3rd iteration to 52-bits
Ans = arg1 * X3             ; generate square root
```

- sqrtf = single-precision (SP) floating-point square root

'C67xx [RSQRSP] performs an 8-bit table-look-up of the reciprocal of the square root of the input argument. Two software Newton-Rhapson iterations are performed to obtain the SP FP square root.

```
X0 = RSQRSP(arg1)           ; get initial TLU value to 8-bits
V = 0.5 * arg1              ; generate term just once
X1 = X0 * ( 1.5 - V * X0^2 ) ; 1st iteration to 16-bits
X2 = X1 * ( 1.5 – V * X1^2 ) ; 2nd iteration to 32-bits
Ans = arg1 * X2             ; generate square root
```

Table 1. Program Timing Estimates and Performance

Name	ASM cycles	Million per sec. at 167 MHz (ASM)	rts6701 cycles vs 2.00	Million per sec. at 167 MHz (rts)	rs6701/ASM ratio
_divd	84	2.0	367	0.5	4.4
_divf	30	5.6	189	0.9	6.3
sqrt	114	1.5	376	0.4	3.3
sqrtf	40	4.2	199	0.8	5.0
Totals	268		1131		
Average		3.3		0.7	4.8

Times listed include only the function execution times; no call overhead time is included. Instructions are on chip (no cache misses). Times assume no memory contentions, no interrupts occur (no ISRs), no DMA activities or Host Port action that would stall the CPU. PUSH and POP will use on-chip SRAM for their stack (if needed).

Double-Precision Floating-Point Division = _divd

The quotient of two input 64-bit FP numbers is generated. Underflow returns zero, overflow returns + or – infinity.

_divd returns the DP FP quotient: ans = arg1 / arg2 (all DP FP)

Single-Precision Floating-Point Division = _divf

The quotient of two input 32-bit FP numbers is generated. Underflow returns zero; overflow returns + or– infinity.

_divf returns the SP FP quotient: ans = arg1 / arg2 (all SP FP)

Double-Precision Floating-Point Square Root = sqrt

The square root of an input 64-bit FP number is generated. Numbers < 0 will have the square root of the magnitude returned.

sqrt returns the DP FP square root: ans = sqrt(arg1) all DP FP

Single-Precision Floating-Point Square Root = sqrtf

The square root of an input 32-bit FP number is generated. Numbers < 0 will have the square root of the magnitude returned.

sqrtf returns the SP FP square root: ans = sqrtf(arg1) all SP FP

Assembly (ASM) Listings

This section contains the hand coded ASM listings and the register allocations used for each function.

❒ _divd = DP floating-point division

'C67xx RCPDP performs an 8-bit table-look-up of the reciprocal of the divisor. Three software Newton-Rhapson iterations are performed to generate the DP FP quotient.

❒ _divf = SP floating-point division

'C67xx RCPSP performs an 8-bit table-look-up of the reciprocal of the divisor. Two software Newton-Rhapson iterations are performed to generate the SP FP quotient.

❒ sqrt = DP floating-point square root

'C67xx RSQRDP performs an 8-bit table-look-up of the reciprocal of the square root of the input argument. Three software Newton-Rhapson iterations are performed to generate the DP FP square root.

❒ sqrtf = SP floating-point square root

'C67xx RSQRSP performs an 8-bit table-look-up of the reciprocal of the square root of the input argument. Two software Newton-Rhapson iterations are performed to generate the SP FP square root.

NOTE:
All labels used in C are generated with an underscore preceding that symbol. For example, label “`sqrt`” in C becomes label “`_sqrt`” in assembly. Likewise if the label already has an underscore in it, a second one will be added. For example, the label “`_divf` ” in C becomes “`_ _divf` ” in assembly.

Double-Precision Floating-Point Division ASM Listing for _divd

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       DIVDP1.asm     Syd Poland    08-08-98        for TMS320C67xx DSPs
;       a5:a4 = a5:a4 / b5:b4
;         ans =  arg1 /  arg2 DP FP divide subroutine

        .text                   ; program memory
        .global _DIVDP,__divd   ; entry labels
        .align  32              ; fetch packet boundary

arg1H   .set    a5      ; input DP FP argument1 register pair
arg1L   .set    a4
arg2H   .set    b5      ; input DP FP argument2 register pair
arg2L   .set    b4
ansH    .set    a5      ; output DP FP answer register pair
ansL    .set    a4
xH      .set    b7      ; x register pair
xL      .set    b6
twoH    .set    a7      ; DP FP 2.0 constant register pair
twoL    .set    a6
tH      .set    b9      ; DP FP temporary register pair
tL      .set    b8
V       .set    b2      ; divide by 0 switch

__divd                                  ; entry in rts6701 library
_DIVDP:                                 ; ans = arg1 / arg2 (all DP FP)
        rcpdp   .S2     arg2H:arg2L, xH:xL              ; x1 = 1/arg2 [8-bits]
||      zero    .L1     twoH                            ; MS word = 0
||      zero    .D1     twoL                            ; LS word = 0

        set     .S1     twoH,30,30,twoH                 ; DP FP 2 (0x4000_0000)

        mpydp   .M2     arg2H:arg2L, xH:xL, tH:tL       ; t = arg2 * x1
||      extu    .S2     arg2H,1,21,V       ; exp2 = 0 ?
        nop             8

        subdp   .L2x    twoH:twoL, tH:tL, tH:tL         ; t = 2.0 - (arg2*x1)
        nop             5

        mpydp   .M2     xH:xL, tH:tL, xH:xL             ; x2 = x1*(2-arg2*x1)
        nop             8

        mpydp   .M2     arg2H:arg2L, xH:xL, tH:tL       ; t = arg2 * x2
        nop             8

        subdp   .L2x    twoH:twoL, tH:tL, tH:tL         ; t = 2.0 - (arg2*x2)
        nop             5

        mpydp   .M2     xH:xL, tH:tL, xH:xL             ; x3 = x2*(2-arg2*x2)
        nop             8

        mpydp   .M2     arg2H:arg2L, xH:xL, tH:tL       ; t = arg2 * x3
        nop             8

        subdp   .L2x    twoH:twoL, tH:tL, tH:tL         ; t = 2.0 - (arg2*x3)
        nop             5

        mpydp   .M2     xH:xL, tH:tL, xH:xL             ; x4 = x3*(2-arg2*x3)
        nop             8

  [V]   mpydp   .M1x    arg1H:arg1L, xH:xL, ansH:ansL   ; ans = arg1 * x4
||[!V]  set     .S1     arg1H,0,30,ansH         ; exp/mant = all 1s (div by 0)
||[!V]  or      .L1     -1,ansL,ansL            ; lower mantissa = all 1s
        nop             3
        b       .S2     b3                      ; normal function return
        nop             5                       ; wait for ans in register pair
        .end
```

Double-Precision Floating-Point Division ASM Registers for _divd

A2		V	B2
A3		return address	B3
A4	arg1L / ansL	arg2L	B4
A5	arg1H / ansH	arg2H	B5
A6	twoL	xL	B6
A7	twoH	xH	B7
A8		tL	B8
A9		tH	B9

Single-Precision Floating-Point Division ASM Listing for _divf

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       DIVSP1.asm     Syd Poland    05-01-98        for TMS320C67xx DSPs
;        a4 =  a4 / b4
;       ans = arg1/arg2 SP FP divide subroutine

         .text                  ; program memory
         .global _DIVSP,__divf   ; entry label
         .align  32              ; fetch packet boundary

arg1    .set    a4      ; input 32-bit floating point argument 1
arg2    .set    b4      ; input 32-bit floating point argument 2
ans     .set    a4      ; output 32-bit floating point answer
xn      .set    b5      ; nth iteration value
two     .set    a3      ; constant 2.0
tmp     .set    b6      ; temp register
V       .set    b0      ; divide by 0 switch (exp2 = 0)

__divf                                  ; entry in rts6701 library

_DIVSP                   ; entry to SP FP divide subroutine

            rcpsp   .S2     arg2,xn        ; x1 = 1/arg2 [8-bits]

            zero    .L1     two            ; two = 0

            mpysp   .M2     arg2,xn,tmp    ; tmp = arg2 * x1

            extu    .S2     arg2,1,24,V    ; is exp2 = 0 ?
            set     .S1     two,30,30,two  ; two = 2.0 in SP FP (0x4000_0000)
            nop             1

            subsp   .L2X    two,tmp,tmp    ; tmp = 2.0 - (arg2*x1)
            nop             3

            mpysp   .M2     xn,tmp,xn      ; x2 = x1*(2 - arg2*x1) [16-bits]
            nop             3

            mpysp   .M2     arg2,xn,tmp    ; tmp = arg2 * x2
            nop             3

            subsp   .L2X    two,tmp,tmp    ; tmp = 2.0 - (arg2*x2)
            nop             3

            mpysp   .M2     xn,tmp,xn      ; x3 = x2*(2 - arg2*x2) [32-bits]
            nop             1
            b       .S2     b3             ; normal return
            nop             1

   [V]      mpysp   .M1     arg1,xn,ans    ; ans = arg1*(x3) where x3 = 1/arg2
|| [!V]     set     .S1     ans,0,31,ans   ; return exp/mant = all 1s (/ 0)
            nop             3              ; wait for answer in register

             .end
```

Single-Precision Floating-Point Division ASM Registers for _divf

A0		V	B0
A1			B1
A2			B2
A3	two	return address	B3
A4	arg1 / ans	arg2	B4
A5		xn	B5
A6		tmp	B6

Double-Precision Floating-Point Square Root ASM Listing for sqrt

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       SQRDP1.asm    Syd Poland    08-08-98   for TMS320C67xx DSPs
;       a5:a4 = a5:a4
;         ans = Square Root (arg1)        DP FP Square Root Subroutine

        .text                    ; program memory
        .global _SQRDP,_sqrt     ; entry labels
        .align  32               ; fetch packet boundary

arg1H   .set    a5      ; input DP FP argument1 register pair
arg1L   .set    a4
ansH    .set    a5      ; output DP FP answer register pair
ansL    .set    a4
VH      .set    a1      ; arg1/2 register pair
VL      .set    a0
KH      .set    b5      ; constant 1.5 register pair
KL      .set    b4
xH      .set    a7      ; xn register pair
xL      .set    a6
tH      .set    a9      ; temp register pair
tL      .set    a8
exp1    .set    a2      ; exp1 = 0 switch

_sqrt                           ; entry in rts6701 library
_SQRDP                          ; entry to DP FP square root function
        clr     .S1     arg1H,31,31,arg1H       ; force sign bit to +
||      zero    .D1     VH                      ; V = 0
        extu    .S1     arg1H,1,21,exp1         ; get exponent 1 field
 [exp1] set     .S1     VH,20,20,VH             ; V = 1 in exponent field
        rsqrdp  .S1     arg1H:arg1L, xH:xL      ; x1 = TLU (1/arg1) [8-bits]
        sub     .L1     arg1H,VH,VH             ; V = arg1 / 2 if exp1 > 0
||      mv      .D1     arg1L,VL
        mpydp   .M1     VH:VL, xH:xL, tH:tL     ; t = (arg1/2) * x1
||      zero    .L2     KH                      ; K = 0
||      zero    .D2     KL
        set     .S2     KH,19,29,KH             ; 1.5 in DP FP (0x3ff8_0000)
        nop             7
        mpydp   .M1     xH:xL, tH:tL, tH:tL     ; t = [(arg1/2)*x1] * x1
        nop             8
        subdp   .L1x    KH:KL, tH:tL, tH:tL     ; t = 1.5-[(arg1/2)*x1]*x1
        nop             5
        mpydp   .M1     xH:xL, tH:tL, xH:xL     ; x2=x1*{1.5-(arg1/2)*(x1*x1)}
        nop             8
        mpydp   .M1     VH:VL, xH:xL, tH:tL     ; t = (arg1/2) * x2
        nop             8
        mpydp   .M1     xH:xL, tH:tL, tH:tL     ; t = [(arg1/2)*x2] * x2
        nop             8
        subdp   .L1x    KH:KL, tH:tL, tH:tL     ; t = 1.5-[(arg1/2)*x2]*x2
        nop             5
        mpydp   .M1     xH:xL, tH:tL, xH:xL     ; x3=x2*{1.5-(arg1/2)*(x2*x2)}
        nop             8
        mpydp   .M1     VH:VL, xH:xL, tH:tL     ; t = (arg1/2) * x3
        nop             8
        mpydp   .M1     xH:xL, tH:tL, tH:tL     ; t = [(arg1/2)*x3] * x3
        nop             8
        subdp   .L1x    KH:KL, tH:tL, tH:tL     ; t = 1.5-[(arg1/2)*x3]*x3
        nop             5
        mpydp   .M1     xH:xL, tH:tL, xH:xL     ; x4=x3*{1.5-(arg1/2)*(x3*x3)}
        nop             8
  [exp1]  mpydp .M1     arg1H:arg1L, xH:xL, ansH:ansL   ; ans=arg1*x4 if exp1>0
||[!exp1] zero  .L1     ansH                    ; ans = 0 if exp1 = 0
||[!exp1] zero  .D1     ansL
        nop             3
        b       .S2     b3                   ; normal return
        nop             5                   ; wait for ans in register pair
```

Double-Precision Floating-Point Square Root ASM Registers for sqrt

A0	VL		B0
A1	VH		B1
A2	exp1		B2
A3		return address	B3
A4	arg1L / ansL	KL	B4
A5	arg1H / ansH	KH	B5
A6	xL		B6
A7	xH		B7
A8	tL		B8
A9	tH		B9

Single-Precision Floating-Point Square Root ASM Listing for sqrtf

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       sqrsp1.asm     Syd Poland      06-22-98
;        a4 = a4
;       ans = square root (arg1) all in SP FP for 'C67xx DSPs

        .global _SQRSP,_sqrtf   ; entry labels
        .text                   ; program section
        .align  32              ; Fetch Packet boundary

arg1    .set    a4      ; input argument SPFP
ans     .set    a4      ; output answer SPFP
xn      .set    a0      ; nth iteration square root reciprocal
exp1    .set    a1      ; input exponent switch
Half    .set    b4      ; SPFP exp of 1 = 0x0080_0000
K15     .set    b5      ; SPFP constant 1.5 = 0x3fc0_0000
tmp     .set    a3      ; temp
wrk     .set    a5      ; work
etc     .set    a2      ; etc.

_sqrtf                  ; entry in rts6701 library
_SQRSP                  ; find square root of arg1 SP FP
        clr     .S1     arg1,31,31,arg1 ; force + sign
||      zero    .L2     Half            ; 0
||      zero    .D2     K15             ; 1.5

        extu    .S1     arg1,1,24,exp1  ; get exponent field
||      set     .S2     Half,23,23,Half ; one in exponent field

        rsqrsp  .S1     arg1,xn         ; X0 = TLU to 8-bits
||      set     .S2     K15,22,29,K15   ; 1.5 in SPFP

  [exp1] sub    .L2x    arg1,Half,Half  ; V=arg1/2 if exp1 is NZ

        mpysp   .M1x    xn,Half,tmp     ; V*X0
        nop             3

        mpysp   .M1     xn,tmp,wrk      ; (V*X0) * X0
        nop             3

        subsp   .L1x    K15,wrk,etc     ; 1.5 - V*X0*X0
        nop             3

        mpysp   .M1     xn,etc,xn       ; X1 = X0*(1.5 - V*X0*X0)
        nop             3

        mpysp   .M1x    xn,Half,tmp     ; V*X1
        nop             3

        mpysp   .M1     xn,tmp,wrk      ; (V*X1) * X1
        nop             3

        subsp   .L1x    K15,wrk,etc     ; 1.5 - V*X1*X1
        nop             3

        mpysp   .M1     xn,etc,xn       ; X2 = X1*(1.5 - V*X1*X1)
        nop             1
        b       .S2     b3              ; normal return
        nop             1
 [exp1]   mpysp  M1     xn,arg1,ans     ; ans = arg1 * X2 if exp1 is NZ
||[!exp1] zero  .D1     ans             ; return 0 if exp1 = 0
          nop           3               ; wait for answer in register
```

Single-Precision Floating-Point Square Root ASM Registers for sqrtf

A0	xn		B0
A1	exp1		B1
A2	etc		B2
A3	tmp	return address	B3
A4	arg1 / ans	Half	B4
A5	wrk	K15	B5

Linear Assembly (SA) Listings

The following sections contain the linear assembly (SA) listings used for each function.

- _divd = double-precision (DP) floating-point (FP) division
- _divf = single-precision (SP) floating-point division
- sqrt = DP floating-point square root
- sqrtf = SP floating-point square root

Double-Precision Floating-Point Division SA Listing for _divd

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       DIVDP.sa     Syd Poland    06-24-98        for TMS320C67xx DSPs
;       a5:a4 = a5:a4 / b5:b4
;         ans =  arg1 /  arg2 DP FP divide subroutine

        .text                   ; program memory
        .global _DIVDP,__divd   ; entry label
        .align  32              ; fetch packet boundary

__divd                                   ; entry in rts6701 library
_DIVDP: .cproc arg1H:arg1L, arg2H:arg2L  ; entry to DP FP divide subroutine

        .reg   xH:xL, twoH:twoL, tH:tL, ansH:ansL, V

        rcpdp           arg2H:arg2L, xH:xL              ; x1 = 1/arg2 [8-bits]
        extu            arg2H,1,21,V                    ; exp2 = 0 ?
        zero            twoH                            ; MS word = 0
        zero            twoL                            ; LS word = 0
        mpydp           arg2H:arg2L, xH:xL, tH:tL       ; t = arg2 * x1
        set             twoH,30,30,twoH                 ; DP FP (0x4000_0000)
        subdp           twoH:twoL, tH:tL, tH:tL         ; t = 2.0 - (arg2*x1)
        mpydp           xH:xL, tH:tL, xH:xL             ; x2 = x1*(2-arg2*x1)
        mpydp           arg2H:arg2L, xH:xL, tH:tL       ; t = arg2 * x2
        subdp           twoH:twoL, tH:tL, tH:tL         ; t = 2.0 - (arg2*x2)
        mpydp           xH:xL, tH:tL, xH:xL             ; x3 = x2*(2-arg2*x2)
        mpydp           arg2H:arg2L, xH:xL, tH:tL       ; t = arg2 * x3
        subdp           twoH:twoL, tH:tL, tH:tL         ; t = 2.0 - (arg2*x3)
        mpydp           xH:xL, tH:tL, xH:xL             ; x4 = x3*(2-arg2*x3)
  [V]   mpydp           arg1H:arg1L, xH:xL, ansH:ansL   ; ans = arg1 * x4
 [!V]   set             arg1H,0,30,ansH         ; exp/mant = all 1s (div by 0)
 [!V]   or              -1,ansL,ansL            ; lower mantissa = all 1s
        .return ansH:ansL
        .endproc
```

Single-Precision Floating-Point Division SA Listing for _divf

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       DIVSP.sa     Syd Poland    05-01-98        for TMS320C67xx DSPs
;        a4 =   a4/b4
;       ans = arg1/arg2 SP FP divide subroutine

        .text                   ; program memory
        .global _DIVSP,__divf   ; entry labels
        .align  32              ; fetch packet boundary

__divf                                  ; entry in rts6701 library
_DIVSP: .cproc arg1, arg2               ; entry to SP FP divide subroutine

        .reg   xn, two, tmp, ans, V

         rcpsp           arg2,xn         ; x1 = 1/arg2 [8-bits]
         extu          arg2,1,24,V     ; exp2 = 0 ?
         zero            two             ; LS halfword = 0
         mpysp           arg2,xn,tmp     ; tmp = arg2 * x1
         set             two,30,30,two   ; two = 2.0 in SP FP (0x4000_0000)
         subsp           two,tmp,tmp     ; tmp = 2.0 - (arg2 * x1)
         mpysp           xn,tmp,xn       ; x2 = x1 * (2 - arg2 * x1) [16-bits]
         mpysp           arg2,xn,tmp     ; tmp = arg2 * x2
         subsp           two,tmp,tmp     ; tmp = 2.0 - (arg2 * x2)
         mpysp           xn,tmp,xn       ; x3 = x2 * (2-arg2 * x2) [32-bits]
  [V]    mpysp           arg1,xn,ans     ; ans = arg1 * (1/arg2) {xn = 1/arg2}
 [!V]    set             arg1,0,30,ans   ; return exp/mant = all ones (div by 0)
         .return ans
         .endproc
```

Double-Precision Floating-Point Square Root SA Listing for sqrt

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       SQRDP.sa     Syd Poland    06-24-98   for TMS320C67xx DSPs
;       a5:a4 = a5:a4
;         ans = Square Root (arg1)        DP FP Square Root Subroutine

        .text                       ; program memory
        .global _SQRDP,_sqrt        ; entry labels
        .align  32                  ; fetch packet boundary

_sqrt                               ; entry in rts6701 library

_SQRDP  .cproc  arg1H:arg1L         ; entry to DP FP square root function

        .reg    VH:VL, ansH:ansL, xH:xL, KH:KL, tH:tL, exp1

        clr             arg1H,31,31,arg1H       ; force sign bit to +
        zero            VH                      ; V = 0
        extu            arg1H,1,21,exp1         ; exp1 =  0 ?
        rsqrdp          arg1H:arg1L, xH:xL      ; x1 = TLU (1/arg1) [8-bits]
        extu            arg1H,1,21,exp1         ; get exponent 1 field
 [exp1] set             VH,20,20,VH             ; V = 1 in exponent field
        sub             arg1H,VH,VH             ; V = arg1 / 2 if exp1 > 0
        mv              arg1L,VL
        mpydp           VH:VL, xH:xL, tH:tL     ; t = (arg1/2) * x1
        zero            KH                      ; K = 0
        zero            KL
        set             KH,19,29,KH             ; 1.5 in DP FP (0x3ff8_0000)
        mpydp           xH:xL, tH:tL, tH:tL     ; t = [(arg1/2)*x1] * x1
        subdp           KH:KL, tH:tL, tH:tL     ; t = 1.5-[(arg1/2)*x1]*x1
        mpydp           xH:xL, tH:tL, xH:xL     ; x2=x1*{1.5-(arg1/2)*(x1*x1)}

        mpydp           VH:VL, xH:xL, tH:tL     ; t = (arg1/2) * x2
        mpydp           xH:xL, tH:tL, tH:tL     ; t = [(arg1/2)*x2] * x2
        subdp           KH:KL, tH:tL, tH:tL     ; t = 1.5-[(arg1/2)*x2]*x2
        mpydp           xH:xL, tH:tL, xH:xL     ; x3=x2*{1.5-(arg1/2)*(x2*x2)}

        mpydp           VH:VL, xH:xL, tH:tL     ; t = (arg1/2) * x3
        mpydp           xH:xL, tH:tL, tH:tL     ; t = [(arg1/2)*x3] * x3
        subdp           KH:KL, tH:tL, tH:tL     ; t = 1.5-[(arg1/2)*x3]*x3
        mpydp           xH:xL, tH:tL, xH:xL     ; x4=x3*{1.5-(arg1/2)*(x3*x3)}

  [exp1] mpydp          arg1H:arg1L, xH:xL, ansH:ansL     ; ans=arg1*x3 (exp1>0)
 [!exp1] zero           ansH                              ; ans = 0 if exp1 = 0
 [!exp1] zero           ansL
        .return         ansH:ansL
        .endproc
```

Single-Precision Floating-Point Square Root SA Listing for sqrtf

```
;       Copyright 1998 by Texas Instruments Inc.  All rights reserved.

;       SQRSP.sa     Syd Poland    05-01-98   for TMS320C67xx DSPs
;        a4 = a4
;       ans = Square Root (arg1)        SP FP Square Root Subroutine

        .text                    ; program memory
        .global _SQRSP,_sqrtf    ; entry labels
        .align  32               ; fetch packet boundary

_sqrtf                                  ; entry in rts6701 library

_SQRSP  .cproc  arg1                    ; entry to SP FP square root function

        .reg    Vd2, ans, xn, K15, tmp, exp1

        clr             arg1,31,31,arg1 ; force sign bit to +
        zero            Vd2             ; Vd2 = 0
        rsqrsp          arg1,xn         ; x1 = square root (1/arg1) [8-bits]
        extu            arg1,1,24,exp1  ; get exponent 1 field = 0 ?
 [exp1] set             Vd2,23,23,Vd2   ; Vd2 = 1 in exponent field if exp1 > 0
        sub             arg1,Vd2,Vd2    ; Vd2 = arg1 / 2
        mpysp           Vd2,xn,tmp      ; tmp = (arg1/2) * x1
        zero            K15             ; K15 = 0
        set             K15,22,29,K15   ; K15 = 1.5 in SP FP (0x3fc0_0000)
        mpysp           xn,tmp,tmp      ; tmp = [(arg1/2)*x1] * x1
        subsp           K15,tmp,tmp     ; tmp = 1.5 - [(arg1/2)*x1] * x1
        mpysp           xn,tmp,xn       ; x2 = x1 * {1.5-(arg1/2)*(x1*x1)}

        mpysp           Vd2,xn,tmp      ; tmp =  (arg1/2) * x2
        mpysp           xn,tmp,tmp      ; tmp = [(arg1/2)*x1] * x1
        subsp           K15,tmp,tmp     ; tmp = 1.5 - [(arg1/2)*x1] * x1
        mpysp           xn,tmp,xn       ; x3 = x2*{1.5-(arg1/2)*(x1*x1)}

  [exp1] mpysp          arg1,xn,ans     ; ans = arg1 * x3 if exp1 > 0
 [!exp1] zero           ans             ; ans = 0 if exp1 = 0
        .return         ans
        .endproc
```

C Source (C) Listings

The following sections contain the C source (C) listings used for each function.

- _divd = Double-precision (DP) floating-point (FP) division
- _divf = Single-precision (SP) floating-point division
- sqrt = DP floating-point square root
- sqrtf = SP floating-point square root

Double-Precision Floating-Point Division C Listing for _divd

```
/*      divdpc.c        Syd Poland  06-24-98 _divd function in C for 'C67xx */

double _divd(double A, double B)
{
double  X0, X1, X2, X3, Y, TWO=2.0 ;

        X0 = _rcpdp(B) ;

        X1 = X0*( TWO - B*X0 ) ;

        X2 = X1*( TWO - B*X1 ) ;

        X3 = X2*( TWO - B*X2 ) ;

        Y = A * X3 ;

        return (Y) ;
}
```

Single-Precision Floating-Point Division C Listing for _divf

```
/*      DIVSPC.c  Syd Poland      6-06-98  _divf function in C for 'C67xx */

float _divf(float A, float B)
{
float BH, X0, X1, X2, Y, TWO=2.0 ;

        X0 = _rcpsp(B) ;

        X1 = X0*( TWO - B*X0 ) ;

        X2 = X1*( TWO - B*X1 ) ;

        Y = A * X2 ;

        return (Y) ;
}
```

Double-Precision Floating-Point Square Root C Listing for sqrt

```
/*      SQRDPC.c  Syd Poland      6-24-98  _sqrt function in C for 'C67xx */

double sqrt(double A)
{
double ZERO=0.0, HALF=0.5, K15=1.5, AH, X0, X1, X2, X3, Y ;


        if (A < ZERO) A = -A ; /* square root of absolute A  */

        AH = A * HALF ;

        X0 = _rsqrdp(A) ;

        X1 = X0 * (K15 - (AH*X0)*X0 ) ;

        X2 = X1 * (K15 - (AH*X1)*X1 );

        X3 = X2 * (K15 - (AH*X2)*X2 );

        Y = A * X3 ;

        return (Y) ;
}
```

Single-Precision Floating-Point Square Root C Listing for sqrtf

```
/*      SQRSPC.c  Syd Poland      6-06-98  _sqrtf function in C for 'C67xx */

float sqrtf(float A)
{
float ZERO=0.0, HALF=0.5, K15=1.5, AH, X0, X1, X2, Y ;

        if (A < ZERO) A = -A ; /* square root of absolute A */

        AH = A * HALF ;

        X0 = _rsqrsp(A) ;

        X1 = X0 * ( K15 - AH*X0*X0 ) ;

        X2 = X1 * ( K15 - AH*X1*X1 ) ;

        Y = A * X2 ;

        return (Y) ;
}
```

Quadratic Formula Example

Solving the Quadratic Formula for one of its two roots will exercise all four floating-point operations plus make use of the square root function. Given the 2nd order polynomial:
$A x^2 + B x + C = 0$

one of the real roots is given by
X1 = [–B + SQRT(B^2 – 4 ***A *** C)] / (2 * A)

assuming that B^2 – 4*A*C is not negative.

Quadratic Formula Performance

Item	ASM cycles	rts6701 cycles vs 2.00	rts / ASM ratio
SP FP root	102	406	4.0
DP FP root	257	785	3.1

Quadratic Formula C Source Listing

```
/*      Copyright 1998 by Texas Instruments Inc.  All rights reserved.  */

/*      test13.c        Syd Poland      06-30-98        */

/*      SP & DP Quadratic Formula for one root on the 'C67xx DSP */

# include <math.h>

int DE3(int,float,float) ;      /* difference table prototype */

float   X1,Y1,A=1.5,B=5.4,C=3.2,Four=4.0,Two=2.0 ;

double  Z1,AA=1.5,BB=5.4,CC=3.2,IV=4.0,II=2.0 ;

int     W, tab3[100], *ptab3, err=0, dif=0 ;

        main ()
{
        ptab3 = &tab3[0] ;                                   /* set ptr */

        X1 = (-B  + sqrtf( B*B  - Four*A*C ) ) / (Two*A) ;      /* SP FP */

        Z1 = (-BB + sqrt (BB*BB - IV*AA*CC ) ) / (II*AA) ;      /* DP FP */

        Y1 = Z1 ;                                            /* DP->SP */

        if (X1 != Y1) DE3(13,X1,Y1) ;                        /* Push ? */
}
```

TI Contact Numbers

INTERNET

TI Semiconductor Home Page
www.ti.com/sc

TI Distributors
www.ti.com/sc/docs/distmenu.htm

PRODUCT INFORMATION CENTERS

Americas

Phone	+1(972) 644-5580
Fax	+1(972) 480-7800
Email	sc-infomaster@ti.com

Europe, Middle East, and Africa

Phone	
Deutsch	+49-(0) 8161 80 3311
English	+44-(0) 1604 66 3399
Español	+34-(0) 90 23 54 0 28
Francais	+33-(0) 1-30 70 11 64
Italiano	+33-(0) 1-30 70 11 67
Fax	+44-(0) 1604 66 33 34
Email	epic@ti.com

Japan

Phone	
International	+81-3-3457-0972
Domestic	0120-81-0026
Fax	
International	+81-3-3457-1259
Domestic	0120-81-0036
Email	pic-japan@ti.com

Asia

Phone	
International	+886-2-23786800
Domestic	
Australia	1-800-881-011
TI Number	-800-800-1450
China	10810
TI Number	-800-800-1450
Hong Kong	800-96-1111
TI Number	-800-800-1450
India	000-117
TI Number	-800-800-1450
Indonesia	001-801-10
TI Number	-800-800-1450
Korea	080-551-2804
Malaysia	1-800-800-011
TI Number	-800-800-1450
New Zealand	000-911
TI Number	-800-800-1450
Philippines	105-11
TI Number	-800-800-1450
Singapore	800-0111-111
TI Number	-800-800-1450
Taiwan	080-006800
Thailand	0019-991-1111
TI Number	-800-800-1450
Fax	886-2-2378-6808
Email	tiasia@ti.com