ON Semiconductor

Is Now

To learn more about onsemi™, please visit our website at
www.onsemi.com

AND9860/D

Implementation of Error Code Correction in EEPROMs

ON Semiconductor®

www.onsemi.com

APPLICATION NOTE

INTRODUCTION

Some of ON's automotive EEPROMs, like the Grade 0 NV25xxx family (SPI, 1 – 64 Kb) and the Grade 1 CAV24Cxx / CAV25xxx (Grade 1, 128 Kb and higher) implement an Error Code Correction scheme. What this means is that for each chunk of data in the EEPROM array (8 bits for 1 – 64 Kb densities, 32 bits for 128 Kb and higher), the memory stores a redundancy code in separate EEPROM cells. When a value is written to a memory address, the memory automatically calculates a redundancy code as a function of that value and stores it in the redundancy bits at the same time when the user value is stored. When the user reads a value back from the EEPROM array, the memory automatically checks the value in the array against the redundancy code. If one of the EEPROM cells being read has failed and contains a different bit from what was written, the calculated redundancy code will differ from the stored redundancy code. Error Code Correction allows the memory to deduce which bit (either data or redundancy) has flipped, and correct the data before sending it out to the user.

This function only covers one corrupt bit per chunk. EEPROM cell failures tend to occur randomly in the memory array, which makes the probability of two successive bit failures in the same data chunk extremely low (excluding the scenario where the user overwrites a particular area of the array repeatedly, exceeding the maximum number of programming cycles specified in the datasheet). However, if more than one bit in the same chunk is corrupted, the data is irrecoverable.

The process is transparent to the user – a memory with ECC is used exactly like a memory without.

This application note will explain the technique by which the corrupted bit is identified. It is called Hamming Coding, and was developed for telecommunications.

Preliminaries: The Parity Bit

A parity bit is a simple error detection mechanism. Given a sequence of bits, the parity bit will be 0 if the number of 1s is even, and 1 if the number of 1s is odd. For instance, *parity*(0101) = 0, since there are two 1s in the sequence; *parity*(1011) = 1, since there are three 1s in the sequence.

The parity bit is stored (or transmitted, depending on the application) together with the data bits. If any one bit in the sequence is corrupted and its value flips, the calculated parity of the data bits will no longer match the value of the stored parity bit. However, the parity bit does not provide any information as to which bit is in error, making it impossible to recover the original data.

Hamming Codes

Hamming codes use multiple parity bits to allow the recipient of the data to determine which bit, if any, is in error. If we are sending four data bits, (d1, d2, d3, d4), the correction code will consist of three parity bits:

p1 = *parity*(d1, d2, d3)

p2 = *parity*(d2, d3, d4)

p3 = *parity*(d3, d4, d1)

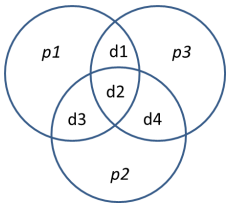

Figure 1. Diagram Showing How Each Data Bit Is Covered By a Combination of Parity Bits

When a chunk of memory is being read, the memory device calculates the expected values of the parity bits, and compares them to the parity bits stored as a redundancy code. Each data bit is covered by a unique combination of parity bits (see the table below). If a data bit is in error – i.e. if its value has flipped – the combination of parity bit mismatches will identify the error bit:

Data bit in error	Parity bit mismatch
d1	p1, p3
d2	p1, p2,p3
d3	p1, p2
d4	p2, p3

If any single parity bit does not match, it was that bit itself that was corrupted. If all parity bits match the parity of their data bits, no data has been corrupted.

This method can be extended to any number of data bits. The necessary condition is that each data bit should be covered by a unique combination of at least 2 parity bits. (If a bit were covered by a single parity bit, there would be no way to distinguish whether the data bit had flipped, or the parity bit.) Here is an example for 8 data bits, which require 4 parity bits:

	d1	d2	d3	d4	d5	d6	d7	d8
p1	x	X		x	x		x	
p2	x		x	x		x	x	
p3		X	x	x				x
p4					x	x	x	x

A four−byte chunk (32 bits) requires 6 parity bits.

PUBLICATION ORDERING INFORMATION

LITERATURE FULFILLMENT:
Literature Distribution Center for ON Semiconductor
19521 E. 32nd Pkwy, Aurora, Colorado 80011 USA
Phone: 303−675−2175 or 800−344−3860 Toll Free USA/Canada
Fax: 303−675−2176 or 800−344−3867 Toll Free USA/Canada
Email: orderlit@onsemi.com

N. American Technical Support: 800−282−9855 Toll Free USA/Canada
Europe, Middle East and Africa Technical Support:
Phone: 421 33 790 2910

ON Semiconductor Website: **www.onsemi.com**

Order Literature: http://www.onsemi.com/orderlit

For additional information, please contact your local Sales Representative