An Assignment On

DESIGN AND VERIFICATION OF IMPROVED HAMMING CODE ON FPGA USING VERILOG

Submitted by

Name: Forhad Hossain
Roll: 1515032
Reg:1166
Session: 2015-2016

Submitted To

Dr. Md. Shahinuzzaman
Professor
Dept. of EEE

Dept. of Electrical & Electronic Engineering.
Islamic University, Kushtia.

ABSTRACT

In mathematics, digital communication, and information theory, error detection and correction are of great practical importance in maintaining data integrity in noisy channels. Error coding is considered a method to detect and correct these errors to ensure that data is transmitted from source to destination without corruption. There are several debugging techniques to detect and fix the error. One of the most popular techniques based on error correction is Hamming Code. This paper focuses on the design and its hardware implementation using a Field Programmable Gate Array (FPGA). The design includes both encoder and decoder systems used for data transmission and reception by wireless transmitter systems. The design is simulated and verified using the ISim simulator and Verilog HDL. The implementation uses the Spartan-3 FPGA training kit for Xilinx 14.3.

1.0 INTRODUCTION

In digital communication systems, environmental disturbances and physical errors can cause random bit errors during data transmission. Error coding is a method for detecting and correcting these errors in various computer memories, magnetic and optical media, satellite and deep space, network communications, cellular networks and almost all other digital data. communication Digital data is transmitted over a channel, and the channel is often noisy. Noise can distort the messages being sent. Therefore, what the receiver receives may not be the same as what the sender sends. The purpose of error coding is to improve the reliability of digital communications by detecting and correcting errors.

1.1 ERROR

Information that is transmitted through a communication channel is not completely error-free. This change in data is caused by external interference, signal distortion, attenuation or noise. There are two types of errors. First, a single error where only one bit changes. And second, an interrupt error where more than one bit changes. There are several error detection and correction techniques such as cyclic redundancy check (CRC), parity, LRC, VRC and Hamming code. This work focuses on the Hamming code.

1.2 HAMMING CODE

Hamming code is usually known as linear Block code. In a block of data, hamming codes can find and fix a single bit error. In these codes, each bit is contained in a unique set of parity bits. The presence and location of a single parity error can be determined by analyzing the parities of received bit combinations to create a parity table, each corresponding to a particular request combination. This table of errors is called error syndrome. If all equalities are correct according to this model, it can be concluded that there are no bit errors in the message (there can be several bit errors). If there are errors in the parities caused by a single bit error, the incorrect data bit can be found by looking at the incorrect parities in the adder. Hamming

codes are easy to implement and used Hamming codes are commonly used in computing, telecommunications, and other applications, including data compression and turbo codes. They are also used in low cost and low power applications.

1.3 FPGA

A Field-Programmable Gate Array (FPGA) is a semiconductor device that can be configured by the customer or designer after manufacturing - hence the name "field-programmable." FPGAs are programmed using hardware description language (HDL) source code to define how the chip works. The Spartan 3 FPGA is an affordable, high-performance logic solution for large consumer applications. It also provides an easy way to test various programs on the FPGA itself by dropping a "bit" file into the FPGA and then observing the output. Figure 1. Shows Xilinx's XC3S4O0-4PQ208 Spartan 3 FPGA tutorial.

Figure 1: Spartan 3 FPGA trainer kit

The Spartan 3 FPGA board is built-in with many peripherals that help in the operation of the board and also in connecting various signals to the board itself. Some of the peripherals on the Spartan 3 FPGA board include an LCD display, a keyboard port, a VGA port, two 9-pin RS-232 ports, a 50 MHz clock oscillator, a USB-based FPGA download and debug interface, and 32 I/Os. pins for UI. Verilog source code was edited and synthesized using xilinx14.3 and then simulated using ISim. The Spartan-3 FPGA Training Kit for Xilinx was used to load the design (.bit file) onto the FPGA chip.

2.1 DESIGN FLOW FOR FPGA

The process of implementing a design on an FPGA can be divided into several steps, which can be loosely defined as design input or capture, synthesis, and place and route. Figure 2 shows the design steps required to implement the design using an FPGA.

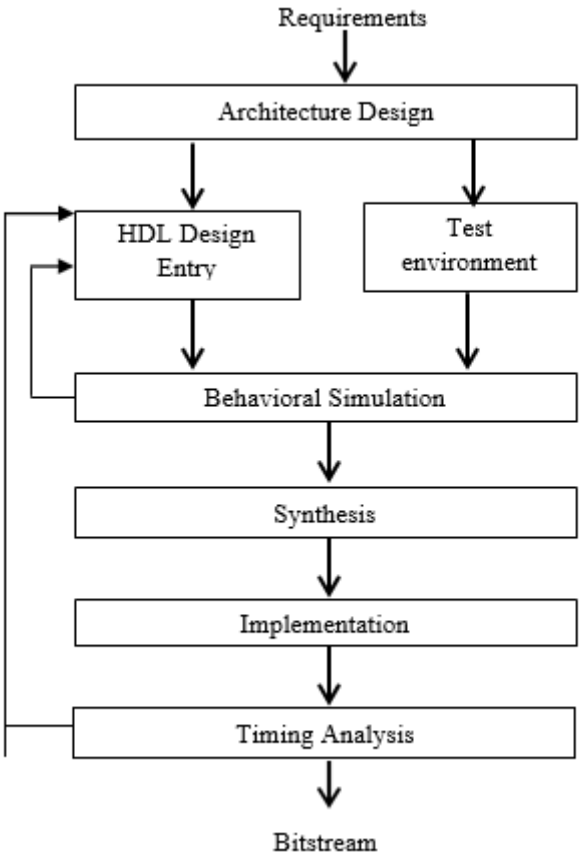


Figure 2: Schematic Showing design flow for FPGA

2.2 ENCODING OF HAMMING CODE

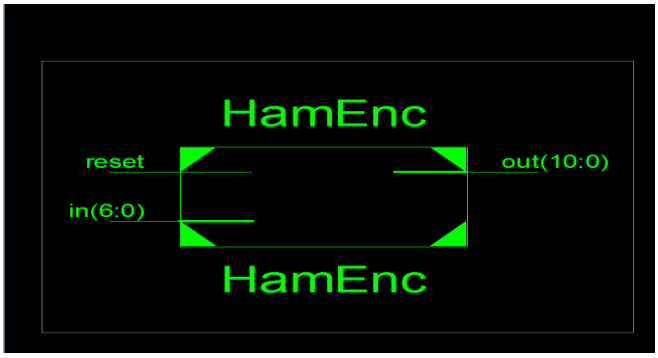
To calculate the number of redundant bits (r) needed to correct d data bits, the relationship between the two is determined. Thus, the total number of bits to be transmitted is (d+r); then r must be able to express at least d+r+ 1 different values. One value of these means no error, and the remaining d+r values indicate the location of the error at each dr position. Thus, d+r+1 state must be separable by r bits, and r bits can point to 2r state. Therefore, 2r must be greater than d+r+1. The value of R must be determined by putting the value of d into the relationship. For example, if d is 7, then the smallest value of r that satisfies the above relationship is 4. Thus, the total number of bits to be transmitted is 11 bits (d+r = 7+4 = 11). These redundancy bits are placed in positions 1, 2, 4, and 8 (positions in the 11-bit sequence that are powers of 2). For clarity, in the examples below, these bits are called r1, r2, r3, and r4. In the Hamming code, each r-bit is the parity of one combination of data bits, as shown below:

r1: 1, 3, 5, 7, 9, 11
r2: 2, 3, 6, 7, 10, 11
r4: 4, 5, 6, 7
r8: 8, 9, 10, 11

11	10	9	8	7	6	5	4	3	2	1
d_6	d_5	d_4	r_8	d_3	d_2	d_1	r_4	d_0	r_2	r_1
Position of redundancy bits in Hamming Code										
Data:1010101										
11	10	9	8	7	6	5	4	3	2	1
1	0	1		0	1	0		1		
Adding r_1										
1	0	1		0	1	0		1		**1**
Adding r_2										
1	0	1		0	1	0		1	**1**	**1**
Adding r_4										
1	0	1		0	1	0	**1**	1	**1**	**1**
Adding r_8										
1	0	1	**0**	0	1	0	1	1	**1**	**1**
Encoded data:10100101111										

Figure 3: Redundancy bit calculation

Top level of Hamming Encoder:

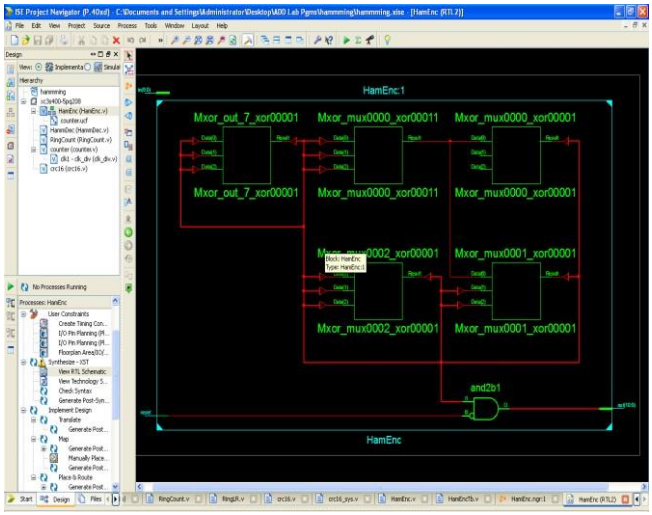


2.3 VERILOG DESIGN CODE FOR ENCODE

```
module hamming_encoder(output[16:1] out, input[10:0] in);

xor(out[1], in[0], in[1], in[3], in[4], in[6], in[8], in[10]);      //p1
xor(out[2], in[0], in[2], in[3], in[5], in[6], in[9], in[10]);      //p2
assign out[3] = in[0];
xor(out[4], in[1], in[2], in[3], in[7], in[8], in[9], in[10]);      //p3
assign out[5] = in[1];
assign out[6] = in[2];
```

```
assign out[7] = in[3];
xor(out[8], in[4], in[5], in[6], in[7], in[8], in[9], in[10]);        //p4
assign out[9] = in[4];
assign out[10] = in[5];
assign out[11] = in[6];
assign out[12] = in[7];
assign out[13] = in[8];
assign out[14] = in[9];
assign out[15] = in[10];
assign out[16] = out[1] + out[2] + out[3] + out[4] + out[5] + out[6] + out[7] + out[8] + out[9] +
out[10] + out[11] + out[12] + out[13] + out[14] + out[15];

endmodule
```

2.4RTL VIEW OF HAMMING ENCODER



3.0 DECODING, DETECTION AND CORRECTION OF HAMMING CODE

Error position	Binary value of position			
0 (no error)	C2	C2	C1	C0
1	0	0	0	0
2	0	0	0	1
3	0	0	1	0
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
10	1	0	1	0
11	1	0	1	1

At the receiving end, the parity bits are recalculated. The decimal value of the k parity bit gives the incorrect position of the bit, if present. The table above shows that the Hamming code is used to correct 7-bit numbers (d7 d6 d5 d4 d3 d2 d1) using four redundant bits (r4 r3 r2 r1). With error positions (C3 C2 C1 C0) For example, for the data 100100101111, the first r1 is calculated considering the equality of bit positions 1, 3, 5, 7,9,11. Second, the parity bits r2 are calculated considering bit positions 2, 3, 6, 7, 10, 11, then the parity bits r4 are is calculated considering bit positions 4, 5, 6 and 7, finally the parity bits r8 are calculated considering bit positions 8, 9, 10, 11 as shown. If errors occur in any of the 10000101111 transmitted codes, the error bit position can be determined by counting r4 r3 r2 r1 at the receiving end. For example, if the received code word is 1000010111, the recalculated value of r4 r3 r2 r1 is 1001, indicating that the error bit position is 9, the decimal value is 1001.

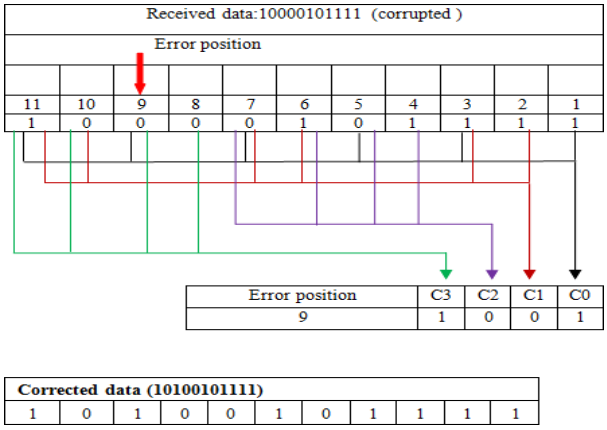
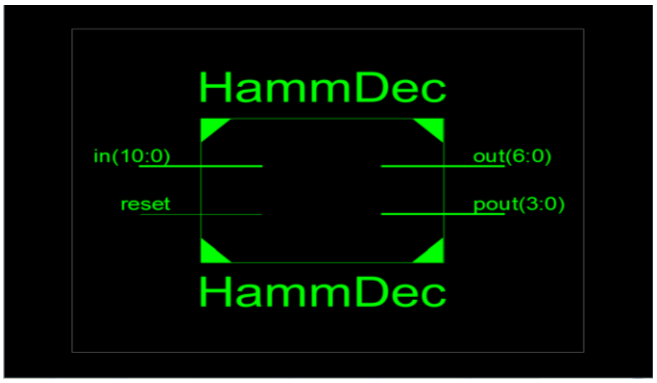


Fig-4: Error detection and correction using Hamming code

Top Level of Hamming Decoder:



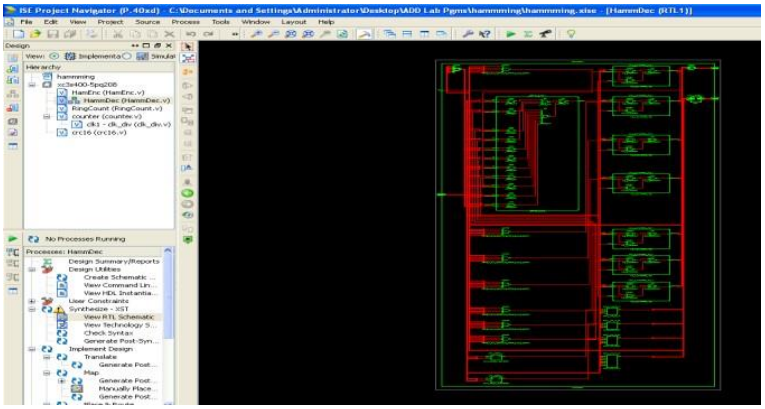
3.1 VERILOG DESIGN CODE FOR DECODE

```
module hamming_decoder(output[10:0] out,
                       output[3:0] error_index,
                       output reg error,
                       output reg uncorrectable,
                       input[16:1] in);
```

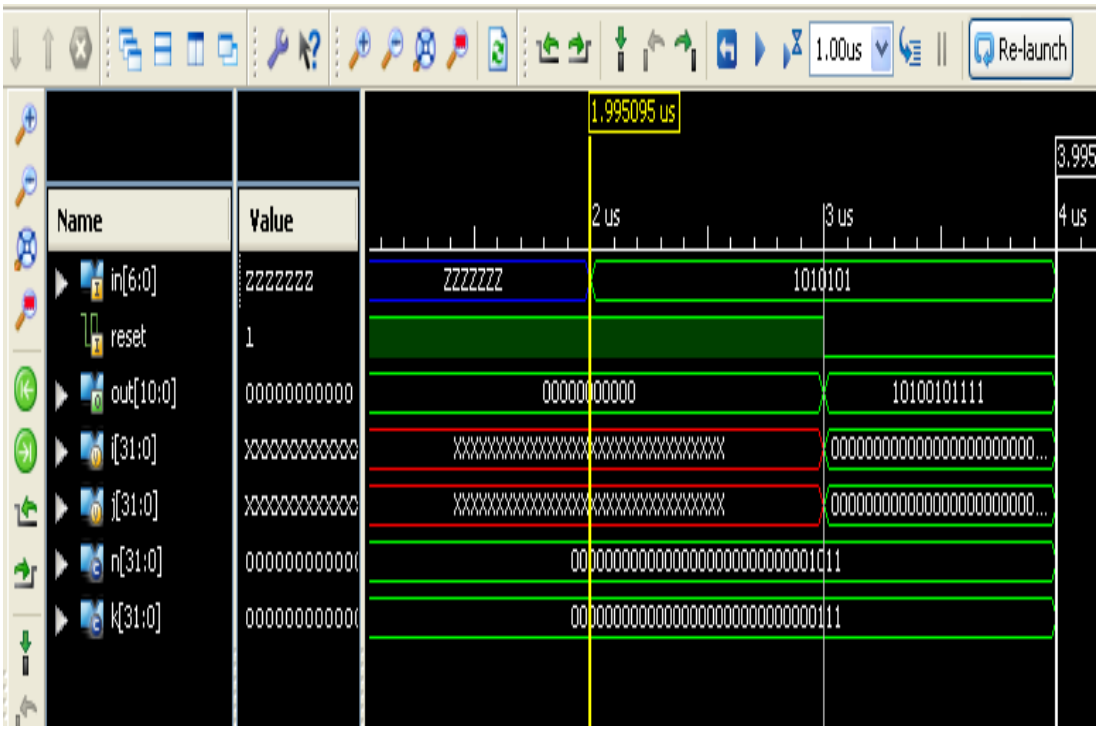
```
reg correction_parity;
integer i;
reg out;
reg error_index;
reg error;
reg uncorrectable;


always @(*)
begin
assign error_index[0] = in[1] + in[3] + in[5] + in[7] + in[9] + in[11] + in[13] + in[15];
assign error_index[1] = in[2] + in[3] + in[6] + in[7] + in[10] + in[11] + in[14] + in[15];
assign error_index[2] = in[4] + in[5] + in[6] + in[7] + in[12] + in[13] + in[14] + in[15];
assign error_index[3] = in[8] + in[9] + in[10] + in[11] + in[12] + in[13] + in[14] + in[15];

if(error_index == 4'b0000) begin
assign error = 0;
assign uncorrectable = 0;

assign out[0] = in[3];
assign out[1] = in[5];
assign out[2] = in[6];
assign out[3] = in[7];
assign out[4] = in[9];
assign out[5] = in[10];
assign out[6] = in[11];
assign out[7] = in[12];
assign out[8] = in[13];
assign out[9] = in[14];
assign out[10] = in[15];
end else begin
assign error = 1;
assign uncorrectable = 0;

for(i=1; i<=15; i=i+1) begin
if(i == error_index) begin
assign in[i] = !in[1];
          end
end
```

```
assign correction_parity = in[1] + in[2] + in[3] + in[4] + in[5] + in[6] + in[7] + in[8] + in[9] +
in[10] + in[11] + in[12] + in[13] + in[14] + in[15];

if(correction_parity != in[16]) begin
assign uncorrectable = 1;
                    end
          end
end
endmodule
```

3.2 RTL VIEW OF HAMMING DECODER



4.0 SIMULATION RESULT



The Hamming Encoder result of the simulation is shown in Figure 11. Which is an active low digital system with 7 bits of information bits [6:0] as inputs. 11-bit output data field [10:0] with the encoded data of the encoder system. Figure 11 shows an example of 7-bit input data 1010101 as input and 10100101111 is 11-bit code data. The simulation results of the Hamming decoder are shown in Figure 12, which shows its active low digital system with 11-bit input data [10:0] and 7-bit error-corrected data at input [6:0] and other 4-bit output data containing. position of the error orbit in the received data output [3:0]. Figure 12 shows Hamming decoding with an example; the received data is 10000101111 and the decoder data output is 1010101, the error position of the input data is 1001.

CONCLUSION

An FPGA design of a Hamming encoder and decoder with error detection and correction functions is simulated and implemented. Implemented on hardware using Xilinx 14.3, Spatan-3 FPGA starter kit. Both Hamming encoding and decoding are provided for inputting different data in the form (11,7,1) both in simulation and in FPGA implementation.

REFERENCES

1. Leena, Mr. Subham Gandhi and Mr. Jitender Khurana, “Implementing (7,4) Hamming Code using CPLD on VHDL” International Journal of New Trends in Electronics, Vol. 1, Issue 1, Aug. 2013.
2. Xilinx “Spartan-3 FPGA Family, complete datasheet”, Xilinx Corp., Aug 2005.
3. Xilinx “Synthesis and Simulation Design Guide”, Xilinx Tech UG626 2012.
4. Ming- Bo Lin “Digital System Design and Practices using Verilog HDL and FPGA”, Wiley-India, ISBN:978-81-265-3694-8.