

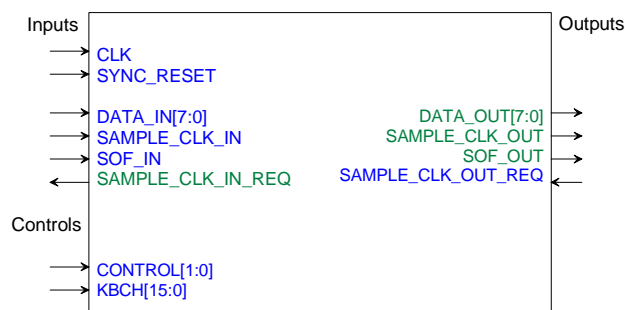
## COM-1209ASOFT HIGH-SPEED DVB-S2 BCH CODE DECODER & ENCODER VHDL SOURCE CODE OVERVIEW

### 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



8-bit parallel data input and output help maximize the throughput. The first byte in the stream is marked by a Start Of Frame (SOF) flag.

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

FPGA	Clock (max)	Encoder output data rate (max)
Spartan-3	83 MHz	650 Mbits/s
Virtex-5	131 MHz	1 Gbits/s

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

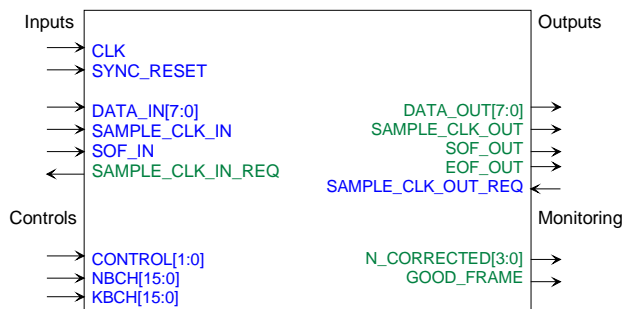
Number of slices	672
Flip Flops	258
4 input LUTs	1268
RAMB16	0
18x18 multiplier	0
GCLKs	1

Device: Xilinx Virtex-5

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

## Decoder

### I/Os



### Speed

FPGA	Clock (max)	Decoder input data rate (max)
Spartan-3	73 MHz	356 Mbits/s (51840,51648,12)  423 Mbits/s (58320,58192,8)
Virtex-5	166 MHz	810 Mbits/s (51840,51648,12)  963 Mbits/s (58320,58192,8)

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

Number of slices	4906
Flip Flops	5002
4 input LUTs	7910
RAMB16	9
18x18 multiplier	0
GCLKs	1
Total equivalent gate count	679,436

Device: Xilinx Virtex-5

Number of slices	2626
Flip Flops	4995
LUTs	6919
RAMB18X2s	5
DSP	0
GCLKs	1

## DVB-S2 BCH

The DVB-S2 standard lists includes 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)

$K_{bch}$	$N_{bch}$	$t$	Frame
16008	16200	12	normal
51648	51840	12	normal
53840	54000	10	normal
58192	58320	8	Normal
3072	3240	12	short

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  $GF(2^{16})$  and  $x^{14} + x^5 + x^3 + x + 1$  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"4E260E83845C511C50CF2CD8DC350889034785F7
660255E7";

constant GENPOLY3:
std_logic_vector(168 downto 0) := "1" &
x"4062DBEA9869B262CD23A39069528FE7D7D11905
A5";
```

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  $g_j(x)$ , where  $g_j(x)$  represent the minimum polynomials of  $\alpha^i$  for  $i = 1$  to  $2t$  (see table below). The twelve minimum polynomials  $g_j(x)$  are listed in the DVB-S2 specifications in Tables 6a and 6b.

The remainder  $b_j(x)$  is then evaluated for  $\alpha^i$  as  $S_i = b_j(\alpha^i)$ .

The roots of  $g_j(x)$  are as follows:

Minimum polynomial $g_j(x)$	Roots $\alpha^i$
$g_1(x)$	$\alpha, \alpha^2, \alpha^4, \alpha^8, \alpha^{16}$
$g_2(x)$	$\alpha^3, \alpha^6, \alpha^{12}, \alpha^{24}$
$g_3(x)$	$\alpha^5, \alpha^{10}, \alpha^{20}$
$g_4(x)$	$\alpha^7, \alpha^{14}$
$g_5(x)$	$\alpha^9, \alpha^{18}$
$g_6(x)$	$\alpha^{11}, \alpha^{22}$
$g_7(x)$	$\alpha^{13}$
$g_8(x)$	$\alpha^{15}$
$g_9(x)$	$\alpha^{17}$
$g_{10}(x)$	$\alpha^{19}$
$g_{11}(x)$	$\alpha^{21}$
$g_{12}(x)$	$\alpha^{23}$

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

$$S_1 = \alpha^{16191} + \alpha^{16184}$$

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

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

$$\dots$$

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

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
...
s24 = (alpha^16191)^24 + (alpha^16184)^24
```

## Error Location Polynomial

The Berlekamp-Massey algorithm is implemented to find the error location polynomial.

$$\sigma(x) = (1 + \alpha^{L_1}x) (1 + \alpha^{L_2}x) (1 + \alpha^{L_3}x) (1 + \alpha^{L_4}x) \dots$$

where  $L_i$  are the error locations.

At the end of this step, the error location polynomial is expressed as

$$\sigma(x) = \sigma_0 + \sigma_1x + \sigma_2x^2 + \sigma_3x^3 + \dots$$

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  $N_{\text{bch}}-1$  (first bit received) to 0), then the error location polynomial is computed by expanding  $(1 + \alpha^{16191}x) (1 + \alpha^{16184}x)$

Matlab:

```
prim_poly16 = primpoly(16,'min');
```

```
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  $L_i$  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

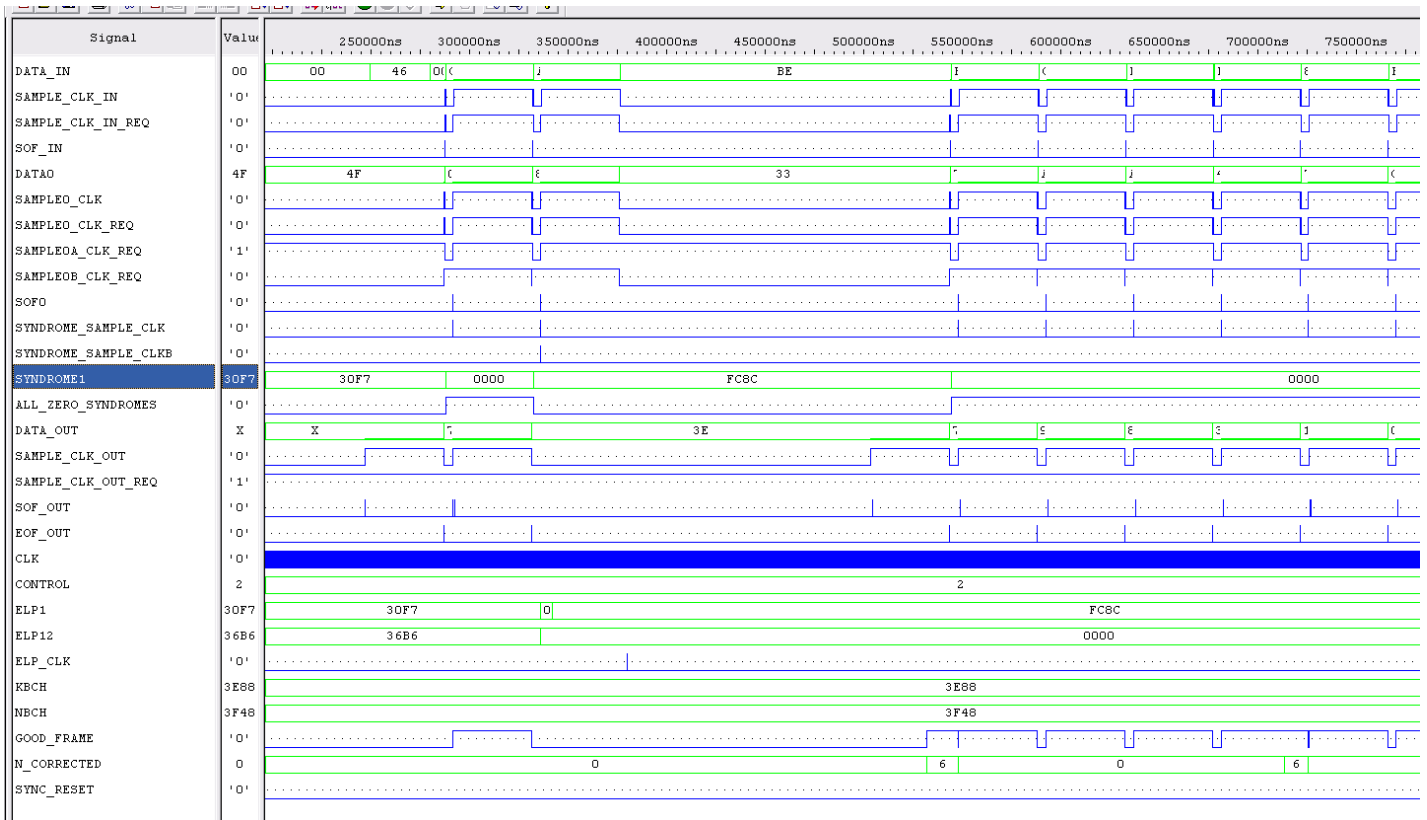
The decoder input first goes through an input elastic buffer to regulate the flow.

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.

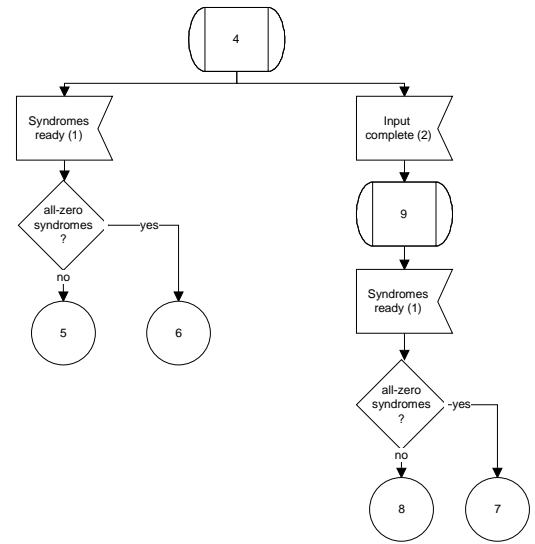
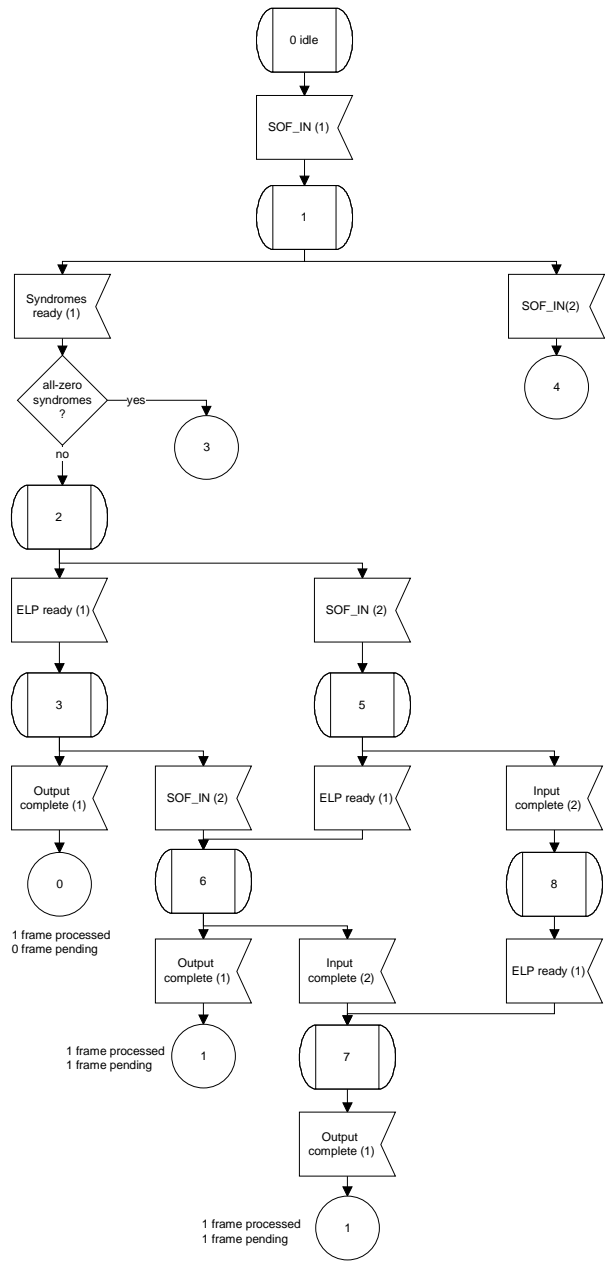


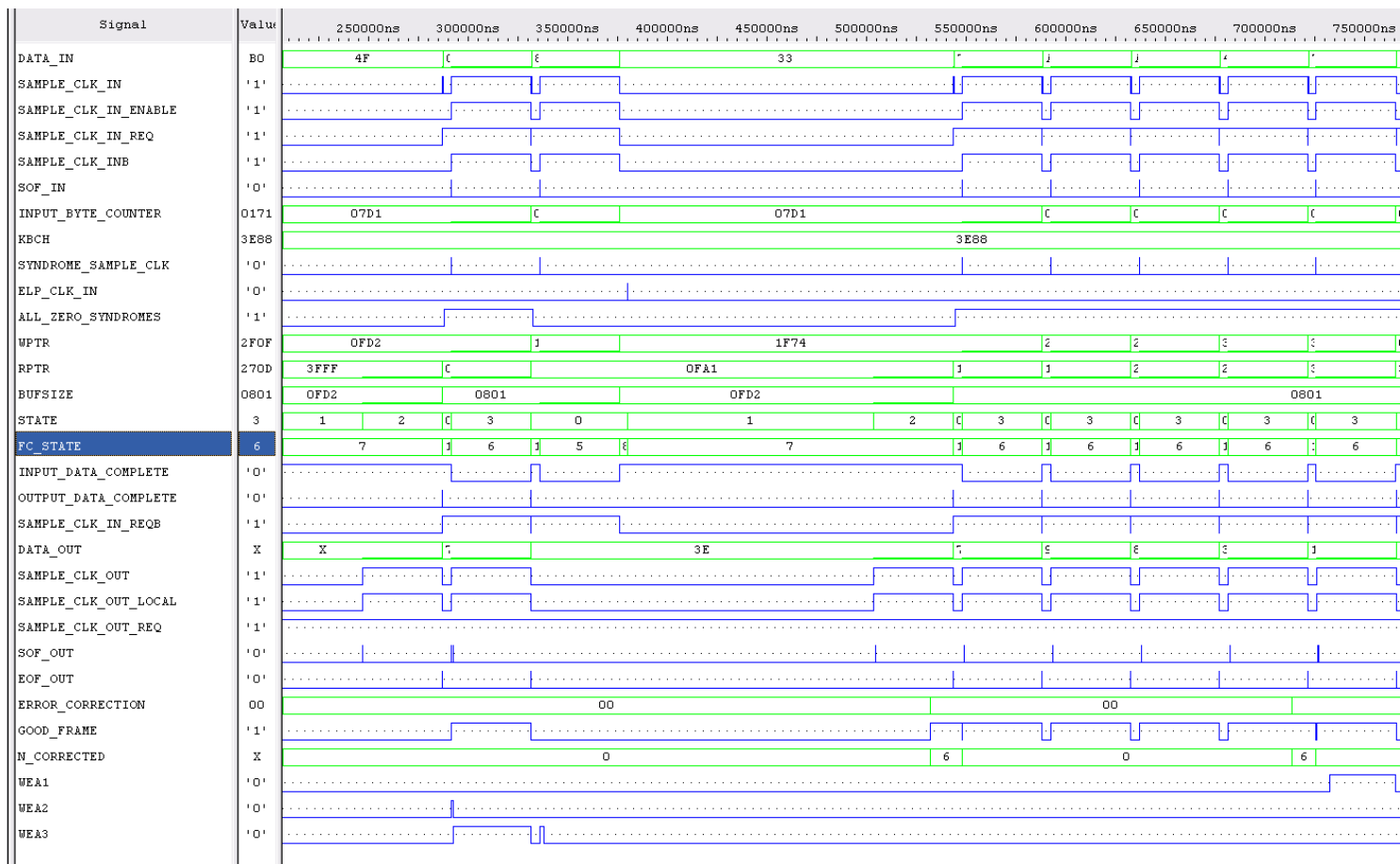
Typical `bch_dec.vhd` capture. Includes input frames with correctable and uncorrectable errors.

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
- [2] “Shift-Register Synthesis and BCH Decoding”, James L. Massey, IEEE Transactions on Information Theory, January 1969.
- [3] “Error Control Coding, Fundamentals and Applications”, Shu Lin / Daniel Costello.

## 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 FPGAs. Other FPGAs may need very minor adjustments.

## Xilinx-specific code

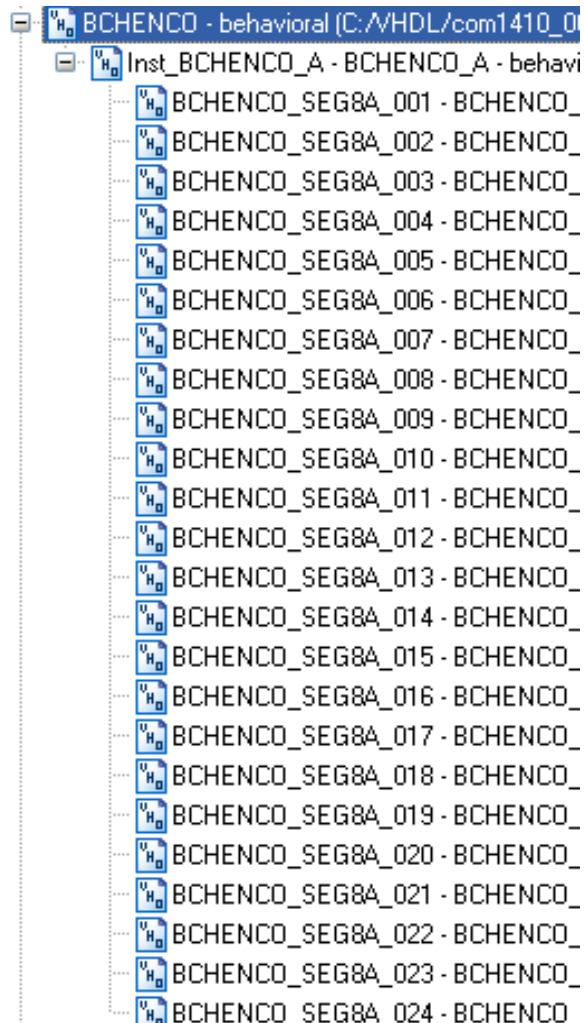
The VHDL source code was written in generic VHDL with few Xilinx primitives. No Xilinx CORE is used. The Xilinx primitives are:

- BUFG
- RAMB16\_S9\_S9

## VHDL software hierarchy

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

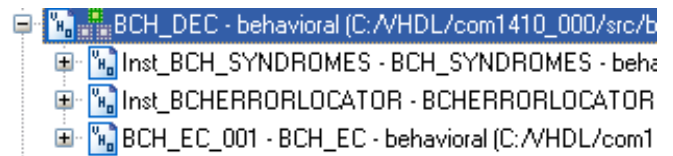
### Encoder



### 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.



### 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