TMS320 DSP
DESIGNER'S NOTEBOOK
Number 70

Parity Generation on the TMS320C54x

Contributed by David Nerge

Design Problem

How can I use the TMS320C54x DSPs to generate parity to ensure data transmission has not been corrupted?

Solution

The Need for Parity Generation

In many applications, simple parity checks are often introduced to provide some minimal error detection capabilities when the transmission channels are less than perfect. The 8-bit data bus in personal computers is protected with the addition of a ninth parity bit. Serial asynchronous data communications frequently embed an extra parity bit in the data which is to be transmitted. Several of the digital cellular phones utilize simple parity checks to protect data transmission over a less than robust channel. Although it is far from an optimal means of ensuring transmission errors are detected, parity tends to be fairly simple to implement in either hardware or software, and introduces minimal overhead in the transmitted message.

What it is

Given a number of bits which are to be transmitted, parity is said to be even if the total number of bits which are logical 'one' is an even number. Otherwise, parity is odd. The value assigned to the even parity bit is logical zero: a logical one if parity is odd. The generated parity bit is frequently appended to the original data bits for subsequent transmission.

The total number of bits which are "1" is three: Three is an odd number, so the parity is odd.

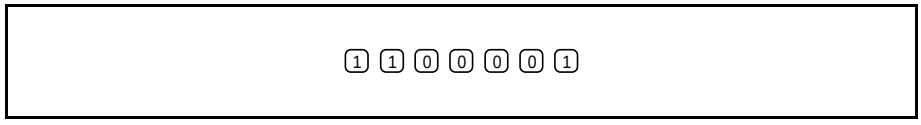

Figure 1. The seven-bit example

Some Examples

Counting Ones

A simple method of determining the value of the parity bit is to count the number of ones in all the bits which are to be transmitted. This can be implemented in software by setting up a loop which tests each bit, and if it's a one, a counter is incremented. This is simple and effective, but also slow. If more bits exist than can be

stored in a single word or byte, then after testing 8 or 16 bits/word (either by shift and test or mask and test), an additional test must be included to index to the next word. When the loop has been completed, the counter is examined. If the least significant bit of the counter is zero, then the total number of ones is even, and parity is even. Else, parity is odd.

```
A pseudo code fragment:
     initialize ones counter to zero
     initialize word counter
     loop1:   get next word containing data bits
     loop2:   shift or mask to test one of X bits in word
              if bit is one, then increment counter
              decrement loop counter
              if loop counter is non zero, then repeat loop2
     if this is not the last word, then get next word and repeat
     loop1
     else, done: check least significant bit in counter to
     determine final parity value
```

Table Lookup

The value of the word containing the data bits may be used as an index into a lookup table to determine the resulting parity. Unfortunately, for longer words, the table tends to be a bit wide, and a bit large. The input word may be subdivided into smaller words, and recursively applied to a smaller lookup table. The tradeoff with this approach is that of table size versus speed and parsing overhead.

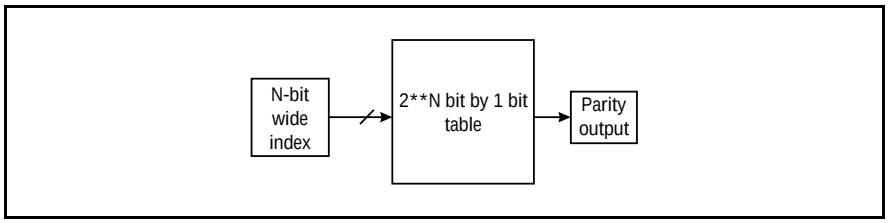

Figure 2. Table lookup

Parity Tree

Hardware designers make use of a parity tree which consists of cascaded stages of two input exclusive or gates as shown in Figure 3.

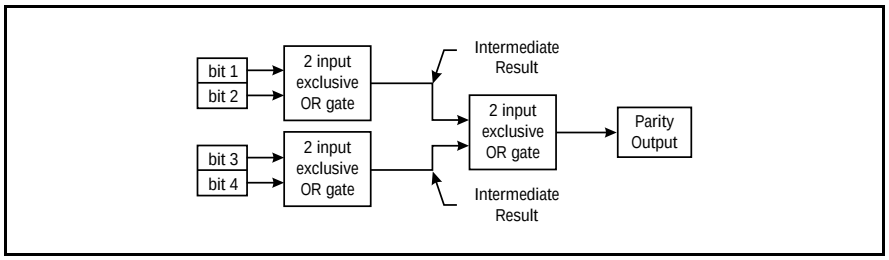

Figure 3. Four-bit parity tree example

Parity trees in hardware are fast, simple, and can be cascaded easily to accommodate N-bit data lengths.

Parallel "Successive Approximations" in Software

In a manner similar to the parity tree described above, the TMS320C54x family of DSPs may be used to generate parity in software. Up to 32 bits of data can be handled relatively quickly and efficiently. The feature of the 'C54x which enables this technique is the ability of the 40-bit accumulators to be exclusive OR'd against itself while a shift is applied. The process splits the accumulator into two subsections, shifts to align the two subsections for the exclusive or operation, and creates an intermediate result which contains half the number of bits that were originally present. Recursively applied, the process can reduce to $\log_2 N$ exclusive OR operations, much like binary successive approximation techniques used in analog to digital converters.

The code fragment below shows how this works. First, the input word containing the data bits is loaded into the accumulator from memory. A copy of accumulator A, shifted by N/2 bits is created, exclusive OR'd with itself to produce an intermediate result which is stored back into accumulator A. It is important to note that the intermediate result (N/2 bits) is located somewhere in the middle of the accumulator. The repetition of the process halves again the number of "output" bits, etc. until only one bit is finally remaining. The remaining bit may then be tested or stored away to indicate the calculated parity value. If the number or input bits is not a power of two, then the data word input to the accumulator must be masked such that bits which should be excluded are not counted in the parity calculation. Masking these bits to zero eliminates them from the parity calculation.

```
An 8-Bit Example:                    (extendable to 32 bits)
          ld    Smem,0,A             ;prep a sample value in low order of A

                                     ;begin kernel: quasi successive approximation
parity:   xor   A,4,A                ;A4:7 XOR with A0:3:       interim result [4 bits] in A4:7
          xor   A,2,A                ;A6:7 XOR with A4:5interim result [2 bits] in A6:7
          xor   A,1,A                ;A7    XOR with A6,        parity remains in bit 7 of A:
                                     ;there's garbage everywhere else
          ld    A,8,A                ;moves parity status into msb for conditional branch test
                                     ;end kernel: 8 bits in 3 cycles and 3 words

          bc    par0,AGEQ            ;A greater or equal to zero ==>parity=0
par1:     b     par1                 ;A negative                 ==>even parity
par0:     b     par0                 ;A greater or equal to zero ==>odd parity
```

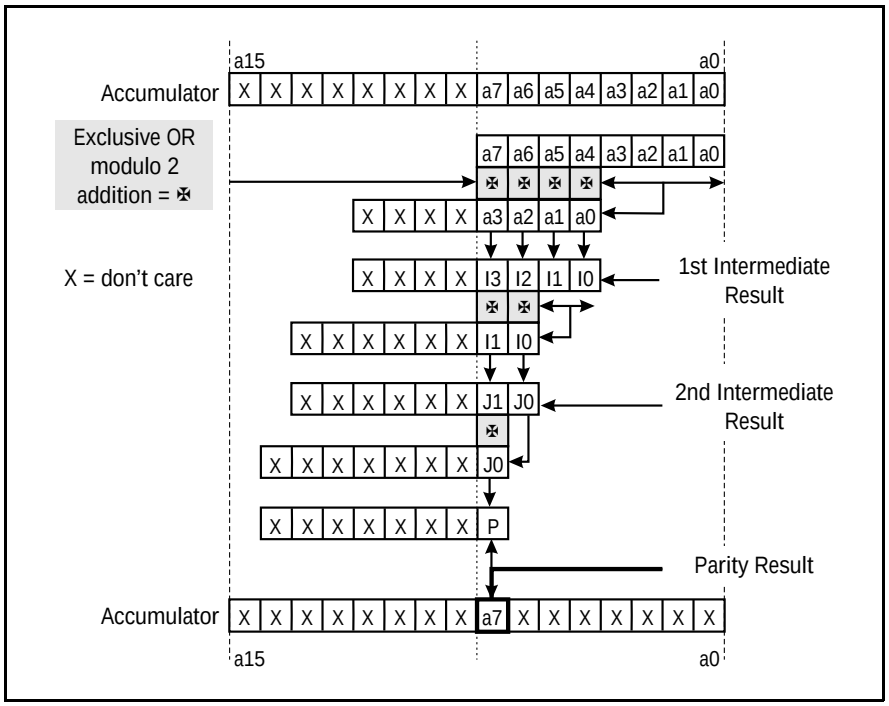

Figure 4.

Summary

The TMS320C54x instruction set accommodates both rapid and efficient generation of simple parity. The calculation of simple parity on 16 bits of data can be accomplished in four instruction words and four machine cycles. One or two cycles are required to get the data into the accumulator. One or two cycles are required to mask off the unused bit positions [length dependent]. The general solution for n bits is:

Number of cycles = $n_{prep} + N$
Number of words = $n_{prep} + N$
Where N is the smallest integer such that 2^N is greater than or equal to n.