AN2124
Rev. 1, 11/2004

# An 8 × 8 Discrete Cosine Transform on the StarCore™ SC140/SC1400 Cores

By Kim-chyan Gan

The 8x8 discrete cosine transform (DCT) is widely used in image compression algorithm because of its energy compaction for correlated image pixels. The basis function for the DCT is the cosine, a real function that is easy to compute. The DCT is a unitary transform, and the sum of the energy in the transform and spatial domains is the same.

Special, fast algorithms for the DCT have been developed to accommodate the many arithmetic operations involved in implementing the DCT directly. This application note presents an implementation of a fast DCT algorithm for Freescale StarCore™-based DSPs.

**CONTENTS**

# 1 Discrete Cosine Transform Basics

The one-dimensional (1-D) DCT of input sequence u(n) is defined as

$$v(k) = \alpha(k) \sum_{n=0}^{(N-1)} u(n) \cos \frac{(2x+1)k\pi}{2N}, \quad 0 \le k \le N-1 \qquad \textbf{Equation 1}$$

where

$$\alpha(0) = \sqrt{\frac{1}{N}} \qquad \alpha(k) = \sqrt{\frac{2}{N}}, \ 1 \le k \le N-1 \qquad \textbf{Equation 2}$$

The direct implementation of the one-dimensional (1-D) DCT is very computation-intensive; the complexity, expressed as the number of operations *O*, is equal to $N^2$. The formula can be rearranged in even and odd terms to obtain a fast DCT [1] in which *O* = N logN. The fast DCT can be represented by the flow diagram in **Figure 1**. The final value of the last stage in the flow diagram is twice the value of the output.

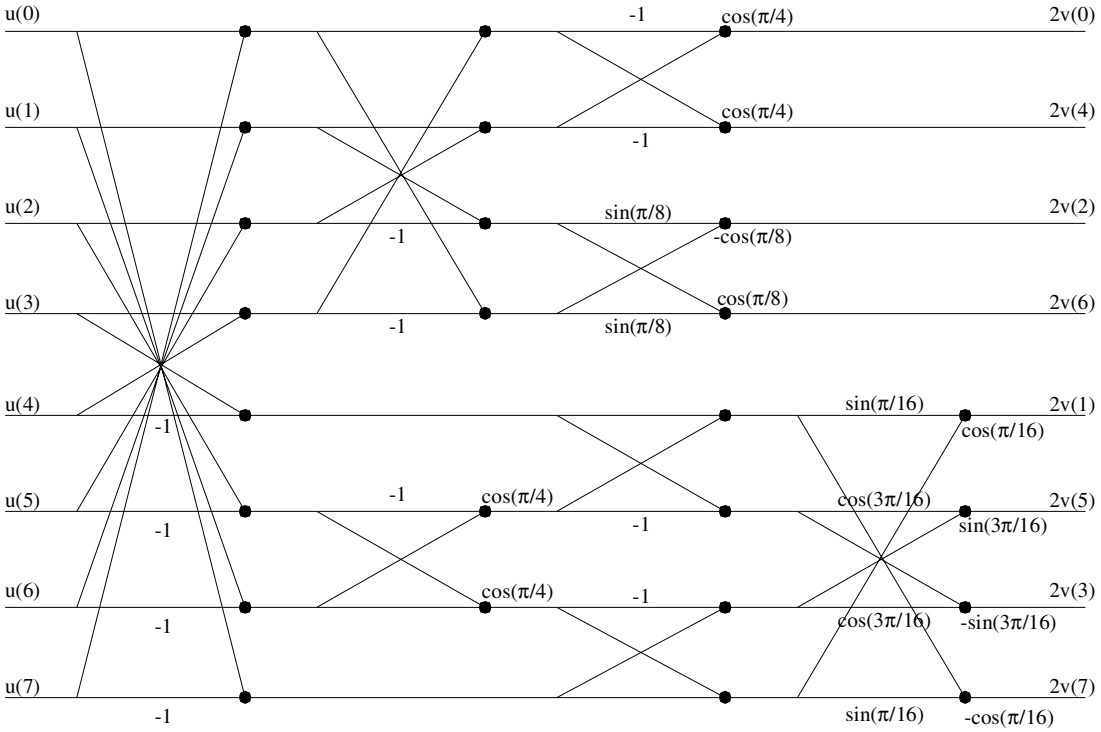


**Figure 1.** 1-D Fast DCT Flow Diagram

An 8x8 DCT can be achieved by applying a fast 8-point DCT to every row followed by every column since the 2-D DCT is a separable unitary transform. The calculation complexity *O* for this algorithm is $N^2$logN. The basis of the 8x8 DCT can be generated by Matlab, as illustrated in and **Figure 2**.

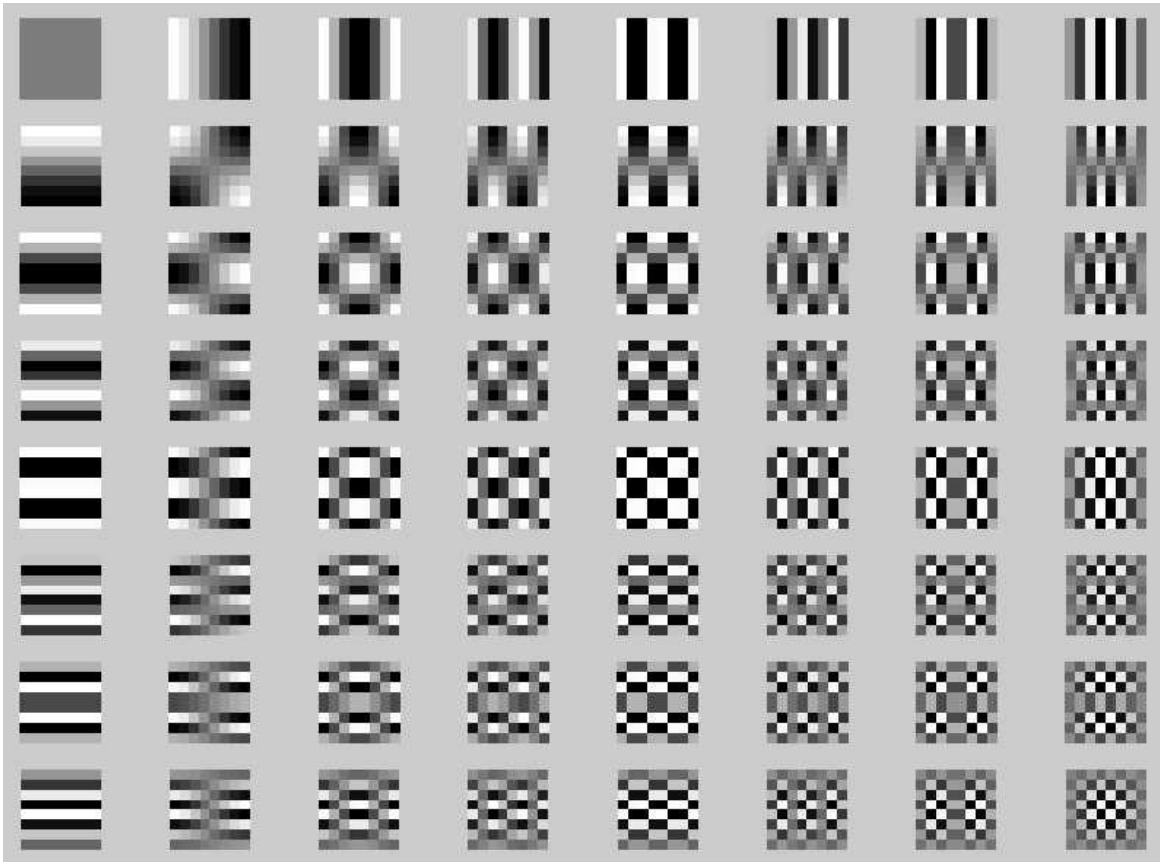

**Figure 2.** 8 × 8 DCT Basis

The Matlab code for **Figure 2** is shown in **Code Listing 1**.

**Code Listing 1.** Matlab Basis of the 8 × 8 DCT

```
N=8
x=(0:N-1)'
C=cos((2*x+1)*x'*pi/(2*N))*sqrt(2/N);
C(:,1)=C(:,1)/sqrt(2)
colormap('gray');
for i=1:N
for j=1:N
aa=C(:,i);
bb=C(:,j)';
X=aa*bb;
subplot(N,N,N*(i-1)+j); imagesc(X); axis off;
end
end
```

The 8 × 8 DCT can also be viewed in the spatial domain as a weighted linear combination of 8 × 8 coefficients. If the input is highly correlated and all the coefficients have the same value, all the transform coefficients have a value of zero except the DC coefficient, and data compression is achieved.

# 2 8 × 8 DCT on StarCore-Based DSPs

There are two passes in the StarCore DCT software. A 1-D DCT is applied to every row in the first pass and every column in the second pass. Within each pass, the 1-D DCT executes eight times.

## 2.1 Data Memory

Data memory is organized in the following blocks:

- a 128-byte passing buffer
- a 128-byte working buffer
- 32 bytes of coefficients

Each of these blocks must be aligned on 16-byte boundaries to use the four-fractional data move instruction. The passing buffer is allocated by the caller, while the working buffer and coefficients are allocated by the DCT routine. A ping-pong buffer approach is used. In the first pass, the passing buffer serves as the input and the working buffer is the output for the 1-D DCT. In the second pass, the working buffer is the input and the passing buffer is the output for the 1-D DCT. **Code Listing 2** shows the data segment of the DCT program.

**Code Listing 2.** Data Segment

```
             align 16
             tmp ds 128
tw           dc $5a82,$7642,$30fc,$5a82,$7d8a,$18f9,$6a6e,$471d
             dc $16a1,$1D90,$0C3F,$5a82,$1f63,$063e,$1a9b,$11c7

; DCT coeffs for 1st pass: cos(π/4), cos(π/8), sin(π/8), cos(π/4),
;                          cos(π/16), sin(π/16), cos(3π/16), sin(3π/16)
; DCT coeffs for 2nd pass: cos(π/4)/4, cos(π/8)/4, sin(π/8)/4, cos(π/4),
                           cos(π/16)/4, sin(π/16)/4, cos(3π/16)/4, sin(3π/16)/4
```

## 2.2 Software Optimization

Optimization techniques applied to increase speed and reduce code size include transposing data, coefficient scaling, collapsing two passes into a do loop, manipulating buffer pointers, pipelining, and circular buffers.

### 2.2.1 Transposed Data

The results of the 1-D DCT are stored in transposed format to enable row memory access for the column 1-D DCT in the second pass. This approach enables the use of the four-fractional data move instruction in the second pass.

### 2.2.2 Coefficient Scaling

The output of the 1-D DCT flow diagram is twice the actual output value. Since each of data is processed twice by the 1-D DCT, the final value should divided by four (shifting right by two bits). After this scaling, rounding is required to obtain the final result. This results in eight additional rounding and eight additional scaling operations to the 1-D DCT inner kernel. These scaling and rounding operations can be omitted by scaling the DCT coefficients of the second pass by four, thereby decreasing the cycle count and increasing the speed of the 8x8 DCT. However, because two sets of DCT coefficients are needed, this technique requires an additional 16-bytes of memory. The error introduced by scaling the coefficients is minimal.

### 2.2.3 Collapsing Two Passes Into a DO Loop

With transposed storage and coefficient scaling, the 1-D DCT inner kernel code is the same for row operations in the first pass and column operations in the second pass. This enables the use of a DO loop, which reduces the size of DCT code.

### 2.2.4 Pointer Usage

Each pass of the 1-D DCT uses one input pointer and two output pointers. The input of 1-D DCT is in normal order but the output is in bit-reverse order. Since the input data is sequential with row access, multiple fractional data move instructions can be used. These instructions require 16-byte alignment of the input data. The StarCore bit-reverse addressing mode is not used because it requires a great deal of initialization overhead and several changes of starting address from column/row to column/row. Instead, two output pointers are used to store the bit-reversed output in normal order. **Figure 1** shows how this is done.

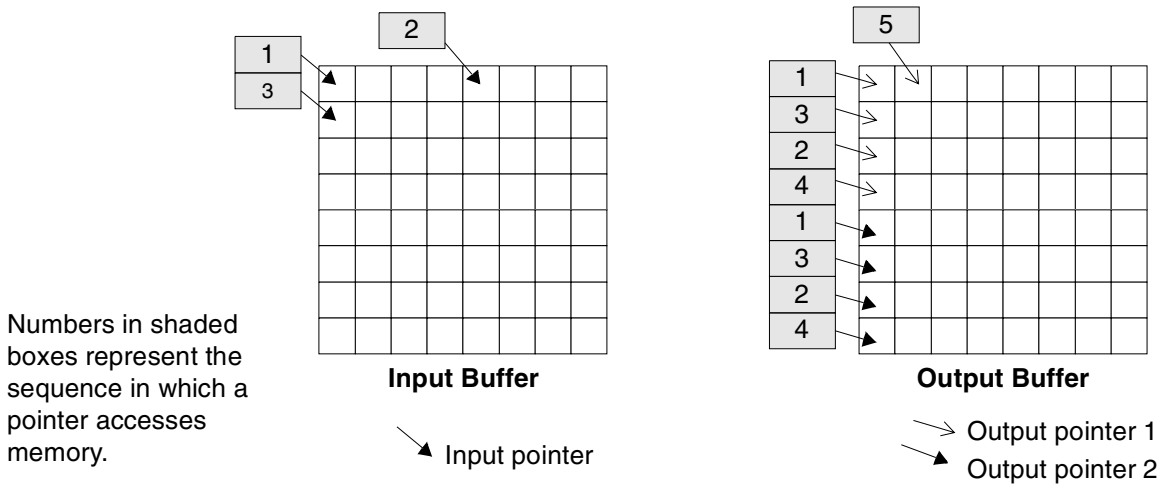


**Figure 1. 1-D DCT Input and Output Pointer Operation**

The input pointer incrementes at each memory access. The output pointer sequence is irregular, so three offset registers (N0–N2) are used to move the output pointer correctly, as shown in **Table 1**.

**Table 1.** Increment Value for Output Pointer Transition

| Output Pointer Transition | Offset | Offset Register |
|---|---|---|
| 1–2 | 16 | N0 |
| 2–3 | –8 | N1 |
| 3–4 | 16 | N0 |
| 4–5 | –23 | N2 |

### 2.2.5 Pipelining

Software pipelining is used to increase efficiency by using the empty slot in an execution set that would otherwise be wasted. The input data for the current loop is read in the previous loop, and the input data read in the current loop is used for the next loop. Four output data words from the current loop are stored in current loop and another four words are stored in next loop.

### 2.2.6 Circular Buffer

A circular buffer is used for the eight DCT coefficients. The buffer is accessed in two read operations, each of which reads four coefficients at a time. The coefficient pointer is simply increased after the first read and decreased after the second read, thereby avoiding the overhead involved in setting up modulo addressing.

From a C code standpoint, the DCT function does in-place-computation. After the function is called, all the input data becomes output data. In order to prevent the input data from being corrupted, which can happen with the in-place computation approach, the DCT function is modified so that it contains both input and output parameters. This is done by modifying the initialization for the second pass so that the output pointer contains the address of desired location.

## 2.3 DCT Code

This section lists and describes the StarCore DCT code, including initialization, four stages of calculations, and re-initialization for the second pass. Note that the d6 and d7 registers are saved by the callee for application binary interface (ABI) compliance. The other data registers are saved by the caller.

### 2.3.1 Initialization

The input pointer (r1), two output pointers (r3 and r4), coefficient pointer (r2), and the offset registers (n0, n1 and n2) are initialized. The outer loop, which loops twice (once for row and once for column operation of the DCT), and the inner loop, which loops eight times, are setup. The `move.4f` instructions are part of the pro-loop that results from software pipelining. Eight input data words are read into registers. **Code Listing 3** shows the initialization routine.

**Code Listing 3.** Initialization

```
            push d6              push d7
            move.l #tw,r2        tfra r0,r1
            move.l #tmp,r5       doen2 #2
            move.w #-8,n1        move.w #-23,n2
            move.w #16,n0        adda #80-2,r5,r4
            dosetup2 dct_outer   adda #16-2,r5,r3
            loopstart2
dct_outer:
            dosetup3 dct_inner   move.4f (r1)+,d0:d1:d2:d3
            doen3 #8             move.4f (r1)+,d4:d5:d6:d7
```

### 2.3.2 First Stage

The first stage consists of eight addition/subtraction operations. The output from the previous loop is stored in this stage because of software pipelining. **Code Listing 4** shows the code of the first DCT stage.

**Code Listing 4.** First Stage

```
[             sub d7,d0,d7
              add d7,d0,d0
              add d6,d1,d1
              sub d6,d1,d6
              moves.f d8,(r3)+n0          ; store previous y(1)
              moves.f d13,(r4)+n0         ; store previous y(5)
]
[             add d5,d2,d2
              sub d5,d2,d5
              add d3,d4,d3
              sub d4,d3,d4
              moves.f d14,(r3)+n2         ; store previous y(3)
              moves.f d9,(r4)+n2          ; store previous y(7)
]
```

### 2.3.3 Second Stage

The second stage of DCT calculations consists of six addition/subtraction and two multiplication operations. The DCT coefficients are read for the use in second and third stages. At the same time, the DCT coefficient pointer is increased to prepare to read next four coefficients. The code for this stage is shown in Code Listing 5.

**Code Listing 5.** Second Stage

```
[             add d0,d3,d0
              sub d3,d0,d3
              sub d5,d6,d5
              add d5,d6,d6
              move.4f (r2)+,d8:d9:d10:d11      ; load coefficients
]

[             add d1,d2,d1
              sub d2,d1,d2
              mpyr d5,d11,d5
              mpyr d6,d11,d6
]
```

### 2.3.4 Third Stage

The third computational stage is shown in **Code Listing 6**. It consists of six addition/subtraction, four multiplication, and two MAC (multiply accumulate) operations. The second set of DCT coefficients is read out for the fourth stage. At the same time, the coefficient pointer is decreased so that it points to the beginning address of the DCT coefficient block. The first four outputs in the third stage are ready to be stored. The first two outputs are stored in this stage. The storing operations may cause some memory contention due to bit-reverse addressing, thus incurring a one-cycle penalty.

**Code Listing 6.** Third Stage

```
[             add d0,d1,d0
              sub d1,d0,d1
              mpy d2,d10,d10
              mpy d3,d10,d11
]

[             mpyr d0,d8,d0
              mpyr d1,d8,d1
              macr -d2,d9,d11
              macr d3,d9,d10
              move.4f (r2)-,d12:d13:d14:d15    ; load coefficients
]

[             add d4,d5,d4
              sub d5,d4,d5
              add d6,d7,d7
              sub d6,d7,d6
              moves.f d0,(r3)+n0               ; store y(0)
              moves.f d1,(r4)+n0               ; store y(4)
]                                              ; may lead stall cycle
```

## 2.3.5 Fourth Stage

In the fourth stage, only the last four points of the third stage are used. The last stage consists of four multiplication and four MAC operations. The input data for the next loop is read. Two output data from the third stage are stored. This stage is shown in **Code Listing 7**.

**Code Listing 7.** Fourth Stage

```
[             mpy d4,d13,d8
              mpy d7,d13,d9
              mpy d5,d14,d13
              mpy d6,d14,d14
              moves.f d10,(r3)+n1        ; store y(2)
              move.4f (r1)+,d0:d1:d2:d3  ; input
]

[             macr -d4,d12,d9
              macr d7,d12,d8
              macr -d5,d15,d14
              macr d6,d15,d13
              moves.f d11,(r4)+n1        ; store y(6)
              move.4f (r1)+,d4:d5:d6:d7  ; input
]
```

## 2.3.6 Second Pass Initialization

After eight executions of the 1-D DCT, the parameters are re-initialized for the second pass. The parameter includes two output pointers, an input pointer, and a coefficient pointer. The two storing instructions are the epi-loop from software pipelining. The code for this stage is shown in **Code Listing 8**.

**Code Listing 8.** Second Pass Initialization

```
moves.f d8,(r3)+n0        moves.f d13,(r4)+n0       ; store y(1) and y(5)
moves.f d14,(r3)+n2       moves.f d9,(r4)+n2        ; store y(3) and y(7)
adda #16-2,r0,r3          adda #80-2,r0,r4          ; initialize output pointer
tfra r5,r1                adda #16,r2,r2            ; initialize input and
                                                    ; coefficient pointers
```

# 3 Conclusion

**Table 2** summarizes the performance of the DCT routine. The metrics include the initialization, DCT kernel, and restoration. The restoration stage includes saving registers for ABI compliance (2 cycles) and returning to the routine (3 cycles).

**Table 2.** DCT Results

| Memory (bytes) | | Cycle Count | |
|---|---|---|---|
| **Code** | **Data** | **Init. + Kernel** | **Restore** |
| 250 | 160 | 176 | 5 |

# 4 References

[1] "A Fast computational Algorithm for the Discrete Cosine Transform," W. A. Chen, C.Harrison, and S.C. Fralick. *IEEE Transactions on Communications*. Vol. COM-25, No. 9, Sept. 1977, pp. 1004–1011.

[2] *Fundamentals of Digital Image Processing*, A. K. Jain. Prentice Hall: 1989.

**NOTES:**

**NOTES:**

## *How to Reach Us:*

**Home Page:**
www.freescale.com

**E-mail:**
support@freescale.com

**USA/Europe or Locations not listed:**
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

**Europe, Middle East, and Africa:**
Freescale Halbleiter Deutschland GMBH
Technical Information Center
Schatzbogen 7
81829 München, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

**Japan:**
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064, Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

**Asia/Pacific:**
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T. Hong Kong
+800 2666 8080

***For Literature Requests Only:***
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com



AN2124
Rev. 1
11/2004

freescale™
semiconductor