```
// Fast sine and cosine for SHARC ADSP-21364, callable from VisualDSP++ C.
// Simultaneously compute sine and cosine of a given angle.
// Polynomial approximation developed by Flemming Pedersen of CERN.
// This implementation by Lippold Haken of Haken Audio, March 2010, October 2012.
//
// C prototype for this function:
// typedef struct { float sin, cos; } SinCos;
// void sinCos( float fRadians, SinCos *result );
//
// 23 cycles to execute this function:
//    18 cycles for the computations
//     5 cycles for compiler environment overhead.

#include <def21364.h>


     // Data table for sinCos function.
    .section/dm/DOUBLE32 seg_dmda;
    sinCosData:
    .global sinCosData;
    .type sinCosData,STT_OBJECT;
    .var =

         // Integer 3 for "mod 4" operation.
         0x00000003, 0x00000003,

         // 1.0 for neutralizing final cos multiply.
         0x3F800000, 0x3F800000,

         // Coefficients for sin and cos polynomials.
         0xBB96BD89, // SQ7 for sin polynomial   -4.6002309092153379e-003
         0xBCA75707, // CQ6 for cos polynomial   -2.0427240364907607e-002
         0x3DA32F1D, // SQ5 for sin polynomial    7.9679708649230657e-002
         0x3E81D8BD, // CQ4 for cos polynomial    2.5360671639164339e-001
         0xBF255DDD, // SQ3 for sin polynomial   -6.4596348437163809e-001
         0xBF9DE9D1, // CQ2 for cos polynomial   -1.2336979844380824e+000
         0x3FC90FDB, // SQ1 for sin polynomial    1.5707963267948966e+000
         0x3F800000, // CQ0 for cos polynomial    1.0000000000000000e+000

         // Sign adjustment table, indexed by integer quadrant.
         0x3F800000, 0x3F800000,    //  1.0, 1.0
         0x3F800000, 0xBF800000,    //  1.0,-1.0
         0xBF800000, 0xBF800000,    // -1.0,-1.0
         0xBF800000, 0x3F800000;    // -1.0, 1.0

    sinCosData.end:


    .section/pm/DOUBLE32 seg_pmco;
    _sinCos:

    // Registers in VisualDSP environment:
        // M5,M13   = 0
        // M6,M14   = 1
        // M7,M15   = -1
        // F4       = first function argument (input angle)
        // R8       = pointer to address for storing sin,cos result
        // R0,R1,R2,R4,R8,R12,I4,I12,I13,M4 need not be preserved
        // I6,I7    = stack and frame pointers
        //            C calling routine has: "CJUMP (DB); DM(I7,M7)=R2; DM(I7,M7)=PC;"
        //            CJUMP does these operations: "R2=I6, I6=I7"
        //            Before exiting must do RFRAME: "I7=I6, I6=DM(0,I6)"

    // Preamble.
        SF4 = F4;
        BIT SET MODE1 SIMD;

    // Pointer to constants.
        I4 = sinCosData;
```

```
    // Compute integer quadrant, fractional quadrant, fractional quadrant squared.
        // Quadrant "q" = fRadians * 2./Pi
        // Quadrant values (0. .. 4.) correspond to (0. .. 2Pi)
        R1 = 0x3F22F983;                          // F1 = 2./Pi
        F0 = F1 * F4,       I12 = DM(M7,I6);      // I12 is execution return address
        // Integer quadrant = round(q) mod 4    (0..3)
        // Since MODE1 TRUNC bit is zero, FIX implements round-to-nearest.
        R1 = FIX F0,        R2  = DM(I4,2);
        R2 = R1 AND R2,     F12 = DM(I4,2);
        // Fractional quadrant "Xq" = q - round(q)    (-.5 .. .5)
        // Fractional quadrant values (-.5 .. .5) correspond to (-Pi/4..Pi/4)
        F1 = FLOAT R1,      F4 = DM(I4,2);
        F1 = F0 - F1,       M4 = R2;
        // Set SZ if integer quadrant is even.
        R2 = LSHIFT R2 BY 31;
        // Fractional quadrant squared "Xq2" = Xq * Xq
        // Set SF1 = 1.0 to avoid final Xq multiply for cosine polynomial.
        F2 = F1 * F1,       F12 <-> SF1;

    // Registers involved in sin and cos computation:
        // F1     = fractional quadrant Xq
        // SF1    = 1.0
        // F2,SF2 = fractional quadrant squared Xq2
        // F4     = SQ7
        // SF4    = CQ6
        // M4     = integer quadrant
        // SZ     = indicates integer quadrant is even
        // DM(I4) = pairs of cosine and sine coefficients

    // Compute sine polynomial (PEx) and cosine polynomial (PEy).
        // Xq2*SQ7                          Xq2*CQ6
        F0 = F2 * F4,       F4 = DM(I4,2);
        // SQ5+Xq2*SQ7                      CQ4+Xq2*CQ6
        F0 = F0 + F4;
        // Xq2(SQ5+Xq2*SQ7)                 Xq2(CQ4+Xq2*CQ6)
        F0 = F0 * F2,       F4 = DM(I4,2);
        // SQ3+Xq2(SQ5+Xq2*SQ7)             CQ2+Xq2(CQ4+Xq2*CQ6)
        F0 = F0 + F4,       F4 = DM(I4,2);
        // Xq2(SQ3+Xq2(SQ5+Xq2*SQ7))        Xq2(CQ2+Xq2(CQ4+Xq2*CQ6))
        F0 = F0 * F2,       MODIFY(I4,M4);
        // SQ1+Xq2(SQ3+Xq2(SQ5+Xq2*SQ7)))   CQ0+Xq2(CQ2+Xq2(CQ4+Xq2*CQ6)))
        F0 = F0 + F4,       F4 = DM(M4,I4);
        // Xq(SQ1+Xq2(SQ3+Xq2(SQ5+Xq2*SQ7))))     no change on PEy
        F0 = F0 * F1,       I4 = R8;        // I4 is data return address

    // Registers:
        // F0     = sine result so far
        // SF0    = cosine result so far
        // F4,SF4 = +/-1 for inverting sign for quadrant
        // SZ     = set for integer quadrant even

    // Invert and swap sin/cos according to integer quadrant:
    //  quadrant F0      SF0
    //  0 = 00   SIN     COS
    //  1 = 01   COS    -SIN   swap results, invert sine
    //  2 = 10  -SIN    -COS   invert signs
    //  3 = 11  -COS     SIN   swap results, invert cosine
        IF NOT SZ F0 <-> SF0;               // swap sin,cos if odd quadrant
        RFRAME;                             // stack frame management before exit
        JUMP (M14,I12) (DB), F0 = F0 * F4;  // negate sin/cos if needed
        DM(M5,I4) = F0;                     // store sin at I4, cos (SF0) at I4+1
        BIT CLR MODE1 SIMD;                 // end SIMD during JUMP (DB) stall

    ._sinCos.end:
    .global _sinCos;
    .type _sinCos,STT_FUNC;
```