

APPENDIX A

Sample Source Code Routine

A.1 Source Code for Vector Sin Routine

The following routine is the source code for the vector sin routine. The routine uses minimax polynomial approximations specifically fitted to a 32-bit processor to generate 7.75 digits of accuracy.

```
*****
*
* Copyright (C) 1995, Wideband Computers, Inc.
* All Rights Reserved
*
*****/
```

%Z% %M% %I% %E% %U% WBC

NAME

vsin - computes the sine of the values held within vector [a]. All arguments are assumed to be in radians.

Example: if x = 2.0 then y = sin(x),
y = sin(2.0) = 0.9092974

SYNTAX

```
void vsin ( a, i, c, k, n)

        float dm *a ;      Pointer to input vector [a]
        int i           ;   Vector [a] element stride
        float dm *c ;      Pointer to output vector [c]
        int k           ;   Vector [c] element stride
        int n           ;   Count of floating point elements
```

DESCRIPTION

vsin () computes the sine of the values held within vector a[]]. The return value is stored in output vector a[]]. This reduces reduces the argument to a modulo PI fraction before using a polynomial approximation from Cody and Waite to estimate the sine value. The polynomial used is appropriate for 32 bit computations. In addition the inner loop of coefficients is unrolled to eliminate setting up a loop each time.

NOTES (C40)

MODE1 Truncation Bit (15) for the FIX instruction must be 0.

REGISTER USE:

```
i1      Pointer to a[]
i2      Pointer to b[]
i3      Pointer to Sine Coefficients
m1      Stride of a[]
m2      Stride of b[]
```

```

f1      Scratch for Computations
f2      Scratch for Computations
f3      Scratch for Computations
f4      Holds value of Constant C1
f5      Holds value of 1 / PI for Argument Reduction
f7      Holds value of Constant C2
r9      Number of elements to compute in loop
f8      Holds the value of the sign to apply at the end
}

#include <asm_sprt.h>

.segment /pm seg_pmco ;

.global _vsin ;

_vsin:
    leaf_entry ;                { Entry Macro that is Required }
    save_reg ;

    i1 = r4 ;                    { Get address of a }
    m1 = r8 ;                    { Get the stride of a }
    i2 = r12 ;                   { Get address of c }
    m2 = reads(1) ;             { Get the stride of c }
    r7 = reads(2) ;            { Get number of elements to sum }
    r9 = r7 ;

    i3 = sine_data ;           { Load pointer to data }
    f5 = dm(i3, 1) ;           { Get 1 / PI }
    f4 = dm(i3, 1) ;           { Constant C1 }
    f7 = dm(i3, 1) ;           { Constant C2 }
    lcntr = r9, do lpl until lce ;

sine:

    f8 = 1.0 ;                  { Remember the sign }
    f0 = dm(i1, m1) ;          { Get the next element in a[] }
    f1 = f0 * f5 ;             { Reduce to fractional radians }
    r1 = fix f1 ;
    f2 = float r1 ;

    btst r1 by 0 ;
    if not sz f8 = -f8 ;

    f3 = f2 * f4 ;
    f0 = f0 - f3 ; f3 = f2 * f7 ;    { Fraction f3 }
    f0 = f0 - f3 ;                { Get x**2 }
                                    { Compute P(x**2) }

    f3 = f0 * f0, f1 = dm(i3, 1) ;    { 1st coefficient }

```

```

f2 = f1 * f3, f1 = dm(i3, 1 ) ;           { 2nd coefficient }
f2 = f1 + f2 ;
f2 = f2 * f3, f1 = dm(i3, 1 ) ;         { 3rd coefficient }
f2 = f1 + f2 ;
f2 = f2 * f3, f1 = dm(i3, -3 ) ;        { 4th coefficient }
f2 = f1 + f2 ;
f2 = f2 * f3 ;

f2 = f2 * f0 ;                           { x*P(x**2)      }
f2 = f2 + f0 ;                           { x*P(x**2) + x }
f2 = f2 * f8 ;                           { Get right sign }

lp1:   dm(i2, m2) = f2 ;                   { Store c[]      }

restore_reg ;
leaf_exit ;                               { Exit Macro that is Required }

.endseg ;

.segment /dm seg_dmda ;

.PRECISION = 32 ;

.VAR sine_data[7] = 0.31830988618379067154, {1/PI}
                 3.140625,                 { C1 }
                 9.67653589793e-4,         { C2 }
                 0.2601903036e-5,
                 -0.1980741872e-3,
                 0.8333025139e-2,
                 -0.1666665668 ;

.endseg ;

```