YAHAMM
Yet Another Hamming Encoder and Decoder
—
Specification

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

Contents

1 Project files **2**

2 Makefile **2**

2.1 Analyze the VHDL 2

2.2 Run the simulation 2

2.3 Cleanup 2

3 Encoder **3**

4 Decoder **4**

5 SEC-DED and other possibilities **5**

Abstract

YAHAMM is a VHDL implementation of an Hamming encoder, as entity yahamm_enc, and and Hamming decoder, as entity yahamm_dec, of message length defined trough generic. All the math needed to compute the code is internally implemented in VHDL and it is transparent to the user. Single Error Correction and Double Error Detection is also provided (see next for details). Encoder has latency 1 clock cycle, decoder latency 2, independently from their configuration.

1 Project files

The projects includes the file:

1. rtl/vhdl/yahamm_enc.vhd: encoder entity
2. rtl/vhdl/yahamm_dec.vhd: decoder entity
3. rtl/vhdl/matrix_pkg.vhd: help functions for matrix manipulation
4. rtl/vhdl/yahamm_pkg.vhd: core math functions
5. bench/vhdl/yahamm_tb[0-3].vhd: test benches
6. sim/rtl_sim/bin/Makefile: a set of rules for the analysis and simulation with GHDL.

2 Makefile

The Makefile targets call ghdl for analyzing the VHDL and running simulations.

2.1 Analyze the VHDL

1. `cd sim/rtl_sim/bin`
2. `make <vhdl file>`

2.2 Run the simulation

1. `cd sim/rtl_sim/bin`

To simulate a single test-bench:

2 `make simulate_tb[0..3]`

or, to simulate all test-benches:

2 `make simulate`

Each test-bench will output "OK" if successful.

The simulation will create waveform as VCD compressed file in `cd sim/rtl_sim/out`. They can be view, for example, with gtkwave by running:

3 `make view_[0..3]`

2.3 Cleanup

1. `cd sim/rtl_sim/bin`
2. `make clean`

3 Encoder

```
entity yahamm_enc is
  generic (
    MESSAGE_LENGTH       : natural := 5;
    EXTRA_PARITY_BIT : natural range 0 to 1 := 1;
    ONE_PARITY_BIT : boolean := false
    );
  port(
    clk_i, rst_i : in  std_logic;
    en_i       : in  std_logic := '1';          -- Synchronous output enable
    data_i        : in  std_logic_vector(MESSAGE_LENGTH - 1 downto 0);  -- Inp
    data_o        : out std_logic_vector(MESSAGE_LENGTH - 1 downto 0);  -- Out
    data_valid_o : out std_logic;        -- High when data_o is valid.
    parity_o   : out std_logic_vector(calc_nparity_bits(MESSAGE_LENGTH, ONE_PA
    );

end yahamm_enc;
```

Generics:

1. MESSAGE_LENGTH: the data payload length, excluding parity bits.
2. EXTRA_PARITY_BIT: if 1 the Hamming codes are extended by an extra parity bit. This way the minimum Hamming distance is increased 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 (SECDED). If the decoder does not attempt to correct errors, it can detect up to three errors. . Default is 1.

→ Sec 4

3. ONE_PARITY_BIT: if true the encoder does not perform Hamming encoding but just adds a single parity bit to the message. Default is false.

The generics MESSAGE_LENGTH, EXTRA_PARITY_BIT and ONE_PARITY_BIT must have the same values on both the encoder and the decoder instances.

Ports:

1. clk_i: input reset, active high.
2. rst_i: input reset, active high.
3. en_i: input enable. When asserted input data are encoded. If deasserted, data and parity outputs are zero. Default is high.
4. data_i: input data. Length equal to MESSAGE_LENGTH.
5. data_o: output data. Length equal to MESSAGE_LENGTH.
6. data_valid_o: output valid signal for data_o. This is just en_i delayed by the core latency.
7. parity_o: output parity bits. The length of this vectors corresponds to the number of parity bits needed by the hamming code and it's computed by the VHDL function calc_nparity_bits(). One extra parity bit is needed if the generic EXTRA_PARITY_BIT is true. The user instantiating the entity and mapping that port can determine the length in three ways:

(a) Calculate the number of parity bits r needed for the specified message length k, looking for the smallest r such that $k \geq 2^r - r - 1$ for $r \geq 2$.

(b) Use the script `sw/calc_nparity_bits.sh`.

(c) Connect a bus or arbitrary random length. Run any VHDL analysis tool and use the syntax error you get to see which is the right length.

4 Decoder

```
entity yahamm_dec is
  generic (
    MESSAGE_LENGTH       : natural := 5;
    CORRECT : boolean := true;
    EXTRA_PARITY_BIT : natural range 0 to 1 := 1;
    ONE_PARITY_BIT : boolean := false;
    ERROR_LEN : natural := 16
    );
  port(
    clk_i, rst_i : in  std_logic;
    cnt_clr_i : in std_logic := '0';             -- Clear monitor counters.
    en_i       : in  std_logic := '1';          -- Input enable.
    data_i        : in  std_logic_vector(MESSAGE_LENGTH - 1 downto 0);  -- Inp
    parity_i   : in std_logic_vector(calc_nparity_bits(MESSAGE_LENGTH, ONE_PAR
    data_o        : out std_logic_vector(MESSAGE_LENGTH - 1 downto 0);  -- Out
    dout_valid_o : out std_logic;                                       --
    cnt_errors_corrected_o, cnt_errors_detected_o : out std_logic_vector(ERROR
    log_wrong_bit_pos_data_o : out std_logic_vector(MESSAGE_LENGTH - 1 downto
    log_wrong_bit_pos_parity_o : out std_logic_vector(calc_nparity_bits(MESSAG
    );

end yahamm_dec;
```

Generics:

1. `CORRECT`: if true, the core corrects detected errors. Default is true.
2. `ERROR_LEN`: bit length of the error counters provided on ports `cnt_errors_corrected_o` and `cnt_errors_detected_o`.
3. `MESSAGE_LENGTH`: see Sec. 3.
4. `EXTRA_PARITY_BIT`: see Sec. 3.
5. `ONE_PARITY_BIT`: see Sec. 3.

The generics MESSAGE_LENGTH, EXTRA_PARITY_BIT and ONE_PARITY_BIT must have the same values on both the encoder and the decoder instances.

Ports:

1. `cnt_clr_i`: input synchronous clear of the error counters.
2. `en_i`: optional input enable. Can be connected to `dout_valid_o` of the encoder to propagate a valid signal to the output port `data_valid_o`: Default is high (`data_valid_o` will be always asserted in this case).
3. `data_i`: input data. It must be connected to the `data_o` of the decoder. Length equal to MESSAGE_LENGTH.

	CORRECT	
EXTRA_PARITY_BIT	false	true
false	SED-DED	SEC
true	SED-DED-TED	SEC-DED

Table 1: SEC = single bit error corrected, SED = single bit error detected, DED = double bit error detected, TED = triple bit error detected

4. `parity_i`: input parity bits. It must be connected to the `parity_o` of the decoder.
5. `data_o`: output data. Length equal to MESSAGE_LENGTH.
6. data_valid_o: output valid signal for data_o. This is just en_i delayed by the core latency.
7. `cnt_errors_corrected_o`: counter of single errors corrected. It is always zero if the generic `CORRECT` is false.
8. `cnt_errors_detected_o`: counter of errors detected. It includes single, double and triple errors (not there's no way to distinguish between a single and a triple error), if the generic `CORRECT` is false, otherwise it only counts double errors.
9. `log_wrong_bit_pos_data_o`: position of the single error in `data_o`. Zero if the generic `CORRECT` is false or if it's not a single error.
10. `log_wrong_bit_pos_data_o`: position of the single error in `parity_o`. Zero if the generic `CORRECT` is false or if it's not a single error.

5 SEC-DED and other possibilities

In order to have SEC-DED (Single Error Corrected - Double Error Detected) capability, user should leave the default values for the generics `EXTRA_PARITY_BITS` to 1, and `CORRECT` to true.

Other choices can be made as described by the following table.

The generic `EXTRA_PARITY_BITS` set to true (default) adds a parity bit so that the Hamming distance is 4 between two code words (number of bits to flip in a code word to obtain another code word). With `EXTRA_PARITY_BITS` set to false the Hamming distance will be 3.

The generic `CORRECT` set to true (default) corrects for single bit errors. It can be set to false if only error detection is needed. This allows triple error detection with `EXTRA_PARITY_BIT` set to true. Indeed, a triple error is indistinguishable from a single error with an Hamming distance 4 and the correction would be wrong in this case. Similarly, with `EXTRA_PARITY_BIT` set to false, hence with Hamming distance 3, a double error is indistinguishable from a single error and the correction would be wrong. Note that if `CORRECT` is false, the port `cnt_errors_detected_o` provides the value of a counter that sums up any kind of error detected (single, double and triple).

For most applications, SEC-DED configuration (the default) is the preferred choice. A communication channel can be considered reliable if the probability of double error is negligible. A noisy channel showing double errors detected is not to be trusted because event a corrected single error may be a triple error and the correction would be wrong.