TMS320 DSP
Number 3
DESIGNER'S NOTEBOOK
TEXAS INSTRUMENTS

Bit-reversed Addressing Without Data Alignment on the 'C3x

Contributed by Tim Grady

Design Problem

Bit-reversed addressing mode requires that the n-element array be aligned on an n-word boundary. When n is large, this may result in a large "hole" in the memory map. To use memory more efficiently, a technique to use bit-reversed addressing without data alignment is required.

Solution

AR2 points to the data.

AR1 is initialized to 0 and becomes an offset into the array. Bit-reversed addressing mode is used to modify AR1. Figure 1 shows an assembly language version. Figure 2 shows a C version which uses in-line assembly to permit bit-reversed addressing.

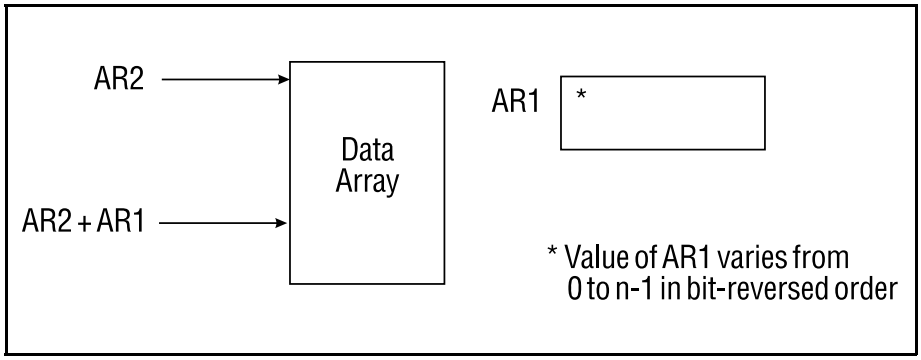

Figure 1. Solution diagram

(continued on next page)

```
            .data
table       .word       8,9,10,11,12,13,14,15
taddr       .word       table
            .text
            .global     _main
_main       ldp         taddr
            ldi  @taddr,ar2             ; pointer to array
            ldi  4,ir0                  ; 1/2 array size for bit-rev addressing
            ldi  0,ar1                  ; first adress in bit-rev list
            ldi  7,rc
            rptb endloop
            ldi  ar1,ir1                ; put new offset into index register
                                        ; This inst may also be put in parallel if
                                        ; the right application comes along.
endloop     ldi  *+ar2(ir1),r0          ; r0 holds array elements one at at time
                                        ; so that results can be observed
          || ldi  *ar1++(ir0)B,r7; calculate next address in parallel
                                        ; r7 is a dummy variable to allow paral ops
            rets
```

Figure 2. Assembly code

```
int x[15]= {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15};
int *y=(int *)&x;
int m;
main()
{
   int i;
   y += 7; /* start with non-aligned array element */
   asm("   ldi @_y,ar0");              /* ar0 points to array */
   asm("   ldi 0,ar2");                /* index for bit-rev   */
   asm("   ldi 4,ir0");                /* set up for bit-rev  */

   for(i=0;i<8;i++)
   {
      asm("    ldi    ar2,ir1");              /* load index of array */
      asm("    ldi    *+ar0(ir1),r7");        /* traverse */
      asm(" || ldi    *ar2++(ir0)b,r6");      /* array with */
      asm("    sti    r7,@_m");               /* bit-rev offset */
   }
}
```

Figure 3. C code