YAHAMM
Yet Another Hamming Encoder and Decoder
—
Design

Nicola De Simone (ndesimone@opencores.org)
Version 0.1

Contents

The present document assumes that the reader is familiar with the Specification Document and with the references *Hamming code* [1] and *Hamming(7,4)* [2] and with the terminology used therein. No additional knowledge is needed, apart from basic matrix algebra. Information already included in those documents are not be duplicated.

1 General structure

The user connects the encoder entity in `yahamm_enc` to the decoder entity `yahamm_dec`, as described by the specification document. No other logic is needed. The amount of code in those two entities is kept to the minimal by choice. In the encoder the core functionality is in the synchronous logic:

```
code_sys <= to_slv(xor_multiply_vec(G, data_i_padded));
```

corresponding to the math operation:

$$\textbf{code\_sys} = \textbf{G data\_i\_padded}$$

where `G` is the code generator matrix in systematic form, `data_i_padded` the input data and `code_sys` the code word in systematic form.

Similarly, in the decoder the core functionality is in the synchronous logic:

```
syndrome <= xor_multiply_vec(H, code_nonsys);
```

corresponding to the math operation:

$$\textbf{syndrome} = \textbf{H code\_nonsys}$$

where `H` is the parity-check matrix in non-systematic form, `code_non_sys` the input data in non-systematic form and `syndrome` the error syndrome vector that identifies the patterns of errors.

There is some more logic in the decoder entity `yahamm_dec` to implement the counters of errors corrected or detected and the position of the error.

The higher complexity is hidden inside the package `yahamm_pkg` where there are the functions to create the code generator matrix and the parity-check-matrix, to calculate the number of parity bits, block length, to swap between systematic and non-systematic form, to multiply two matrices or matrix and vector. `matrix_pkg` contains some help functions for `yahamm_pkg`.

One parity bit If the encoder and the decoder are configured with the generic `ONE_PARITY_BIT` set to true, their behavior change to a simple one parity bit encoder and decoder. This is still described and handled with same matrix formalism of the Hamming code, so all math functions accept a parameter `ONE_PARITY_BIT` and behave differently in this case.

Additional parity bit (SECDED) Hamming codes can be extended by an extra parity bit. This way, it is possible to increase the minimum distance of the Hamming code to 4, which allows the decoder to distinguish between single bit errors and two-bit errors. Thus the decoder can detect and correct a single error and at the same time detect (but not correct) a double error. If the decoder does not attempt to correct errors, it can detect up to three errors [1].

`EXTRA_PARITY_BIT` adds an extra row and extra column to the parity-check matrix and it's handled as a special case in the construction of the code generator matrix.

Bus padding Incoming data bus `data_i` are zero-padded on the most significant bit to length $2^r - r - 1$, where r is the number of parity bits `NPARITY_BITS` specified by generic. This allow the user to use a data bus of any width, ideally the smallest necessary in order to economize routing resources, while the logic for the Hamming code can operate with the standard block length for any given number of parity bits.

2 matrix_pkg

File matrix_pkg.vhd.

3 Encoder

See Fig. 1. Entity `yahamm_enc`, file yahamm_enc.vhd. Latency 1.

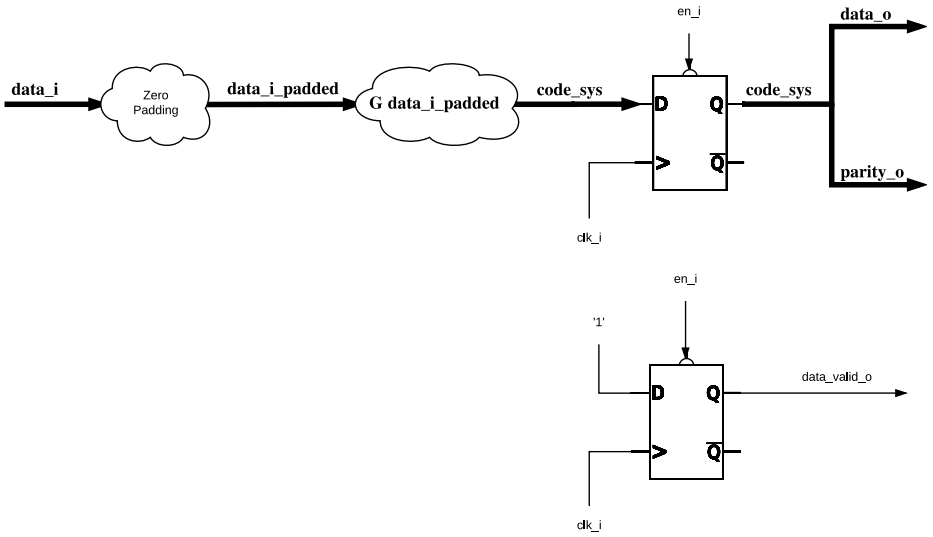

Figure 1: Encoder structure.

Input data `data_i` are zero-padded, then multiplied by the code generator matrix **G** in systematic form (see Sec. 5) to compute the code word in systematic form. Note that the only synthesized logic up to this point are the xor operations used to perform the product with the code generator matrix.

The code word in systematic form had the data in lower significant bits, routed on the output port `data_o`, and the parity bits in the highest significant bits, route on the output port `parity_o`. `data_valid_o` is the synchronous clear signal `en_i` delayed.

4 Decoder

See Fig. 2. Entity `yahamm_dec`, file yahamm_dec.vhd. Latency 2.

Input parity bits `parity_i` together with zero-padded input data `data_i` form the systematic code word. This is swapped to non-systematic form

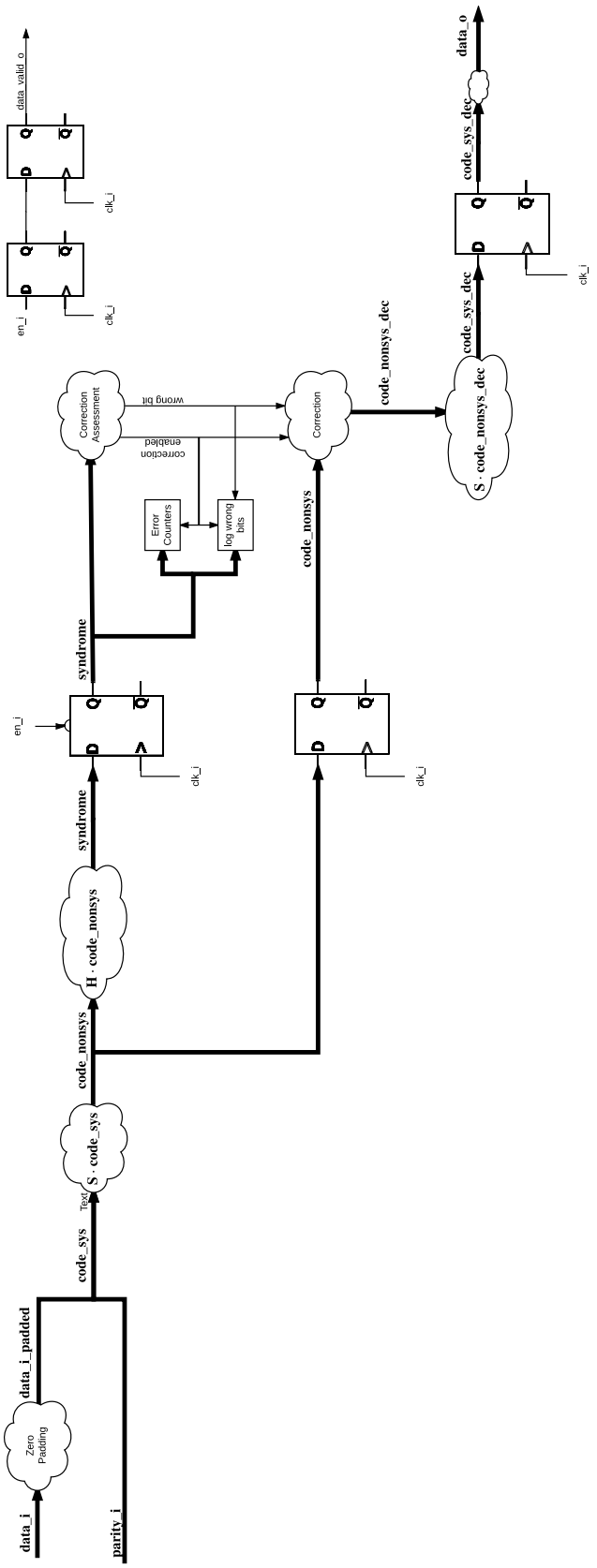

Figure 2: Decoder structure.

<table>
<tr><td></td><td></td><td colspan="4">CORRECT</td></tr>
<tr><td></td><td></td><td>false</td><td colspan="3">true</td></tr>
<tr><td rowspan="2">EXTRA_PARITY_BIT</td><td>0</td><td rowspan="2">Correction disabled</td><td colspan="3">(SEC)
<table><tr><td rowspan="2">syndrome</td><td>0</td><td>no error</td></tr><tr><td>≠ 0</td><td>wrong_bit <= syndrome</td></tr></table></td></tr>
<tr><td>1</td><td colspan="3">(SECDED)
<table><tr><td colspan="2"></td><td colspan="2">syndrome's MSB</td></tr><tr><td colspan="2"></td><td>'0'</td><td>'1'</td></tr><tr><td rowspan="2">syndrome's LSBs</td><td>0</td><td>no error</td><td>SEC
wrong_bit <= extra_bit</td></tr><tr><td>≠ 0</td><td>DED
no correction</td><td>SEC
wrong_bit <= syndrome LSBs</td></tr></table></td></tr>
</table>

Table 1:

multiplying for the swapping matrix **S** (see Sec. 5). The non-systematic code word is multiplied by the parity-check matrix **H** in non-systematic form (see Sec. 5) to compute the syndrome. Note that the only synthesized logic up to this point are the xor operations used to perform the product with the parity-check matrix.

A non zero syndrome means indicates an error. There are different possibilities, depending on the user choice of the value of the generics `CORRECT` and `EXTRA_PARITY_BIT`, as show by Table 1. A combinatorial logic (see *Correction Assessment* in Fig. 2) determines if the correction has to be done (`correction_en` signal|) and the position of the wrong bit (`wrong_bit` signal). In the SECDED configuration (`CORRECT` true and `EXTRA_PARITY_BIT` 1), a double error can be detected observing that the syndrome is odd with only the extra parity bit '1', and no correction is done.

In the Single Event Detected case (SEC), and with `CORRECT` generic set to `true`, the wrong bit is then corrected in the non-systematic form of the code word. This is then swapped to systematic form and `data_o` data output is the least signal part of this code word.

Independently from the configuration of the generic CORRECT, Single Error and Double Error are counted (`cnt_proc` process). Note that it is not possible to count Double Error unless the `EXTRA_PARITY_BIT` generic has value 1. Counters are connected to the ports `cnt_errors_corrected_o` and `cnt_errors_detected_o` and can be synchronously cleared with the input `cnt_clr`.

Independently from the configuration of the generic CORRECT, the position of the wrong bit in case of Single Error is used to flip the corresponding bit in the internal `log_wrong_bit_pos_data_o_nonsys` that, after form swapping and slicing, is mapped to the output ports `log_wrong_bit_pos_data_o` and `log_wrong_bit_pos_parity_o`.

5 yahamm_pkg

File yahamm_pkg.vhd.

function calc_nparity_bits

```
function calc_nparity_bits (
  k : natural;
  ONE_PARITY_BIT : boolean := false)
  return natural;
```

r parity bits can cover up to $2^r - 1$ bits, including data and parity bits. That is $2^r - r - 1$ data bits. The function returns the smallest r such that $2^r - r - 1 \geq k$, where k is the message length (number of data bits). It returns 1 if `ONE_PARITY_BIT` is true, by definition.

function calc_block_length

```
function calc_block_length (
  k : natural;
  ONE_PARITY_BIT : boolean := false)
  return natural;
```

Length of message bits plus parity bits. r parity bits can cover up to $2^r - 1$ bits, including data and parity bits. The function returns $2^r - 1$ where `r := calc_nparity_bits(k)`. It returns $k+1$ if `ONE_PARITY_BIT` is true.

function calc_block_length

```
function calc_block_length (
  k : natural;
  ONE_PARITY_BIT : boolean := false)
  return natural;
```

Length of message bits plus parity bits. r parity bits can cover up to $2^r - 1$ bits, including data and parity bits. The function returns $2^r - 1$ where `r := calc_nparity_bits(k)`. It returns $k+1$ if `ONE_PARITY_BIT` is true. Note that block length does not depend from `EXTRA_PARITY_BIT`; this is an arbitrary choice to simplify the code.

function check_parameters Sanity check on the generic settings for `yahamm_enc` and `yahamm_dec` entities. It works in both synthesis and simulation.

function get_parity_check_matrix)

```
function get_parity_check_matrix (
  MESSAGE_LENGTH : natural;
  EXTRA_PARITY : natural range 0 to 1  := 1;
  ONE_PARITY_BIT : boolean := false)
  return matrix_t;
```

It returns the non-systematic form of parity check matrix, built following the general algorithm described in [1].

E.g. we want to get the parity-check matrix for the code Hamming(7,4) that encodes four bits of data into seven bits by adding three parity bits. The matrix has dimension 3x7 if no extra parity bit is added. The following snippet of test-bench:

```
entity test is
end entity test;

architecture std of test is
```

```
  constant MESSAGE_LENGTH   : natural              := 4;
  constant EXTRA_PARITY_BIT : natural range 0 to 1 := 0;
  constant ONE_PARITY_BIT   : boolean              := false;

  constant NPARITY_BITS : natural := calc_nparity_bits(MESSAGE_LENGTH, ONE
  constant BLOCK_LENGTH : natural := calc_block_length(MESSAGE_LENGTH, ONE

  constant H : matrix_t(0 to NPARITY_BITS + EXTRA_PARITY_BIT - 1,
                        0 to BLOCK_LENGTH + EXTRA_PARITY_BIT - 1) :=
    get_parity_check_matrix(MESSAGE_LENGTH, EXTRA_PARITY_BIT, ONE_PARITY_BI

begin

  process is
  begin

    pretty_print_matrix(H);

    stop(0);
  end process;

end architecture std;
```

outputs the parity-check matrix **H**: in non-systematic form:

```
1 0 1 0 1 0 1
0 1 1 0 0 1 1
0 0 0 1 1 1 1
```

If in the above code, `EXTRA_PARITY_BIT` is 1, the output is:

```
1 0 1 0 1 0 1 0
0 1 1 0 0 1 1 0
0 0 0 1 1 1 1 0
1 1 1 1 1 1 1 1
```

A right-most column and a bottom row has been added to the previous parity-check matrix. The addition of '1' in the bottom row means that the forth parity bit encodes the parity for the entire code word. No change on the existing parity bits ('0' in the last column for those parity bits).

If in the above code, `ONE_PARITY_BIT` is true, the output is:

```
1 1 1 1 1
```

That means that the only parity bit encodes the parity of the entire code word.

function get_form_swap_matrix)

```
  function get_form_swap_matrix (
    MESSAGE_LENGTH : natural;
    EXTRA_PARITY : natural;
    ONE_PARITY_BIT : boolean := false)
    return matrix_t;
```

Returns a $n \times n$ matrix **S** to convert a n-row matrix or vector from non-systematic form $\mathbf{M}_{ns}$ to systematic form $\mathbf{M}_bs$ and vice-versa:

$$\mathbf{M}_s = \mathbf{M}_{ns}\,\mathbf{S}$$

The construction of the matrix can be understood by noticing that the systematic form of the parity-check matrix $\mathbf{H}$ for the $Hamming(n,k)$ code is:

$$\mathbf{H} := (\mathbf{A}|\mathbf{I}_{n-k}) \tag{1}$$

where $\mathbf{I_{n-k}}$ is the $(n-k)\times(n-k)$ identify matrix . This can be obtained by swapping the columns 2^r-1-th of the identify matrix $\mathbf{I}_n$ corresponding to the position, in the non-systematic form, of the parity bits r with the column $n-k+r$. Note that since $\mathbf{S} = \mathbf{S}^T = \mathbf{S}^{-1}$, the matrix can be used to transform from systematic to non-systematic form and vice-versa.

Example:

```
entity test is
end entity test;

architecture std of test is

  constant MESSAGE_LENGTH   : natural              := 4;
  constant EXTRA_PARITY_BIT : natural range 0 to 1 := 0;
  constant ONE_PARITY_BIT   : boolean              := false;

  constant NPARITY_BITS : natural := calc_nparity_bits(MESSAGE_LENGTH, ONE
  constant BLOCK_LENGTH : natural := calc_block_length(MESSAGE_LENGTH, ONE

  constant H : matrix_t(0 to NPARITY_BITS + EXTRA_PARITY_BIT - 1,
                        0 to BLOCK_LENGTH + EXTRA_PARITY_BIT - 1) :=
    get_parity_check_matrix(MESSAGE_LENGTH, EXTRA_PARITY_BIT, ONE_PARITY_BI

  constant swap_matrix : matrix_t(0 to BLOCK_LENGTH + EXTRA_PARITY_BIT - 1
                                  0 to BLOCK_LENGTH + EXTRA_PARITY_BIT - 1)
    get_form_swap_matrix(MESSAGE_LENGTH, EXTRA_PARITY_BIT, ONE_PARITY_BIT)

begin

  process is
  begin

    pretty_print_matrix(H);
    pretty_print_matrix(swap_matrix);
    pretty_print_matrix(xor_multiply(H, swap_matrix));
    pretty_print_matrix(xor_multiply(xor_multiply(H, swap_matrix), swap_mat

    stop(0);
  end process;

end architecture std;
```

Outputs:

```
-- parity-check matrix, non-systematic form
1 0 1 0 1 0 1
0 1 1 0 0 1 1
0 0 0 1 1 1 1

-- form-swap matrix
0 0 0 0 1 0 0
0 0 0 0 0 1 0
```

```
0 0 1 0 0 0 0
0 0 0 0 0 0 1
1 0 0 0 0 0 0
0 1 0 0 0 0 0
0 0 0 1 0 0 0

-- parity-check matrix, systematic form
1 0 1 1 1 0 0
0 1 1 1 0 1 0
1 1 0 1 0 0 1

-- parity-check matrix, non-systematic form
1 0 1 0 1 0 1
0 1 1 0 0 1 1
0 0 0 1 1 1 1
```

The first three matrices are: the parity-check matrix for Hamming(7,4) in non-systematic form, the form swapping matrix, the parity-check matrix in systematic form. Note the identity matrix in the right-most columns in the systematic form of the parity-check matrix. The forth matrix is the parity-check matrix obtained by multiplying its systematic form for the form-swap matrix.

If `EXTRA_PARITY_BIT` is 1, the matrix has an additional column and row. Last column is the same as in the identity matrix (no swapping).

function get_code_generator_matrix

```
function get_code_generator_matrix (
  MESSAGE_LENGTH : natural;
  EXTRA_PARITY : natural range 0 to 1 := 1;
  ONE_PARITY_BIT : boolean := false)
  return matrix_t;
```

Returns the code generator matrix in systematic form. The construction algorithm, as suggested in [1], Sec. *Construction of G and H*, is based on the comparison of G in systematic form:

$$\mathbf{G} := (\mathbf{I}_k | \mathbf{A}^T) \tag{2}$$

with the expression of the parity-check matrix in systematic form in Eq. 2. So the left hand side of $\mathbf{H}$ in systematic form can be transposed and combined with the $\mathbf{I}_k$ identity matrix.

E.g. we want to get the code generator matrix for the code Hamming(7,4) that encodes four bits of data into seven bits by adding three parity bits. The matrix has dimension 7x4 if no extra parity bit is added. The following snippet of test-bench:

```
architecture std of test is

  constant MESSAGE_LENGTH   : natural              := 4;
  constant EXTRA_PARITY_BIT : natural range 0 to 1 := 0;
  constant ONE_PARITY_BIT   : boolean              := false;

  constant NPARITY_BITS : natural := calc_nparity_bits(MESSAGE_LENGTH, ONE
  constant BLOCK_LENGTH : natural := calc_block_length(MESSAGE_LENGTH, ONE

  constant G : matrix_t(0 to BLOCK_LENGTH + EXTRA_PARITY_BIT - 1,
```

```
                                0 to BLOCK_LENGTH - NPARITY_BITS - 1) :=
    get_code_generator_matrix(MESSAGE_LENGTH, EXTRA_PARITY_BIT, ONE_PARITY_

begin

  process is
  begin

    pretty_print_matrix(G);

    stop(0);
  end process;

end architecture std;
```

Outputs:

```
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
1 0 1 1
0 1 1 1
1 1 0 1
```

If in the above code, `EXTRA_PARITY_BIT` is 1, the output is:

```
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
1 0 1 1
0 1 1 1
1 1 0 1
1 1 1 0
```

A bottom row has been added to the previous code generator matrix, based on the parity of the rows of the parity-check matrix.

If in the above code, `ONE_PARITY_BIT` is true, the output is:

```
1 0 0 0
0 1 0 0
0 0 1 0
0 0 0 1
1 1 1 1
```

Where the result is a row of '1' at the bottom of an identity matrix. That means that the code word is the message plus its parity bit.

References

[1] Wikipedia. Hamming code — Wikipedia, the free encyclopedia, 2017. [Online; accessed 19-Mar-2017].

[2] Wikipedia. Hamming(7,4) — Wikipedia, the free encyclopedia, 2017. [Online; accessed 19-Mar-2017].