Image Rotation Algorithm Series 32000® Graphics Note 4

National Semiconductor
Application Note 528
Dave Rand
May 1988

1.0 INTRODUCTION

Fast image rotation of 90 and 270 degrees is important in printer applications, since both Portrait and Landscape orientation printing may be desired. With a fast image rotation algorithm, only the Portrait orientation fonts need to be stored. This minimizes ROM storage requirements.

This application note shows a fast image rotation algorithm that may be used to rotate an 8 pixel by 8 line image. Larger image sizes may be rotated by successive application of the rotation primitive.

2.0 DESCRIPTION

This Rotate Image algorithm (developed by the Electronic Imaging Group at National Semiconductor) does a very fast 8 by 8 (64 bit) rotation of font data. Note also that this algorithm does not exclusively deal with fonts, but any 64 bit image. Larger images can be rotated by breaking the image down into 8 x 8 segments, and using a 'source warp' constant to index into the source data.

The source data is pointed to by R0 on entry. A 'source warp' is contained in R1, and is added to R0 after each read of the source font. This allows the rotation of 16 by 16, 32 by 32 and larger fonts.

ROTIMG deals with the 8 by 8 destination character as 8 sequential bytes in two registers (R2 and R3), as follows:

Destination Font Matrix

Low Address

1				
2				
3				
4	= R2	4	3	2	1
5	= R3	8	7	6	5
6				
7				
8				

High Address

ROTIMG uses an external table (a pointer to the start of the table is located in register R4) to speed the rotation and to minimize the code. This table consists of 256 64 bit entries, or a total of 2,048 bytes. The table may be located code (PC) or data (SB) relative. The complete table is at the end of this document (see *Figure 1*). A few entries of the table are reproduced above.

Entry	Definition
0	0x00000000 00000000
1	0x00000000 00000001
2	0x00000000 00000100
3	0x00000000 00000101
. . .	
253	0x01010101 01010001
254	0x01010101 01010100
255	0x01010101 01010101

The bytes in the table are standard LSB to MSB format. Since there is no quad-byte assembler pseudo-op (other than LONG, which is floating point), we must reverse the 'double' declaration to get the correct byte ordering, as is shown below:

Entry	Definition
0	double 0,0
1	double 1,0
2	double 256,0
3	double 257,0
. . .	
253	double 16842753,16843009
254	double 0x01010100,0x01010101
255	double 0x01010101,0x01010101

Each byte within each eight byte table entry represents one bit of output data. By indexing into the table, and ORing the table's contents with R2 and R3, we set the destination byte if the corresponding source bit is set. In this manner, the character is rotated.

3.0 IMPLEMENTATION

What we are doing is setting the LS Bit of the destination byte if the source bit corresponding to that byte is set. We then shift the entire 64 bit destination left one bit, and repeat this process until we have set all eight bits, and processed all eight bytes of source information.

The source data for an 8 by 8 character ''>'' appears below:

Character Table for '>'

	Bit Number	Hex Value
	0 1 2 3 4 5 6 7	
Byte	0 0 1 0 0 0 0 0 0	02
	1 0 0 1 0 0 0 0 0	04
	2 0 0 0 1 0 0 0 0	08
	3 0 0 0 0 1 0 0 0	10
	4 0 0 0 0 1 0 0 0	10
	5 0 0 0 1 0 0 0 0	08
	6 0 0 1 0 0 0 0 0	04
	7 0 1 0 0 0 0 0 0	02

RRD-B30M105/Printed in U. S. A.

The ROTIMG algorithm, expressed in 32000 code, appears below:

```
#
#
#Rotate image emulation code
#
#    Inputs:
#         R0 = Source font address
#         R1 = Source font warp
#         R4 = Rotate table address
#
#    Outputs:
#         R2 = Destination font low 4 bytes (lsb->msb, 0 - 3)
#         R3 = Destination font high 4 bytes (lsb->msb, 4 - 7)
#
ROTIMG:   save [r0,r5,r6,r7]       #save registers we will use
          movqd     0,r2           #clear destination font
          movd      r2,r3          #clear high bits of dest.
          movd      r2,r5          #clear high bits of temp.
          addr      8,r6           #deal with 8 bytes of src.
rotlp:    movb      0(r0),r5       #get a byte of source
          addd      r1,r0          #add source warp
          addd      r2,r2          #shift destination left one bit
          addd      r3,r3          #top 32 bits too
          addrd     r4[r5:q],r7    #get pointer to table
          ord       0(r7),r2       #or in low bits
          ord       4(r7),r3       #or in high bits
          acbd      -1,r6,rotlp    #and back for more
          restore   [r0,r5,r6,r7]  #restore registers
          ret       $0             #and return
```

TL/EE/9698–1

Now, let's look at what happens to the data, given the example font of '>'.

Loop #	Source Font	R3	R2	
0	—	00000000	00000000	;0 destination
1	02 hex	00000000	00000100	;first bits in
2	04	00000000	00010200	;next bits in
3	08	00000000	01020400	;and so on
4	10	00000001	02040800	
5	10	00000003	04081000	
6	08	00000006	09102000	
7	04	0000000C	12214000	
8	02	00000018	24428100	;last iteration

Now, arranging this in the appropriate order gives us:

Destination Character Table for '>', 90 degree

	Bit Number	Hex Value
	0 1 2 3 4 5 6 7	
Byte	0 0 0 0 0 0 0 0	00
	1 1 0 0 0 0 0 0 1	81
	2 0 1 0 0 0 0 1 0	42
	3 0 0 1 0 0 1 0 0	24
	4 0 0 0 1 1 0 0 0	18
	5 0 0 0 0 0 0 0 0	00
	6 0 0 0 0 0 0 0 0	00
	7 0 0 0 0 0 0 0 0	00

Destination Character Table for '>', 270 degree

	Bit Number	Hex Value
	0 1 2 3 4 5 6 7	
Byte	0 0 0 0 0 0 0 0	00
	1 0 0 0 0 0 0 0 0	00
	2 0 0 0 0 0 0 0 0	00
	3 0 0 0 1 1 0 0 0	18
	4 0 0 1 0 0 1 0 0	24
	5 0 1 0 0 0 0 1 0	42
	6 1 0 0 0 0 0 0 1	81
	7 0 0 0 0 0 0 0 0	00

Note that by re-ordering the output data, we may rotate 90 or 270 degrees. This may also be accomplished by using a different table (see *Figure 2*).

4.0 TIMING

With unrolled 32000 code, the time for this algorithm is about 588 clocks on the 32016. Subtracting the font read time from this (about 113 clocks), the actual time for rotation is 475 clocks. On the 32332, the time is about 388 clocks. On the 32532, the unrolled loop time is 120–180 clocks, depending on burst mode availability. Repetition of the character data also affects the 32532, due to the data cache. See *Figure 3* for an unrolled code listing.

This table is used for the ROTIMG code. It is 256 entries of 64 bits each (8 bytes * 256 = 2048 bytes). There are two entries per line. This table is used for 90° rotation.

```
rottab1:  .double   0x00000000,0x00000000,0x00000001,0x00000000 ;0,1
          .double   0x00000100,0x00000000,0x00000101,0x00000000 ;2,3
          .double   0x00010000,0x00000000,0x00010001,0x00000000 ;4,5
          .double   0x00010100,0x00000000,0x00010101,0x00000000 ;6,7
          .double   0x01000000,0x00000000,0x01000001,0x00000000 ;...
          .double   0x01000100,0x00000000,0x01000101,0x00000000
          .double   0x01010000,0x00000000,0x01010001,0x00000000
          .double   0x01010100,0x00000000,0x01010101,0x00000000
          .double   0x00000000,0x00000001,0x00000001,0x00000001
          .double   0x00000100,0x00000001,0x00000101,0x00000001
          .double   0x00010000,0x00000001,0x00010001,0x00000001
          .double   0x00010100,0x00000001,0x00010101,0x00000001
          .double   0x01000000,0x00000001,0x01000001,0x00000001
          .double   0x01000100,0x00000001,0x01000101,0x00000001
          .double   0x01010000,0x00000001,0x01010001,0x00000001
          .double   0x01010100,0x00000001,0x01010101,0x00000001
          .double   0x00000000,0x00000100,0x00000001,0x00000100
          .double   0x00000100,0x00000100,0x00000101,0x00000100
          .double   0x00010000,0x00000100,0x00010001,0x00000100
          .double   0x00010100,0x00000100,0x00010101,0x00000100
          .double   0x01000000,0x00000100,0x01000001,0x00000100
          .double   0x01000100,0x00000100,0x01000101,0x00000100
          .double   0x01010000,0x00000100,0x01010001,0x00000100
          .double   0x01010100,0x00000100,0x01010101,0x00000100
          .double   0x00000000,0x00000101,0x00000001,0x00000101
          .double   0x00000100,0x00000101,0x00000101,0x00000101
          .double   0x00010000,0x00000101,0x00010001,0x00000101
          .double   0x00010100,0x00000101,0x00010101,0x00000101
          .double   0x01000000,0x00000101,0x01000001,0x00000101
          .double   0x01000100,0x00000101,0x01000101,0x00000101
          .double   0x01010000,0x00000101,0x01010001,0x00000101
          .double   0x01010100,0x00000101,0x01010101,0x00000101
          .double   0x00000000,0x00010000,0x00000001,0x00010000
          .double   0x00000100,0x00010000,0x00000101,0x00010000
          .double   0x00010000,0x00010000,0x00010001,0x00010000
          .double   0x00010100,0x00010000,0x00010101,0x00010000
          .double   0x01000000,0x00010000,0x01000001,0x00010000
          .double   0x01000100,0x00010000,0x01000101,0x00010000
          .double   0x01010000,0x00010000,0x01010001,0x00010000
          .double   0x01010100,0x00010000,0x01010101,0x00010000
          .double   0x00000000,0x00010001,0x00000001,0x00010001
          .double   0x00000100,0x00010001,0x00000101,0x00010001
          .double   0x00010000,0x00010001,0x00010001,0x00010001
          .double   0x00010100,0x00010001,0x00010101,0x00010001
          .double   0x01000000,0x00010001,0x01000001,0x00010001
          .double   0x01000100,0x00010001,0x01000101,0x00010001
          .double   0x01010000,0x00010001,0x01010001,0x00010001
          .double   0x01010100,0x00010001,0x01010101,0x00010001
          .double   0x00000000,0x00010100,0x00000001,0x00010100
```

TL/EE/9698–2

FIGURE 1

```
.double  0x00000100,0x00010100,0x00000101,0x00010100
.double  0x00010000,0x00010100,0x00010001,0x00010100
.double  0x00010100,0x00010100,0x00010101,0x00010100
.double  0x01000000,0x00010100,0x01000001,0x00010100
.double  0x01000100,0x00010100,0x01000101,0x00010100
.double  0x01010000,0x00010100,0x01010001,0x00010100
.double  0x01010100,0x00010100,0x01010101,0x00010100
.double  0x00000000,0x00010101,0x00000001,0x00010101
.double  0x00000100,0x00010101,0x00000101,0x00010101
.double  0x00010000,0x00010101,0x00010001,0x00010101
.double  0x00010100,0x00010101,0x00010101,0x00010101
.double  0x01000000,0x00010101,0x01000001,0x00010101
.double  0x01000100,0x00010101,0x01000101,0x00010101
.double  0x01010000,0x00010101,0x01010001,0x00010101
.double  0x01010100,0x00010101,0x01010101,0x00010101
.double  0x00000000,0x01000000,0x00000001,0x01000000
.double  0x00000100,0x01000000,0x00000101,0x01000000
.double  0x00010000,0x01000000,0x00010001,0x01000000
.double  0x00010100,0x01000000,0x00010101,0x01000000
.double  0x01000000,0x01000000,0x01000001,0x01000000
.double  0x01000100,0x01000000,0x01000101,0x01000000
.double  0x01010000,0x01000000,0x01010001,0x01000000
.double  0x01010100,0x01000000,0x01010101,0x01000000
.double  0x00000000,0x01000001,0x00000001,0x01000001
.double  0x00000100,0x01000001,0x00000101,0x01000001
.double  0x00010000,0x01000001,0x00010001,0x01000001
.double  0x00010100,0x01000001,0x00010101,0x01000001
.double  0x01000000,0x01000001,0x01000001,0x01000001
.double  0x01000100,0x01000001,0x01000101,0x01000001
.double  0x01010000,0x01000001,0x01010001,0x01000001
.double  0x01010100,0x01000001,0x01010101,0x01000001
.double  0x00000000,0x01000100,0x00000001,0x01000100
.double  0x00000100,0x01000100,0x00000101,0x01000100
.double  0x00010000,0x01000100,0x00010001,0x01000100
.double  0x00010100,0x01000100,0x00010101,0x01000100
.double  0x01000000,0x01000100,0x01000001,0x01000100
.double  0x01000100,0x01000100,0x01000101,0x01000100
.double  0x01010000,0x01000100,0x01010001,0x01000100
.double  0x01010100,0x01000100,0x01010101,0x01000100
.double  0x00000000,0x01000101,0x00000001,0x01000101
.double  0x00000100,0x01000101,0x00000101,0x01000101
.double  0x00010000,0x01000101,0x00010001,0x01000101
.double  0x00010100,0x01000101,0x00010101,0x01000101
.double  0x01000000,0x01000101,0x01000001,0x01000101
.double  0x01000100,0x01000101,0x01000101,0x01000101
.double  0x01010000,0x01000101,0x01010001,0x01000101
.double  0x01010100,0x01000101,0x01010101,0x01000101
.double  0x00000000,0x01010000,0x00000001,0x01010000
.double  0x00000100,0x01010000,0x00000101,0x01010000
.double  0x00010000,0x01010000,0x00010001,0x01010000
.double  0x00010100,0x01010000,0x00010101,0x01010000
.double  0x01000000,0x01010000,0x01000001,0x01010000
.double  0x01000100,0x01010000,0x01000101,0x01010000
.double  0x01010000,0x01010000,0x01010001,0x01010000
.double  0x01010100,0x01010000,0x01010101,0x01010000
```

TL/EE/9698–3

FIGURE 1 (Continued)

```
.double   0x00000000,0x01010001,0x00000001,0x01010001
.double   0x00000100,0x01010001,0x00000101,0x01010001
.double   0x00010000,0x01010001,0x00010001,0x01010001
.double   0x00010100,0x01010001,0x00010101,0x01010001
.double   0x01000000,0x01010001,0x01000001,0x01010001
.double   0x01000100,0x01010001,0x01000101,0x01010001
.double   0x01010000,0x01010001,0x01010001,0x01010001
.double   0x01010100,0x01010001,0x01010101,0x01010001
.double   0x00000000,0x01010100,0x00000001,0x01010100
.double   0x00000100,0x01010100,0x00000101,0x01010100
.double   0x00010000,0x01010100,0x00010001,0x01010100
.double   0x00010100,0x01010100,0x00010101,0x01010100
.double   0x01000000,0x01010100,0x01000001,0x01010100
.double   0x01000100,0x01010100,0x01000101,0x01010100
.double   0x01010000,0x01010100,0x01010001,0x01010100
.double   0x01010100,0x01010100,0x01010101,0x01010100
.double   0x00000000,0x01010101,0x00000001,0x01010101
.double   0x00000100,0x01010101,0x00000101,0x01010101
.double   0x00010000,0x01010101,0x00010001,0x01010101
.double   0x00010100,0x01010101,0x00010101,0x01010101
.double   0x01000000,0x01010101,0x01000001,0x01010101
.double   0x01000100,0x01010101,0x01000101,0x01010101 ;250,251
.double   0x01010000,0x01010101,0x01010001,0x01010101 ;252,253
.double   0x01010100,0x01010101,0x01010101,0x01010101 ;254,255
```

TL/EE/9698–4

FIGURE 1 (Continued)

This table is used for the ROTIMG code. It is 256 entries of 64 bits each (8 bytes * 256 = 2048 bytes). There are two entries per line. This gives a 270° rotation.

```
rottab2:  .double   0x00000000,0x00000000,0x00000000,0x01000000
          .double   0x00000000,0x00010000,0x00000000,0x01010000
          .double   0x00000000,0x00000100,0x00000000,0x01000100
          .double   0x00000000,0x00010100,0x00000000,0x01010100
          .double   0x00000000,0x00000001,0x00000000,0x01000001
          .double   0x00000000,0x00010001,0x00000000,0x01010001
          .double   0x00000000,0x00000101,0x00000000,0x01000101
          .double   0x00000000,0x00010101,0x00000000,0x01010101
          .double   0x01000000,0x00000000,0x01000000,0x01000000
          .double   0x01000000,0x00010000,0x01000000,0x01010000
          .double   0x01000000,0x00000100,0x01000000,0x01000100
          .double   0x01000000,0x00010100,0x01000000,0x01010100
          .double   0x01000000,0x00000001,0x01000000,0x01000001
          .double   0x01000000,0x00010001,0x01000000,0x01010001
          .double   0x01000000,0x00000101,0x01000000,0x01000101
          .double   0x01000000,0x00010101,0x01000000,0x01010101
          .double   0x00010000,0x00000000,0x00010000,0x01000000
          .double   0x00010000,0x00010000,0x00010000,0x01010000
          .double   0x00010000,0x00000100,0x00010000,0x01000100
          .double   0x00010000,0x00010100,0x00010000,0x01010100
          .double   0x00010000,0x00000001,0x00010000,0x01000001
          .double   0x00010000,0x00010001,0x00010000,0x01010001
          .double   0x00010000,0x00000101,0x00010000,0x01000101
          .double   0x00010000,0x00010101,0x00010000,0x01010101
```

TL/EE/9698–5

FIGURE 2

```
.double  0x01010000,0x00000000,0x01010000,0x01000000
.double  0x01010000,0x00010000,0x01010000,0x01010000
.double  0x01010000,0x00000100,0x01010000,0x01000100
.double  0x01010000,0x00010100,0x01010000,0x01010100
.double  0x01010000,0x00000001,0x01010000,0x01000001
.double  0x01010000,0x00010001,0x01010000,0x01010001
.double  0x01010000,0x00000101,0x01010000,0x01000101
.double  0x01010000,0x00010101,0x01010000,0x01010101
.double  0x00000100,0x00000000,0x00000100,0x01000000
.double  0x00000100,0x00010000,0x00000100,0x01010000
.double  0x00000100,0x00000100,0x00000100,0x01000100
.double  0x00000100,0x00010100,0x00000100,0x01010100
.double  0x00000100,0x00000001,0x00000100,0x01000001
.double  0x00000100,0x00010001,0x00000100,0x01010001
.double  0x00000100,0x00000101,0x00000100,0x01000101
.double  0x00000100,0x00010101,0x00000100,0x01010101
.double  0x01000100,0x00000000,0x01000100,0x01000000
.double  0x01000100,0x00010000,0x01000100,0x01010000
.double  0x01000100,0x00000100,0x01000100,0x01000100
.double  0x01000100,0x00010100,0x01000100,0x01010100
.double  0x01000100,0x00000001,0x01000100,0x01000001
.double  0x01000100,0x00010001,0x01000100,0x01010001
.double  0x01000100,0x00000101,0x01000100,0x01000101
.double  0x01000100,0x00010101,0x01000100,0x01010101
.double  0x00010100,0x00000000,0x00010100,0x01000000
.double  0x00010100,0x00010000,0x00010100,0x01010000
.double  0x00010100,0x00000100,0x00010100,0x01000100
.double  0x00010100,0x00010100,0x00010100,0x01010100
.double  0x00010100,0x00000001,0x00010100,0x01000001
.double  0x00010100,0x00010001,0x00010100,0x01010001
.double  0x00010100,0x00000101,0x00010100,0x01000101
.double  0x00010100,0x00010101,0x00010100,0x01010101
.double  0x01010100,0x00000000,0x01010100,0x01000000
.double  0x01010100,0x00010000,0x01010100,0x01010000
.double  0x01010100,0x00000100,0x01010100,0x01000100
.double  0x01010100,0x00010100,0x01010100,0x01010100
.double  0x01010100,0x00000001,0x01010100,0x01000001
.double  0x01010100,0x00010001,0x01010100,0x01010001
.double  0x01010100,0x00000101,0x01010100,0x01000101
.double  0x01010100,0x00010101,0x01010100,0x01010101
.double  0x00000001,0x00000000,0x00000001,0x01000000
.double  0x00000001,0x00010000,0x00000001,0x01010000
.double  0x00000001,0x00000100,0x00000001,0x01000100
.double  0x00000001,0x00010100,0x00000001,0x01010100
.double  0x00000001,0x00000001,0x00000001,0x01000001
.double  0x00000001,0x00010001,0x00000001,0x01010001
.double  0x00000001,0x00000101,0x00000001,0x01000101
.double  0x00000001,0x00010101,0x00000001,0x01010101
.double  0x01000001,0x00000000,0x01000001,0x01000000
.double  0x01000001,0x00010000,0x01000001,0x01010000
.double  0x01000001,0x00000100,0x01000001,0x01000100
.double  0x01000001,0x00010100,0x01000001,0x01010100
.double  0x01000001,0x00000001,0x01000001,0x01000001
.double  0x01000001,0x00010001,0x01000001,0x01010001
.double  0x01000001,0x00000101,0x01000001,0x01000101
```

TL/EE/9698–6

FIGURE 2 (Continued)

```
.double  0x01000001,0x00010101,0x01000001,0x01010101
.double  0x00010001,0x00000000,0x00010001,0x01000000
.double  0x00010001,0x00010000,0x00010001,0x01010000
.double  0x00010001,0x00000100,0x00010001,0x01000100
.double  0x00010001,0x00010100,0x00010001,0x01010100
.double  0x00010001,0x00000001,0x00010001,0x01000001
.double  0x00010001,0x00010001,0x00010001,0x01010001
.double  0x00010001,0x00000101,0x00010001,0x01000101
.double  0x00010001,0x00010101,0x00010001,0x01010101
.double  0x01010001,0x00000000,0x01010001,0x01000000
.double  0x01010001,0x00010000,0x01010001,0x01010000
.double  0x01010001,0x00000100,0x01010001,0x01000100
.double  0x01010001,0x00010100,0x01010001,0x01010100
.double  0x01010001,0x00000001,0x01010001,0x01000001
.double  0x01010001,0x00010001,0x01010001,0x01010001
.double  0x01010001,0x00000101,0x01010001,0x01000101
.double  0x01010001,0x00010101,0x01010001,0x01010101
.double  0x00000101,0x00000000,0x00000101,0x01000000
.double  0x00000101,0x00010000,0x00000101,0x01010000
.double  0x00000101,0x00000100,0x00000101,0x01000100
.double  0x00000101,0x00010100,0x00000101,0x01010100
.double  0x00000101,0x00000001,0x00000101,0x01000001
.double  0x00000101,0x00010001,0x00000101,0x01010001
.double  0x00000101,0x00000101,0x00000101,0x01000101
.double  0x00000101,0x00010101,0x00000101,0x01010101
.double  0x01000101,0x00000000,0x01000101,0x01000000
.double  0x01000101,0x00010000,0x01000101,0x01010000
.double  0x01000101,0x00000100,0x01000101,0x01000100
.double  0x01000101,0x00010100,0x01000101,0x01010100
.double  0x01000101,0x00000001,0x01000101,0x01000001
.double  0x01000101,0x00010001,0x01000101,0x01010001
.double  0x01000101,0x00000101,0x01000101,0x01000101
.double  0x01000101,0x00010101,0x01000101,0x01010101
.double  0x00010101,0x00000000,0x00010101,0x01000000
.double  0x00010101,0x00010000,0x00010101,0x01010000
.double  0x00010101,0x00000100,0x00010101,0x01000100
.double  0x00010101,0x00010100,0x00010101,0x01010100
.double  0x00010101,0x00000001,0x00010101,0x01000001
.double  0x00010101,0x00010001,0x00010101,0x01010001
.double  0x00010101,0x00000101,0x00010101,0x01000101
.double  0x00010101,0x00010101,0x00010101,0x01010101
.double  0x01010101,0x00000000,0x01010101,0x01000000
.double  0x01010101,0x00010000,0x01010101,0x01010000
.double  0x01010101,0x00000100,0x01010101,0x01000100
.double  0x01010101,0x00010100,0x01010101,0x01010100
.double  0x01010101,0x00000001,0x01010101,0x01000001
.double  0x01010101,0x00010001,0x01010101,0x01010001
.double  0x01010101,0x00000101,0x01010101,0x01000101
.double  0x01010101,0x00010101,0x01010101,0x01010101
```

TL/EE/9698–7

FIGURE 2 (Continued)

The following is an unrolled version of the rotate image algorithm. For the NS32532, the address computation, currently done with a separate addr instruction, may be done with the ORD instruction. This makes the execution time slightly faster.

```
#
#
#Rotate image emulation code
#
#    Inputs:
#         R0 = Source font address
#         R1 = Source font warp
#         R4 = Rotate table address
#
#    Outputs:
#         R2 = Destination font low 4 bytes (lsb->msb, 0 - 3)
#         R3 = Destination font high 4 bytes (lsb->msb, 4 - 7)
#
ROTIMG:
          movqd     0,r2           #clear destination font
          movd      r2,r3          #clear high bits of dest.
          movd      r2,r5          #clear high bits of temp.
          movb      0(r0),r5       #get a byte of source
          addd      r1,r0          #add source warp
          addd      r2,r2          #shift destination left one bit
          addd      r3,r3          #top 32 bits too
          addr      r4[r5:q],r6    #get pointer to table
          ord       0(r6),r2       #or in low bits
          ord       4(r6),r3       #or in high bits
          movb      0(r0),r5       #get a byte of source
          addd      r1,r0          #add source warp
          addd      r2,r2          #shift destination left one bit
          addd      r3,r3          #top 32 bits too
          addr      r4[r5:q],r6    #get pointer to table
          ord       0(r6),r2       #or in low bits
          ord       4(r6),r3       #or in high bits
          movb      0(r0),r5       #get a byte of source
          addd      r1,r0          #add source warp
          addd      r2,r2          #shift destination left one bit
          addd      r3,r3          #top 32 bits too
          addr      r4[r5:q],r6    #get pointer to table
          ord       0(r6),r2       #or in low bits
          ord       4(r6),r3       #or in high bits
          movb      0(r0),r5       #get a byte of source
          addd      r1,r0          #add source warp
          addd      r2,r2          #shift destination left one bit
          addd      r3,r3          #top 32 bits too
          addr      r4[r5:q],r6    #get pointer to table
          ord       0(r6),r2       #or in low bits
          ord       4(r6),r3       #or in high bits
          movb      0(r0),r5       #get a byte of source
          addd      r1,r0          #add source warp
```

TL/EE/9698–8

FIGURE 3

```
addd    r2,r2        #shift destination left one bit
addd    r3,r3        #top 32 bits too
addr    r4[r5:q],r6  #get pointer to table
ord     0(r6),r2     #or in low bits
ord     4(r6),r3     #or in high bits
movb    0(r0),r5     #get a byte of source
addd    r1,r0        #add source warp
addd    r2,r2        #shift destination left one bit
addd    r3,r3        #top 32 bits too
addr    r4[r5:q],r6  #get pointer to table
ord     0(r6),r2     #or in low bits
ord     4(r6),r3     #or in high bits
movb    0(r0),r5     #get a byte of source
addd    r1,r0        #add source warp
addd    r2,r2        #shift destination left one bit
addd    r3,r3        #top 32 bits too
addr    r4[r5:q],r6  #get pointer to table
ord     0(r6),r2     #or in low bits
ord     4(r6),r3     #or in high bits
movb    0(r0),r5     #get a byte of source
addd    r1,r0        #add source warp
addd    r2,r2        #shift destination left one bit
addd    r3,r3        #top 32 bits too
addr    r4[r5:q],r6  #get pointer to table
ord     0(r6),r2     #or in low bits
ord     4(r6),r3     #or in high bits
ret     $0           #and return
```

TL/EE/9698–9

FIGURE 3 (Continued)

Lit. # 100528

LIFE SUPPORT POLICY

NATIONAL'S PRODUCTS ARE NOT AUTHORIZED FOR USE AS CRITICAL COMPONENTS IN LIFE SUPPORT DEVICES OR SYSTEMS WITHOUT THE EXPRESS WRITTEN APPROVAL OF THE PRESIDENT OF NATIONAL SEMICONDUCTOR CORPORATION. As used herein:

1. Life support devices or systems are devices or systems which, (a) are intended for surgical implant into the body, or (b) support or sustain life, and whose failure to perform, when properly used in accordance with instructions for use provided in the labeling, can be reasonably expected to result in a significant injury to the user.
2. A critical component is any component of a life support device or system whose failure to perform can be reasonably expected to cause the failure of the life support device or system, or to affect its safety or effectiveness.

National Semiconductor Corporation
1111 West Bardin Road
Arlington, TX 76017
Tel: 1(800) 272-9959
Fax: 1(800) 737-7018

National Semiconductor Europe
Fax: (+49) 0-180-530 85 86
Email: cnjwge@tevm2.nsc.com
Deutsch Tel: (+49) 0-180-530 85 85
English Tel: (+49) 0-180-532 78 32
Français Tel: (+49) 0-180-532 93 58
Italiano Tel: (+49) 0-180-534 16 80

National Semiconductor Hong Kong Ltd.
13th Floor, Straight Block,
Ocean Centre, 5 Canton Rd.
Tsimshatsui, Kowloon
Hong Kong
Tel: (852) 2737-1600
Fax: (852) 2736-9960

National Semiconductor Japan Ltd.
Tel: 81-043-299-2309
Fax: 81-043-299-2408