Overview

- High-speed BCH block code encoder and decoder for FPGAs
- Fully compliant with the DVB-S2 standard ETSI EN 302 307
- Parallel I/Os and processing for high-speed operation:
  - 1 Gbits/s encoding [Virtex-5]
  - 650 Mbits/s encoding [Spartan-3]
  - up to 950 Mbits/s decoding [Virtex-5]
  - up to 400 Mbits/s decoding [Spartan-3]
- Corrects $t = 8, 10$ or 12 errors per block.
- Decoder flags frames with uncorrectable errors.
- Decoder reports number of bit errors corrected at the end of each decoded block.

Encoder

I/Os

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK</td>
<td>DATA_OUT[7:0]</td>
</tr>
<tr>
<td>SYNC_RESET</td>
<td>SAMPLE_CLK_OUT</td>
</tr>
<tr>
<td>DATA_IN[7:0]</td>
<td>SAMPLE_CLK_IN</td>
</tr>
<tr>
<td>SAMPLE_CLK_IN_REQ</td>
<td>SOF_IN</td>
</tr>
<tr>
<td>SOF_OUT</td>
<td>SAMPLE_CLK_OUT_REQ</td>
</tr>
</tbody>
</table>

Flow control is ensured through the 
SAMPLE_CLK_x_REQ signals which convey "Clear To Send" information from the stream recipient. The data source must immediately stop sending data when the data sink clears this signal.

All inputs and outputs are synchronous with the rising edge of the synchronous clock CLK.

Speed

<table>
<thead>
<tr>
<th>FPGA</th>
<th>Clock (max)</th>
<th>Encoder output data rate (max)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spartan-3</td>
<td>83 MHz</td>
<td>650 Mbits/s</td>
</tr>
<tr>
<td>Virtex-5</td>
<td>131 MHz</td>
<td>1 Gbits/s</td>
</tr>
</tbody>
</table>

A minimum guard time of at least $(N_{bch} - K_{bch})/8 + 2$ clocks must be inserted between successive input frames to let the encoder send the parity bits to its output. More generally, the data source should check the flow control signal SAMPLE_CLK_IN_REQ before sending any input data to the encoder.

Device Utilization Summary

Device: Xilinx Spartan-3

| Number of slices | 672 |
| Flip Flops       | 258 |
| 4 input LUTs     | 1268|
| RAMB16           | 0   |
| 18x18 multiplier | 0   |
| GCLKs            | 1   |

Device: Xilinx Virtex-5

| Number of slices | 482 |
| Flip Flops       | 265 |
| LUTs             | 964 |
| RAMB             | 0   |
| DSP              | 0   |
| GCLKs            | 1   |
**Decoder**

### I/Os

<table>
<thead>
<tr>
<th>Inputs</th>
<th>Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>CLK</td>
<td>DATA_OUT[7:0]</td>
</tr>
<tr>
<td>SYNC_RESET</td>
<td>SAMPLE_CLK_OUT</td>
</tr>
<tr>
<td>DATA_IN[7:0]</td>
<td>SOF_OUT</td>
</tr>
<tr>
<td>SAMPLE_CLK_IN</td>
<td>EOF_OUT</td>
</tr>
<tr>
<td>SOF_IN</td>
<td>SAMPLE_CLK_OUT_REQ</td>
</tr>
<tr>
<td>SAMPLE_CLK_IN_REQ</td>
<td>Control[1:0]</td>
</tr>
<tr>
<td>CONTROL[1:0]</td>
<td>N_CORRECTED[3:0]</td>
</tr>
<tr>
<td>NBCH[15:0]</td>
<td>GOOD_FRAME</td>
</tr>
<tr>
<td>KBCH[15:0]</td>
<td></td>
</tr>
</tbody>
</table>

### Controls

- **CLK**: Clock input
- **SYNC_RESET**: Synchronization reset input
- **DATA_IN[7:0]**: Decoder input data
- **SAMPLE_CLK_IN** and **SAMPLE_CLK_IN_REQ**: Sample clock input and request
- **CONTROL[1:0]**: Control signal
- **NBCH[15:0]**: Number of bad channel inputs
- **KBCH[15:0]**: Number of known bad channel inputs

### Monitoring

- **N_CORRECTED[3:0]**: Number of corrected errors
- **GOOD_FRAME**: Frame is good flag
- **EOF_OUT**: End of frame output

**Speed**

<table>
<thead>
<tr>
<th>FPGA</th>
<th>Clock (max)</th>
<th>Decoder input data rate (max)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Spartan-3</td>
<td>73 MHz</td>
<td>356 Mbits/s (51840,51648,12)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>423 Mbits/s (58320,58192,8)</td>
</tr>
<tr>
<td>Virtex-5</td>
<td>166 MHz</td>
<td>810 Mbits/s (51840,51648,12)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>963 Mbits/s (58320,58192,8)</td>
</tr>
</tbody>
</table>

The decoder architecture is such that the three decoding stages are pipelined and the input data is stored in a 128 Kbits elastic buffer until the error locations are found. Therefore, it is possible to input a new frame even before the previous one is completely decoded.

The processing time budget for each decoding stage can be expressed as follows:

1. Syndrome computation: 16.5 * t clocks.
2. Error location polynomial: 328 * t clocks in the worst case, when t errors are present in the received frame.
3. Factoring the error location polynomial before the first output byte: \((2^{16} - N_{bch})/8 + 4\) clocks for GF(2\(^{16}\))
4. Output: \(K_{bch}/8\) clocks

Using the above information, one can compute the maximum decoding speed for each DVB-S2 BCH code variant. For example, the best decoding speed for the (51840,51648,12) code is \(51840/8 + (16.5 + 328)*12 = 10614\) clocks per frame.

### Device Utilization Summary

#### Device: Xilinx Spartan-3

<table>
<thead>
<tr>
<th>Resource</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of slices</td>
<td>4906</td>
</tr>
<tr>
<td>Flip Flops</td>
<td>5002</td>
</tr>
<tr>
<td>4 input LUTs</td>
<td>7910</td>
</tr>
<tr>
<td>RAMB16</td>
<td>9</td>
</tr>
<tr>
<td>18x18 multiplier</td>
<td>0</td>
</tr>
<tr>
<td>GCLKs</td>
<td>1</td>
</tr>
<tr>
<td>Total gate count</td>
<td>679,436</td>
</tr>
</tbody>
</table>

#### Device: Xilinx Virtex-5

<table>
<thead>
<tr>
<th>Resource</th>
<th>Quantity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of slices</td>
<td>2626</td>
</tr>
<tr>
<td>Flip Flops</td>
<td>4995</td>
</tr>
<tr>
<td>LUTs</td>
<td>6919</td>
</tr>
<tr>
<td>RAMB18X2s</td>
<td>5</td>
</tr>
<tr>
<td>DSP</td>
<td>0</td>
</tr>
<tr>
<td>GCLKs</td>
<td>1</td>
</tr>
</tbody>
</table>

**DVB-S2 BCH**

The DVB-S2 standard lists 21 variants of long BCH codes. Each variant is identified by its code block size \(N_{bch}\), uncoded block size \(K_{bch}\), error correction capability \(t\) and frame type.

The VHDL code implements all 21 variants listed in tables 5a and 5b of the specifications [1].

**Examples (see [1] for complete list)**

<table>
<thead>
<tr>
<th>(K_{bch})</th>
<th>(N_{bch})</th>
<th>(t)</th>
<th>Frame</th>
</tr>
</thead>
<tbody>
<tr>
<td>16008</td>
<td>16200</td>
<td>12</td>
<td>normal</td>
</tr>
<tr>
<td>51648</td>
<td>51840</td>
<td>12</td>
<td>normal</td>
</tr>
<tr>
<td>53840</td>
<td>54000</td>
<td>10</td>
<td>normal</td>
</tr>
<tr>
<td>58192</td>
<td>58320</td>
<td>8</td>
<td>Normal</td>
</tr>
<tr>
<td>3072</td>
<td>3240</td>
<td>12</td>
<td>short</td>
</tr>
</tbody>
</table>

The codes for normal frames are computed in GF(2\(^{16}\)), whereas the short frame codes are computed over GF(2\(^{14}\)).

The primitive polynomials used to generate the Galois fields are:

- \(x^{16} + x^5 + x^3 + x^2 + 1\) for GF(2\(^{16}\))
- \(x^{14} + x^3 + x + 1\) for GF(2\(^{14}\))

Matlab:

- Primpoly(16,’min’)
- Primpoly(14,’min’)
The specification document lists the 12 minimum polynomials $g_i(x)$ for GF($2^{16}$) and GF($2^{14}$) in tables 6a and 6b respectively.

By multiplying the first 8, 10 or 12 minimum polynomials, we can construct the generator polynomials for four configurations: GF($2^{16}$) $t=8,10,12$ and GF($2^{14}$) $t=12$. The resulting generator polynomials can be represented by their binary coefficients as listed below:

constant GENPOLY0:
std_logic_vector(128 downto 0) := "1" & x"1c07255f712797bd19fc6d7504f9662B";

constant GENPOLY1:
std_logic_vector(160 downto 0) := "1" & "60150CEDFC2A331F6A785703EFD12301B8BB6591";

constant GENPOLY2:
std_logic_vector(192 downto 0) := "1" & x"4E260E83845C511C50CF2CD8DC35089034785F7660255E7";

constant GENPOLY3:
std_logic_vector(168 downto 0) := "1" & x"4062DBEA9869B262CD23A39069528FE7D7D11905A5";

The encoder uses these generator polynomials to generate the ($N_{\text{bch}}$ - $K_{\text{bch}}$) parity bits appended to the $K_{\text{bch}}$ input data bits. As described in section 5.3.1 of the DVB-S2 specifications [1], the parity bits are the remainder of a polynomial division of the shifted input bits by the generator polynomial.

### DVB-S2 BCH Decoding

Decoding a BCH block is done in three steps.

1. compute the syndromes
2. derive the error location polynomial
3. find the roots of the error location polynomial and correct the bit errors.

### Syndromes

To compute a syndrome $S_i$, one must first divide the input block by the twelve polynomials $g_i(x)$, where $g_i(x)$ represent the minimum polynomials of $\alpha^i$ for $i = 1$ to $2t$ (see table below). The twelve minimum polynomials $g_i(x)$ are listed in the DVB-S2 specifications in Tables 6a and 6b.

When a received frame is error-free, all syndromes are zero.

Verifying the syndrome computation is easy. Assuming two bit errors at locations 16191 and 16184 (with bit locations being numbered from $N_{\text{bch}}-1$ (first bit received) to 0), then

$S_1 = \alpha_{16191} + \alpha_{16184}$

$S_2 = (\alpha_{16191})^2 + (\alpha_{16184})^2$

$S_3 = (\alpha_{16191})^3 + (\alpha_{16184})^3$

$\ldots$

$S_{2t} = (\alpha_{16191})^{2t} + (\alpha_{16184})^{2t}$

Matlab:

```matlab
prim_poly16 = primpoly(16,'min');
alpha = gf(2,16,prim_poly16);
s1 = alpha^16191 + alpha^16184;
s2 = (alpha^16191)^2 + (alpha^16184)^2;
s3 = (alpha^16191)^3 + (alpha^16184)^3;
\ldots
s24 = (alpha^16191)^24 + (alpha^16184)^24
```
**Error Location Polynomial**

The Berlekamp-Massey algorithm is implemented to find the error location polynomial. 
\[ \sigma(x) = (1+\alpha^{L_1}x)(1+\alpha^{L_2}x)(1+\alpha^{L_3}x)(1+\alpha^{L_4}x) \ldots \]
where Li are the error locations.

At the end of this step, the error location polynomial is expressed as
\[ \sigma(x) = \sigma_0 + \sigma_1x + \sigma_2x^2 + \sigma_3x^3 + \ldots \]

Comparing the VHDL simulation with Matlab is easy. Let us assume two bit errors at locations 16191 and 16184 (with bit locations being numbered from \(N_{bch}-1\) (first bit received) to 0), then the error location polynomial is computed by expanding \((1+\alpha^{16191}x)(1+\alpha^{16184}x)\)

Matlab:
```
alpha = gf(2,16,prim_poly16);
p1 = [alpha^16191 1];
p2 = [alpha^16184 1];
elp = conv(p1,p2);
```

**Factoring and Error Correction**

Chien’s search circuit [3] is used to factor the error location polynomial \(\sigma(x)\). While the data bit at location Li is streamed to the output, the algorithm assesses whether \(\alpha^{-L_i}\) is a root of \(\sigma(x)\). If so, it is erroneous and is corrected.

128Kbits of block RAM is used as elastic buffer to temporarily store the received bits while error decoding takes place.

**Flow Control**

The decoder input first goes through an input elastic buffer to regulate the flow.

The buffer output data flow is sent to two components: the syndrome computation `bch_syndromes.vhd` and the error correction `bch_ec.vhd`. Thus, both components are able to control the data flow from the input elastic buffer using their flow control signals `SAMPLE0A_CLK_REQ` and `SAMPLE0B_CLK_REQ` respectively.

Syndromes computation is performed on the fly. Upon reading the last frame byte from the input elastic buffer, `bch_syndromes.vhd` exercises its `SAMPLE0A_CLK_REQ` flow control signal to immediately stop the input flow before a new start of frame. The end of syndromes computation is marked by the availability of the syndromes (SYNDROME1 through 24) and a pulse `SYNDROME_SAMPLE_CLK`. At this point `bch_syndromes.vhd` is ready for the next input frame.

The syndromes are passed to `bcherrorlocator.vhd` to compute the error location polynomials. The computation is triggered by the `SYNDROME_SAMPLE_CLK` pulse and ends at the `ELP_CLK` pulse. The resulting error location polynomials are available in `ELP1` through 12. In the special case of an error-free frame, there is no need to compute the error location polynomials. The `ALL_ZERO_SYNDROMES` net goes high when this happens.

The final decoding step, error correction, is implemented within the `bch_ec.vhd` component. This component includes a 128 Kbit elastic buffer large enough receive a new frame while processing the previous one. The purpose of the `SAMPLE0B_CLK_REQ` flow control flag is stop the input data flow unless at least 1/32th of the internal elastic buffer is available.
The flow control is primarily located within `bch_ec.vhd`. It is a little bit complex. There are five key events in the life of a BCH frame decoding, in the order of occurrence:

- input start of frame pulse SOF_IN
- received all input data (excluding parity bits) INPUT_DATA_COMPLETE
- syndrome ready pulse SYNDROME_SAMPLE_CLK
- error location ready pulse ELP_CLK_IN
- all decoded bytes sent out OUTPUT_DATA_COMPLETE

Note: the error location computation is skipped if ALL_ZERO_SYNDROMES is high.
Typical bch_ec.vhd capture. Includes frames with correctable and uncorrectable errors.

Reference documents

[1] ETSI EN 302 307, Section 5.3 FEC encoding


Configuration Management

The current software revision is 1.

VHDL development environment

The VHDL software was developed using the Xilinx ISE 8.2, 9.1 and 10.1 development environments. The synthesis tool is Xilinx XST.

Target FPGA

The VHDL code is ready-to-use with Xilinx Spartan-4, Virtex-4 and Virtex-5 FGPAs. Other FPGAs may need very minor adjustments.

Xilinx-specific code

The VHDL source code was written in generic VHDL with few Xilinx primitives. No Xilinx CORE is used. The Xilinx primitives are:

- BUFG
- RAMB16_S9_S9
**VHDL software hierarchy**

The code is stored with one, and only one, entity per file.

**Encoder**

```
BCHenco behavioral (C:\VHDL\com1410_01
   Inst_BCHenco_A - BCHenco_A - behavi
   BCHenco SEG8A_001 - BCHenco_
   BCHenco SEG8A_002 - BCHenco_
   BCHenco SEG8A_003 - BCHenco_
   BCHenco SEG8A_004 - BCHenco_
   BCHenco SEG8A_005 - BCHenco_
   BCHenco SEG8A_006 - BCHenco_
   BCHenco SEG8A_007 - BCHenco_
   BCHenco SEG8A_008 - BCHenco_
   BCHenco SEG8A_009 - BCHenco_
   BCHenco SEG8A_010 - BCHenco_
   BCHenco SEG8A_011 - BCHenco_
   BCHenco SEG8A_012 - BCHenco_
   BCHenco SEG8A_013 - BCHenco_
   BCHenco SEG8A_014 - BCHenco_
   BCHenco SEG8A_015 - BCHenco_
   BCHenco SEG8A_016 - BCHenco_
   BCHenco SEG8A_017 - BCHenco_
   BCHenco SEG8A_018 - BCHenco_
   BCHenco SEG8A_019 - BCHenco_
   BCHenco SEG8A_020 - BCHenco_
   BCHenco SEG8A_021 - BCHenco_
   BCHenco SEG8A_022 - BCHenco_
   BCHenco SEG8A_023 - BCHenco_
   BCHenco SEG8A 024 - BCHenco
```

**Decoder**

The decoder hierarchical structure reflects the three successive decoding steps:

1. `bch_syndromes.vhd`: compute the syndromes
2. `bcherrorlocator.vhd`: derive the error location polynomial
3. `bch_ec.vhd`: find the roots of the error location polynomial and correct the bit errors.

```
BCH_DEC behavioral (C:\VHDL\com1410_000\src\b
   Inst_BCH_SYNDROMES - BCH_SYNDROMES - beha
   Inst_BCHERRORLOCATOR - BCHERRORLOCATOR
   BCH_EC_001 - BCH_EC - behavioral (C:\VHDL\com1
```

**Clock / Timing**

The software uses a single master clock (CLK) which serves as input clock, output clock and signal processing clock.

**Test Benches**

Several test benches are included for end-to-end and component-level VHDL simulation:

- `tbbchencdec2.vhd`: end-to-end simulation including encoder, decoder and added bit errors.

**ComBlock Ordering Information**

COM-1209ASOFT High-speed DVB-S2 BCH encoder & decoder. VHDL source code.

**Contact Information**

MSS • 18221-A Flower Hill Way • Gaithersburg, Maryland 20879 • U.S.A.
Telephone: (240) 631-1111
Facsimile: (240) 631-1676
E-mail: info@comblock.com