One question that I often am asked is "How many parity bytes are consumed" for various ECC implementations. To parallel the previous post, this entry will address parity sizes for Hamming, Reed/Solomon, and BCH implementations.

**Hamming**

Block Hamming codes correct only a single error and require $2 \times \log_2(n)$ of parity for a data block with $n$ data bits. Block Hamming codes are capable of correcting a single-bit error and detecting most double-bit errors.

Example

A 512B data block consists of 4096 ($2^{12}$) bits, thus a Hamming code requires 24 parity bits.

**Reed-Solomon**

Parity required for an RS code depends on the symbol size, Galois field size (GF), and ECC level provided by the code. There are trade-offs in selecting the most efficient symbol size for a given application, but generally the idea is to minimize the Galois Field size for a given blocksize by selecting an appropriate symbol size. The general formula is $2 \times GF \times ECC$.

Example

An RS over a 512B data block consisting of 9-bit symbols capable of supporting 8 symbol correction would require $2 \times 9 \times 8$, or 144 bits of parity.

**BCH**

Parity required for BCH is dependent on the Galois field size (GF) (determined by the data block size) and the ECC correction level. The number of parity bits can be computed as $GF \times ECC$.

Example

The following table shows the number of ECC bits/bytes required per correction block.

<table>
<thead>
<tr>
<th>Blocksize</th>
<th>ECC Level</th>
<th>ECC Bits</th>
<th>ECC Bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>512B</td>
<td>ECC 8</td>
<td>13*8=104</td>
<td>13</td>
</tr>
<tr>
<td>512B</td>
<td>ECC 16</td>
<td>13*16=208</td>
<td>26</td>
</tr>
<tr>
<td>1024B</td>
<td>ECC 24</td>
<td>14*24=336</td>
<td>42</td>
</tr>
<tr>
<td>1024B</td>
<td>ECC 40</td>
<td>14*40=560</td>
<td>70</td>
</tr>
</tbody>
</table>

While the GF field for BCH is larger than that for Reed-Solomon, the factor of 2 in the RS equation makes BCH more efficient for NAND applications since errors tend to not occur in groups (RS is better at these applications).
The move to larger block sizes also makes the BCH code more efficient since higher correction levels more than compensate for the additional bytes protected (see [ECC Trends whitepaper](#)).