

مدارهای واسط

زمستان ۱۴۰۳

اشکان تاریوردی ۵۷۵۳ - ۱۱۰۵۷۵ - روزین تقیزادگان



دانشکده‌ی مهندسی کامپیوتر

گزارش پژوهش

۱ رمزگذاری 64B/66B

در رمزگذاری 64B/66B، ۲ بیت پرارزش^۱ داده رمزگذاری شده، سربرگ همگام‌سازی^۲ و ۶۴ بیت باقی‌مانده داده^۳ هستند. در این رمزگذاری به صورت زیر است:

preamble	bits payload
00	error code
01	field data
10	field data/control mixed
11	error code

جدول ۱: جدول دو بیت پرارزش

۲ تبدیل رمزگذاری 8B/10B به 64B/66B

برای تبدیل رمزگذاری 64B/66B به 8B/10B لازم است ابتدا دو بیت header رمزگذاری 64B/66B را جدا کنیم. سپس ۶۴ بیت باقی‌مانده را به ۸ بخش ۸ بیتی تقسیم می‌کنیم و هر بخش را با توجه به بیت‌های header و با استفاده از رمزگذاری 8B/10B رمزگذاری می‌کنیم.

۳ رمزگذاری 8B/10B

۱.۳ پیاده‌سازی

در این رمزگذاری، ۸ بیت داده در قالب یک character یا symbol می‌شود. ۵ بیت کم‌ارزش داده با استفاده از رمزگذاری 5B/6B به یک گروه ۶ بیتی تبدیل می‌شود. همچنین ۳ بیت پرارزش داده با استفاده از رمزگذاری 3B/4B به یک گروه ۴ بیتی تبدیل می‌شود. این دو گروه در نهایت کنار هم قرار می‌گیرند و یک symbol ۱۰ بیتی را تشکیل می‌دهند که روی سیم فرستاده می‌شود.

Bit Significant Most^۱
sync header^۲
payload^۳

symbol های داده به صورت D.x.y نشان داده می شوند که x بین صفر تا ۳۱ و y بین صفر تا ۷ است.

استانداردهایی که از رمزگذاری ۸B/10B استفاده می کنند همچنین ۱۲ symbol خاص یا symbol کنترلی تعریف می کنند که به جای symbol داده فرستاده می شوند. آنها غالبا برای نشان دادن آغاز frame، پایان frame، بیکار بودن لینک و موارد مشابه استفاده می شوند. این symbol ها به صورت K.x.y نشان داده می شوند.

از آن جا که رمزگذاری ۸B/10B از symbol های ۱۰ بیتی برای رمزگذاری داده های ۸ بیتی استفاده می کند، برخی از ۱۰۲۴ symbol موجود استفاده نمی شوند تا تضمین شود که در این رمزگذاری اختلاف تعداد بیت های صفر و یک بیشتر از دو نمی شود. همچنین برخی از ۲۵۶ داده ۸ بیتی می توانند به دو صورت رمزگذاری شوند. استفاده از این رمزگذاری های جایگزین باعث می شود که این رمزگذاری بتواند DC-balance را به صورت طولانی مدت حفظ کند.

Running disparity ۲.۳

رمزگذاری ۸B/10B عمل می کند، به این معنی که در یک بازه طولانی مدت تعداد بیت های صفر و یک فرستاده شده با هم برابر هستند. برای دستیابی به این قابلیت، تفاوت تعداد صفرها و یکهای فرستاده شده همواره محدود به ۲ است و در انتهای در symbol برابر با ۱ + ۱ - است. این تفاوت به عنوان running disparity یا RD نشانه می شود.

این رمزگذاری تنها به دو حالت ۱ + ۱ - نیاز دارد. کدگذاری از حالت ۱ - آغاز می شود.

برای هر کد ۵B/۶B و ۳B/۴B که تعداد بیت های صفر و یک نابرابر دارند، دو الگوی بیتی قابل استفاده وجود دارد که در یک الگو تعداد بیت های ۱ بیشتر است و در الگوی دیگر تعداد بیت های صفر بیشتر است. با توجه به running disparity کنونی سیگنال، رمزگذار تصمیم می گیرد که کدام گروه ۶ بیتی یا ۴ بیتی را برای ارسال داده انتخاب کند. واضح است در صورتی که گروه های ۶ بیتی و ۴ بیتی تعداد بیت های صفر و یک برابر داشته باشند، تنها از یک الگو استفاده می شود.

۴ پیاده سازی به کمک زبان توصیف سخت افزار

۱.۴ ماژول رمزگذاری ۸B/10B

۱.۱.۴ ورودی ها

```
3      input wire clk,          // Clock signal
4      input wire rst,          // Reset signal
5      input wire en,           // Enable signal
6      input wire kin,          // K- or D-symbol selection (1 - K, 0 - D)
7      input wire [7:0]data_in,  // 8-bit input data
```

شکل ۱: ورودی های ماژول رمزگذاری ۸B/10B

• clk: سیگنال clk به منظور هم گام سازی

• rst: سیگنال rst به منظور reset کردن رمزگذار

• en: سیگنال en به منظور فعال سازی رمزگذار

• kin: سیگنال kin تعیین می کند که آیا ورودی کاراکتر کنترلی است یا خیر (K-symbol)

• ۸ بیت داده‌ی ورودی برای رمزگذاری :data-in

۲.۱.۴ خروجی‌ها

```
8   output wire [9:0]data_out, // 10-bit encoded data
9   output wire disp,        // Disparity output wire
10  output wire kin_err;    // K-symbol error output wire
11 );
```

شکل ۲: خروجی‌های ماثول رمزگذاری 8B/10B

• ۱۰ بیت داده‌ی خروجی که رمزگذاری شده است :data-out

• DC-Balance disp به منظور نشان دادن Disparity

• سیگنال kin-err به منظور نشان دادن اینکه آیا K-Symbol نامعتبر تشخیص داده شده است یا خیر

۳.۱.۴ ثبات‌ها و متغیرهای موقت

```
13 reg tmp_disp;           // Temporary disparity register
14 reg tmp_k_err;          // Temporary K-symbol error register
15 reg [18:0]t;            // Temporary register for encoding
16 reg [9:0]tmp_data_out; // Temporary encoded data output
17 wire [7:0]tmp_data_in; // Temporary input data
18 wire tmp_kin;           // Temporary K-symbol selection
```

شکل ۳: ثبات‌های ماثول رمزگذاری 8B/10B

• این ثبات مقدار Disparity رمزگذاری فعلی را ذخیره می‌کند :tmp-disp

• این ثبات بررسی می‌کند که به هنگام رمزگذاری K-Symbol، خط رخ داده است یا خیر :tmp-k-err

• t: این ثبات موقت ۱۹ بیتی است و مقادیر میانی به هنگام محاسبه رمزگذاری را ذخیره می‌کند

• tmp-data-out: این ثبات ۱۰ بیت نهایی رمزگذاری شده را ذخیره می‌کند

• tmp-data-in: این سیم داده‌ی ورودی را ذخیره می‌کند

• tmp-kin: این سیم مقدار kin ورودی را ذخیره می‌کند

٤.١.٤ الگوریتم رمزگذاری

```

27   always @(posedge clk) begin
28     if (rst) begin
29       tmp_disp <= 1'b0;
30       tmp_k_err <= 1'b0;
31       tmp_data_out <= 10'b0;
32     end else begin
33       if (en == 1'b1) begin
34         tmp_disp <= ((tmp_data_in[5]&tmp_data_in[6]&tmp_data_in[7])|(!tmp_data_in[5]&!tmp_data_in[6]&tmp_data_in[7]));
35         tmp_k_err <= (tmp_kin&(tmp_data_in[0]|tmp_data_in[1]|!tmp_data_in[2]|!tmp_data_in[3]));
36         tmp_data_out[9] <= t[12]^t[0];
37         tmp_data_out[8] <= t[12]^t[1]|t[2];
38         tmp_data_out[7] <= t[12]^t[3]|t[4];
39         tmp_data_out[6] <= t[12]^t[5];
40         tmp_data_out[5] <= t[12]^t[6]|t[7];
41         tmp_data_out[4] <= t[12]^t[8]|t[9]|t[10]|t[11];
42         tmp_data_out[3] <= t[13]^t[15]&!t[14];
43         tmp_data_out[2] <= t[13]^t[16];
44         tmp_data_out[1] <= t[13]^t[17];
45         tmp_data_out[0] <= t[13]^t[18]|t[14]);
46       end
47     end
48   end

```



```

50   always @(posedge clk) begin
51     if(rst) begin
52       t <= 0;
53     end else begin
54       if (en == 1'b1) begin
55         t[0] <= tmp_data_in[0];
56         t[1] <= tmp_data_in[1]&!tmp_data_in[0]&tmp_data_in[1]&tmp_data_in[2]&tmp_data_in[3];
57         t[2] <= (!tmp_data_in[0]&!tmp_data_in[1]&!tmp_data_in[2]&!tmp_data_in[3]);
58         t[3] <= (!tmp_data_in[0]&!tmp_data_in[1]&!tmp_data_in[2]&!tmp_data_in[3])|tmp_data_in[2];
59         t[4] <= tmp_data_in[4]&tmp_data_in[3]&!tmp_data_in[2]&!tmp_data_in[1]&!tmp_data_in[0];
60         t[5] <= tmp_data_in[3]&!tmp_data_in[0]&tmp_data_in[1]&tmp_data_in[2];
61         t[6] <= tmp_data_in[4]|(!tmp_data_in[0]&tmp_data_in[1])|(!tmp_data_in[0]&!tmp_data_in[1]);
62         t[7] <= !(tmp_data_in[4]&tmp_data_in[3]&!tmp_data_in[2]&!tmp_data_in[1]&!tmp_data_in[0]);
63         t[8] <= (((tmp_data_in[0]&tmp_data_in[1]&!tmp_data_in[2]&!tmp_data_in[3])|(tmp_data_in[2]&t[1]));
64         t[9] <= tmp_data_in[4]&!tmp_data_in[3]&!tmp_data_in[2]&!tmp_data_in[1];
65         t[10] <= tmp_kin&tmp_data_in[4]&tmp_data_in[3]&tmp_data_in[2]&!tmp_data_in[1]&!tmp_data_in[0];
66         t[11] <= tmp_data_in[4]&!tmp_data_in[3]&tmp_data_in[2]&!tmp_data_in[1]&!tmp_data_in[0];
67         t[12] <= (((tmp_data_in[4]&tmp_data_in[3])&!tmp_data_in[2]&!tmp_data_in[1])&!tmp_data_in[0]);
68         t[13] <= (((tmp_data_in[5]&!tmp_data_in[6])|(tmp_kin&((tmp_data_in[5]&!tmp_data_in[6])|(!tmp_data_in[5]&tmp_data_in[6]))));
69         t[14] <= tmp_data_in[5]&tmp_data_in[6]&tmp_data_in[7]&(tmp_kin|tmp_disp?(!tmp_data_in[4]&tmp_data_in[5])&tmp_data_in[6]);
70         t[15] <= tmp_data_in[5];
71         t[16] <= tmp_data_in[6]|(!tmp_data_in[5]&!tmp_data_in[6]&!tmp_data_in[7]);
72         t[17] <= tmp_data_in[7];
73         t[18] <= !tmp_data_in[7]&(tmp_data_in[6]^tmp_data_in[5]);
74       end
75     end
76   end

```

شکل ٤: الگوریتم مازول رمزگذاری 8B/10B

Disparity : در هر چرخه، مقدار Disparity به روزرسانی می شود.

```

1   tmp_disp <= ((tmp_data_in[5] & tmp_data_in[6] & tmp_data_in[7]) |(!tmp_data_in[5] & !tmp_data_in[6])) ^ (tmp_disp ^ (complex parity logic));

```

این خط اطمینان حاصل می کند که تفاضل بین تعداد یک ها و صفرها در دادهی رمزگذاری شده، از یک بیشتر نشود.

- تشخیص خط K-Symbol: ابتدا بررسی می شود که ورودی، یک K-Symbol است یا خیر.

```

1   tmp_k_err <= (tmp_kin & (invalid K-symbol conditions));

```

کدهای کنترلی مخصوصی K-Symbol به صورت رزرو ذخیره شده‌اند و با پترون‌های خاصی تطبیق داده می‌شود. این خط، شرایط ذکر شده را بررسی می‌کند و در صورت لزوم، مقدار پرچم را set می‌کند.

- ثبات میانی t: ثبات t، مقادیر لازم برای رمزگذاری ۱۰ بیتی را محاسبه می‌کند:

- مقادیر [0] تا t[4] از بیت‌های کمارزش data-in بدست می‌آیند.

- مقادیر [5] تا t[7] از بیت‌های پارزش data-in به کمک منطق برای اصلاح Disparity بدست می‌آیند.

- مقادیر [8] تا t[12] با ترکیب کردن بیت‌های مختلف بدست می‌آیند.

- مقادیر t[13] تا t[18] برای کنترل Disparity و هندل کردن K-Symbol ها است.

- نگاشت ۱۰ بیت خروجی: هر بیت tmp-data-out توسط ترکیبی از درایه‌ها ثبات t بدست می‌آید.

```

1   tmp_data_out[9] <= t[12] ^ t[0];
2   tmp_data_out[8] <= t[12] ^ (t[1] | t[2]);
3   tmp_data_out[7] <= t[12] ^ (t[3] | t[4]);
4   tmp_data_out[6] <= t[12] ^ t[5];
5   tmp_data_out[5] <= t[12] ^ (t[6] & t[7]);
6   tmp_data_out[4] <= t[12] ^ (t[8] | t[9] | t[10] | t[11]);
7   tmp_data_out[3] <= t[13] ^ (t[15] & !t[14]);
8   tmp_data_out[2] <= t[13] ^ t[16];
9   tmp_data_out[1] <= t[13] ^ t[17];
10  tmp_data_out[0] <= t[13] ^ (t[18] | t[14]);

```

۲.۴ مژوول Converter

۱.۲.۴ ورودی‌ها

```

1 module converter (
2   input wire clk,           // Clock signal
3   input wire rst,           // Reset signal
4   input wire en,            // Enable signal
5   input wire [65:0] din_66b, // 66-bit input (64-bit data + 2-bit sync)
6   input wire kin,           // Control character

```

شکل ۵: ورودی‌های مژوول Converter

• سیگنال clk به منظور هم‌گام‌سازی

• سیگنال rst به منظور reset کردن مبدل

• سیگنال en به منظور فعال‌سازی مبدل

• ۶۶ بیت داده‌ی ورودی برای تبدیل به رمزگذاری 8B/10B

- ۲ بیت برای هم‌گام‌سازی din-66b[65:64]

• kin: سیگنال kin تعیین می‌کند که آیا ورودی کاراکتر کنترلی است یا خیر (K-symbol)

٢.٢.٤ خروجی‌ها

```

7   output wire [79:0] dout_8b10, // 8 x 10-bit output
8   output wire disp_err,      // Disparity error
9   output wire kin_err,      // Control character error
10  );

```

شکل ٦: خروجی‌های مأذول Converter

• dout-8b10: ٨٠ بیت خروجی که ٨ تا ١٠ بیتی است

• disp-err: سیگنال disp-err به منظور نشان دادن خطأ در محاسبه Disparity است

• kin-err: سیگنال kin-err به منظور نشان دادن اینکه آیا K-Symbol نامعتبر تشخیص داده شده است یا خیر

٣.٢.٤ ثبات‌ها و متغیرهای موقت

```

12 // Extract 2 sync bits and 64-bit data
13 wire [1:0] sync_bits = din_66b[65:64];
14 wire [63:0] data_64b = din_66b[63:0];
15
16 // Intermediate wires for each encoder instance
17 wire [7:0] data_chunk[7:0];
18 wire [9:0] encoded_chunk[7:0];
19 wire disparity[7:0];
20 wire kin_err_chunk[7:0];

```

شکل ٧: ثبات‌های مأذول Converter

• sync-bits: ٢ بیت مربوط به هم‌گام‌سازی را از ورودی din-66b استخراج می‌کند

• data-64b: ٦٤ بیت دیتا را از ورودی din-66b استخراج می‌کند

• data-chunk: هشت تکه‌ی ٨ بیتی که از جداسازی data-64b بدست می‌آید.

• encoded-chunk: هشت تکه‌ی ١٠ بیتی که توسط encoder 8B/10B تولید می‌شود

• disparity: هشت بیت مجزای برای هر بررسی disparity هر تکه است

• kin-err-chunk: هشت بیت مجزا برای بررسی خطاهای کنترلی هر تکه است

```

22 // Assign each 8-bit data chunk
23 genvar i;
24 generate
25   for (i = 0; i < 8; i = i + 1) begin
26     assign data_chunk[i] = data_64b[(i+1)*8-1 : i*8];
27   end
28 endgenerate
29
30 // Instantiate 8 encoder_8b10 modules
31 genvar j;
32 generate
33   for (j = 0; j < 8; j = j + 1) begin : ENCODER_INSTANCES
34     encoder_8b10 encoder_inst (
35       .clk(clk),
36       .rst(rst),
37       .en(en),
38       .kin(kin),
39       .data_in(data_chunk[j]),
40       .data_out(encoded_chunk[j]),
41       .disp(disparity[j]),
42       .kin_err(kin_err_chunk[j])
43     );
44   end
45 endgenerate
46
47 // Combine all 10-bit outputs
48 assign dout_8b10 = {encoded_chunk[7], encoded_chunk[6], encoded_chunk[5], encoded_chunk[4],
49                     encoded_chunk[3], encoded_chunk[2], encoded_chunk[1], encoded_chunk[0]};
50
51 // Combine disparity and control character error signals
52 assign disp_err = {disparity[0], disparity[1], disparity[2], disparity[3],
53                     disparity[4], disparity[5], disparity[6], disparity[7]};
54 assign kin_err = {kin_err_chunk[0], kin_err_chunk[1], kin_err_chunk[2], kin_err_chunk[3],
55                     kin_err_chunk[4], kin_err_chunk[5], kin_err_chunk[6], kin_err_chunk[7]};

```

شکل ۸: الگوریتم مژاول Converter

۴.۲.۴ الگوریتم مبدل

- تقسیم‌بندی ۶۴ بیت به تکه‌های ۸ بیتی

```

1 genvar i;
2 generate
3   for (i = 0; i < 8; i = i + 1) begin
4     assign data_chunk[i] = data_64b[(i+1)*8-1 : i*8];
5   end
6 endgenerate

```

در این بلوک generate، ۸ بیت متوالی از ۶۴ بیت داده‌ی اصلی در یک تکه data-chunk[i] قرار می‌گیرند.

- رمزگذاری هر تکه ۸ بیتی توسط encoder 8B/10B

```

1 genvar j;
2 generate
3   for (j = 0; j < 8; j = j + 1) begin : ENCODER_INSTANCES
4     encoder_8b10 encoder_inst (
5       .clk(clk),
6       .rst(rst),
7       .en(en),
8       .kin(kin),

```

```

9      .data_in(data_chunk[j]),
10     .data_out(encoded_chunk[j]),
11     .disp(disparity[j]),
12     .kin_err(kin_err_chunk[j])
13   );
14
15 end
endgenerate

```

در این بلوک generate، هشت نمونه از رمزگذار 10B/8B تولید می‌شود. هر نمونه، یک تکه ۸ بیتی را به یک تکه ۱۰ بیتی، رمزگذاری می‌کند. همچنین [i] disparity و [i] kin-err-chunk را نیز برای هر تکه، مجزا بررسی می‌شود.

- ترکیب تکه‌های رمزگذاری شده به خروجی نهایی

```
1 assign dout_8b10 = {encoded_chunk[7], encoded_chunk[6], encoded_chunk  
[5], encoded_chunk[4], encoded_chunk[3], encoded_chunk[2],  
encoded_chunk[1], encoded_chunk[0]};
```

هشت تکه‌ی ۱۰ بیتی را پشت سر هم قرار می‌دهیم تا ۸۰ بیت داده‌ی خروجی dout-8b10 تولید شود.

• ترکیب سیگنال‌های خطأ

disp-err =

```
assign disp_err = |{disparity[0], disparity[1], disparity[2],  
    disparity[3], disparity[4], disparity[5], disparity[6],  
    disparity[7]};
```

عملگرد disparity، های هر تکه را با یکدیگر OR منقطعی می‌کند. در صورتی که حتی یک تکه [i] پر ایر یک می‌شود == داشته باشد، disp-err

kin-err =

```
assign kin_err = |{kin_err_chunk[0], kin_err_chunk[1],  
    kin_err_chunk[2], kin_err_chunk[3], kin_err_chunk[4],  
    kin_err_chunk[5], kin_err_chunk[6], kin_err_chunk[7]};
```

مانند قسمت قیا، از عملگر OR منقطع استفاده می‌کنم.

۳۰۴ مادول آزمون

در این مازول ورودی‌های مختلف به عنوان ورودی به مازول converter داده می‌شوند و خروجی مازول بررسی مم شود تا از صحیت عملکرد مازول مبدل اطمینان حاصل کرد.

شکل ۹: ماژول Testbench

با بررسی هر یک از ورودی‌ها و استفاده از جدول رمزگذاری $5B/6B$ و $4B/3B$ می‌توان متوجه شد که ماژول converter به درستی داده‌ها را تبدیل می‌کند.

۴.۴ ماثول تزريق خط

۱.۴.۴ ورودی‌ها

```
module error_injection #(  
    parameter ERROR_RATE = 1, // Error rate (1 in ERROR_RATE chance of error)  
    parameter NUM_BITS = 1   // Number of bits to flip  
)  
(  
    input wire clk,          // Clock signal  
    input wire rst,          // Reset signal  
    input wire en,           // Enable signal  
    input wire [79:0] din,    // 80-bit input (8 x 10-bit encoded data)
```

شکل ۱۰: ورودی‌های ماثول تزريق خط

- clk: سیگنال به منظور همگام‌سازی
- rst: سیگنال به منظور reset کردن ماثول
- en: سیگنال به منظور فعال‌سازی ماثول
- din: ۸۰ بیت ورودی برای تزريق خط

۲.۴.۴ خروجی‌ها

```
    output wire [79:0] dout // 80-bit output with injected errors
```

شکل ۱۱: خروجی‌های ماثول تزريق خط

- dout: ۸۰ بیت داده‌ی خروجی که خط دارد

۳.۴.۴ ثبات‌ها و متغیرهای موقت

```
reg [79:0] data_out;  
reg [31:0] lfsr; // Linear Feedback Shift Register for pseudo-random number generation
```

شکل ۱۲: ثبات‌های ماثول تزريق خط

- data_out: ثبات موقت برای ذخیره داده‌ی نهایی
- lfsr: ثبات برای ذخیره‌سازی مقدار خروجی LFSR

۴.۴.۴ الگوریتم تزريق خط

- تولید عدد تصادفی با LFSR: اگر reset=1 باشد، مقدار اولیه LFSR با یک عدد تصادفی مقداردهی می‌شود. در هر لبه بالا رونده clk، اگر en=1 باشد، مقدار lfsr آپدیت می‌شود.

```
1      lfsr <= {lfsr[30:0], lfsr[31] ^ lfsr[21] ^ lfsr[1] ^ lfsr[0]};
```

```

assign dout = data_out;

// LFSR for pseudo-random number generation
always @(posedge clk or posedge rst) begin
    if (rst) begin
        lfsr <= $random; // Seed value
    end else if (en) begin
        lfsr <= {lfsr[30:0], lfsr[31] ^ lfsr[21] ^ lfsr[1] ^ lfsr[0]};
    end
end

// Error injection logic
integer i;
always @(posedge clk or posedge rst) begin
    if (rst) begin
        data_out <= 80'b0;
    end else if (en) begin
        data_out <= din; // Default to no error
        if (lfsr % ERROR_RATE == 0) begin // Inject error with 1 in ERROR_RATE chance
            for (i = 0; i < NUM_BITS; i = i + 1) begin
                data_out[lfsr % 80] <= ~data_out[lfsr % 80]; // Flip a random bit
            end
        end
    end
end

```

شکل ۱۳: الگوریتم مازول تزریق خطا

این دستور، با فیدبک خاص، مقدار جدیدی تولید می‌کند که به تولید توالی تصادفی کمک می‌کند.

- تزریق خطا: اگر $rst=1$ باشد، مقدار $data_out$ صفر می‌شود. اگر $en=1$ باشد، مقدار din در $lfsr \% ERROR_RATE$ قرار می‌گیرد (یعنی در حالت عادی خروجی بدون خطا خواهد بود). اگر مقدار out باشد، یک خطا ایجاد می‌شود.

۵.۴ تزریق خطای داده‌های ۸ بیتی

برای تزریق خطای داده‌های ۸ بیتی، لازم است مژول converter را کمی تغییر دهیم:

۱.۵.۴ Converter مژول تعریف

```
module converter (
    input wire clk,                      // Clock signal
    input wire rst,                      // Reset signal
    input wire en,                       // Enable signal
    input wire error_injection_enable,   // Error injection enable signal
    input wire [65:0] din_66b,           // 66-bit input (64-bit data + 2-bit sync)
    input wire kin,                      // Control character
    output wire [63:0] chunk_original,    // 8 x 10-bit output
    output wire [63:0] chunk_corrupted,   // 8 x 10-bit corrupted output
    output wire [79:0] dout_original,     // Disparity error original data
    output wire [79:0] dout_corrupted,    // Disparity error corrupted data
    output wire disp_err_original,       // Original control character error
    output wire disp_err_corrupted,      // Corrupted control character error
);

```

شکل ۱۴: ورودی و خروجی‌های مژول Converter

```

// Extract 2 sync bits and 64-bit data
wire [1:0] sync_bits = din_66b[65:64];
wire [63:0] data_64b = din_66b[63:0];

// Intermediate wires for each encoder instance
wire [7:0] data_chunk_original[7:0];
wire [7:0] data_chunk_corrupted[7:0];

wire [9:0] encoded_chunk_original[7:0];
wire [9:0] encoded_chunk_corrupted[7:0];

wire [7:0] disparity_original;
wire [7:0] disparity_corrupted;

wire [7:0] kin_err_chunk_original;
wire [7:0] kin_err_chunk_corrupted;

```

شکل ۱۵: سیم‌های مازول Converter

```

// Assign each 8-bit data chunk
genvar i;
generate
  for (i = 0; i < 8; i = i + 1) begin
    assign data_chunk_original[i] = data_64b[(i+1)*8-1 : i*8];
  end
endgenerate

```

شکل ۱۶: بخش‌بندی داده ۶۴ بیتی

```

// Instantiate 8 encoder_8b10 modules
genvar j;
generate
    for (j = 0; j < 8; j = j + 1) begin : ENCODER_INSTANCES
        error_injection #(
            .ERROR_RATE(1),
            .NUM_BITS(1),
            .WIDTH(8)
        ) error_injection_1 (
            .clk(clk),
            .rst(rst),
            .en(error_injection_enable),
            .din(data_chunk_original[j]),
            .dout(data_chunk_corrupted[j])
        );

        encoder_8b10 encoder_inst_original (
            .clk(clk),
            .rst(rst),
            .en(en),
            .kin(kin),
            .data_in(data_chunk_original[j]),
            .data_out(encoded_chunk_original[j]),
            .disp(disparity_original[j]),
            .kin_err(kin_err_chunk_original[j])
        );
        encoder_8b10 encoder_inst_corrupted (
            .clk(clk),
            .rst(rst),
            .en(en),
            .kin(kin),
            .data_in(data_chunk_corrupted[j]),
            .data_out(encoded_chunk_corrupted[j]),
            .disp(disparity_corrupted[j]),
            .kin_err(kin_err_chunk_corrupted[j])
        );
    end
endgenerate

```

شکل ۱۷: نمونه‌سازی ماثولهای

```

// Combine all 8-bit inputs
assign chunk_original = {data_chunk_original[7], data_chunk_original[6], data_chunk_original[5], data_chunk_original[4],
                        data_chunk_original[3], data_chunk_original[2], data_chunk_original[1], data_chunk_original[0]};

assign chunk_corrupted = {data_chunk_corrupted[7], data_chunk_corrupted[6], data_chunk_corrupted[5], data_chunk_corrupted[4],
                         data_chunk_corrupted[3], data_chunk_corrupted[2], data_chunk_corrupted[1], data_chunk_corrupted[0]};

// Combine all 10-bit outputs
assign dout_original = {encoded_chunk_original[7], encoded_chunk_original[6], encoded_chunk_original[5], encoded_chunk_original[4],
                       encoded_chunk_original[3], encoded_chunk_original[2], encoded_chunk_original[1], encoded_chunk_original[0]};

assign dout_corrupted = {encoded_chunk_corrupted[7], encoded_chunk_corrupted[6], encoded_chunk_corrupted[5], encoded_chunk_corrupted[4],
                        encoded_chunk_corrupted[3], encoded_chunk_corrupted[2], encoded_chunk_corrupted[1], encoded_chunk_corrupted[0]};

// Combine disparity error signals
assign disp_err_original = {disparity_original[0], disparity_original[1], disparity_original[2], disparity_original[3],
                            disparity_original[4], disparity_original[5], disparity_original[6], disparity_original[7]};

assign disp_err_corrupted = {disparity_corrupted[0], disparity_corrupted[1], disparity_corrupted[2], disparity_corrupted[3],
                            disparity_corrupted[4], disparity_corrupted[5], disparity_corrupted[6], disparity_corrupted[7]};

// Combine control character error signals
assign kin_err_original = {kin_err_chunk_original[0], kin_err_chunk_original[1], kin_err_chunk_original[2], kin_err_chunk_original[3],
                           kin_err_chunk_original[4], kin_err_chunk_original[5], kin_err_chunk_original[6], kin_err_chunk_original[7]};

assign kin_err_corrupted = {kin_err_chunk_corrupted[0], kin_err_chunk_corrupted[1], kin_err_chunk_corrupted[2], kin_err_chunk_corrupted[3],
                           kin_err_chunk_corrupted[4], kin_err_chunk_corrupted[5], kin_err_chunk_corrupted[6], kin_err_chunk_corrupted[7]};

```

شکل ۱۸: سیگنال‌های مازول Converter

در این بخش، در ابتدا مقدار داده ۶۴ بیتی اصلی و داده ۶۴ بیتی دچار خطا محاسبه می‌شود.

در ادامه خروجی‌های ۸۰ بیتی به ازای ورودی اصلی و ورودی دارای خطا محاسبه می‌شود.

در نهایت سیگنال‌های kin-err و disp-err به ازای ورودی‌های اصلی و دارای خطا محاسبه می‌شوند.

در نهایت با استفاده از یک مازول آزمون، فرآیند تزریق خطا را بررسی می‌کنیم:


```
input: 000001000100010001000100010001000100010001000100010001000100010001000100010001000100010001000100010001  
chunk original:  
0001000100010001000100010001000100010001000100010001000100010001000100010001000100010001000100010001  
chunk corrupted:  
11111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111111  
dout original:  
1000111011100011101110001110111000111011100011101110001110111000111011100011101110001110111000111011  
dout corrupted:  
1010110001101011000110101100011010110001101011000110101100011010110001101011000110101100011010110001  
chunk sub:  
0001000100010001000100010001000100010001000100010001000100010001000100010001000100010001000100010001  
dout sub:  
1110001001111000100111100010011110001001111000100111100010011110001001111000100111100010011110001010  
disparity error original: 0 | disparity error corrupted: 0  
kin error original: 0 | kin error corrupted: 0
```

```
input: 000000000100100011010001010110011110001001101010111100110111101111  
chunk original:  
00000000100100011010001010110011110001001101010111100110111101111  
chunk corrupted:  
000101010001100100110001001100010011000100010011000100010001  
dout original:  
01110101001100011001101001010111100011001001011101110100101010110001100101110001  
dout corrupted:  
10101010111001101011100011100110001110011000111001100011101110001110011000111011  
chunk sub:  
111011000001010000101000011011001011000100110101001110011011110  
dout sub:  
11001010010010101110000101110001010100110001001000100011100100011001100110110  
disparity error original: 0 | disparity error corrupted: 0  
kin error original: 0 | kin error corrupted: 0
```

شکل ۱۹: خروجی ماذول آزمون

همان طور که مشخص است، با استفاده از این مازول آزمون می‌توان تفاوت دو خروجی در صورت وقوع خطأ را مشاهده کرد. همچنین می‌توان دید که در صورت وقوع خطأ، این پدیده توسط مبدل تشخیص داده نمی‌شود زیرا تغییری در سیگنال‌های *error* و *kin disparity* ایجاد نمی‌شود.

۶.۴ تزریق خطا در خروجی‌های ۸۰ بیتی

نوع دیگری از خطا ممکن است در خروجی مبدل پس از پایان تبدیل رمزگذاری‌ها رخ دهد. به این صورت که ممکن است در خروجی ۸۰ بیتی ماژول converter یک یا چند بیت دچار تغییر شوند. این حالت را به صورت زیر پیاده‌سازی می‌کنیم:

```
// Combine all 8-bit inputs
assign chunk_original = {data_chunk_original[7], data_chunk_original[6], data_chunk_original[5], data_chunk_original[4],
| data_chunk_original[3], data_chunk_original[2], data_chunk_original[1], data_chunk_original[0]};

assign chunk_corrupted = {data_chunk_corrupted[7], data_chunk_corrupted[6], data_chunk_corrupted[5], data_chunk_corrupted[4],
| data_chunk_corrupted[3], data_chunk_corrupted[2], data_chunk_corrupted[1], data_chunk_corrupted[0]};

// Combine all 10-bit outputs
assign dout_original = {encoded_chunk_original[7], encoded_chunk_original[6], encoded_chunk_original[5], encoded_chunk_original[4],
| encoded_chunk_original[3], encoded_chunk_original[2], encoded_chunk_original[1], encoded_chunk_original[0]};

assign dout_corrupted = {encoded_chunk_corrupted[7], encoded_chunk_corrupted[6], encoded_chunk_corrupted[5], encoded_chunk_corrupted[4],
| encoded_chunk_corrupted[3], encoded_chunk_corrupted[2], encoded_chunk_corrupted[1], encoded_chunk_corrupted[0]};

// Combine disparity error signals
assign disp_err_original = {|disparity_original[0], disparity_original[1], disparity_original[2], disparity_original[3],
| disparity_original[4], disparity_original[5], disparity_original[6], disparity_original[7]};

assign disp_err_corrupted = {|disparity_corrupted[0], disparity_corrupted[1], disparity_corrupted[2], disparity_corrupted[3],
| disparity_corrupted[4], disparity_corrupted[5], disparity_corrupted[6], disparity_corrupted[7]};

// Combine control character error signals
assign kin_err_original = {|kin_err_chunk_original[0], kin_err_chunk_original[1], kin_err_chunk_original[2], kin_err_chunk_original[3],
| kin_err_chunk_original[4], kin_err_chunk_original[5], kin_err_chunk_original[6], kin_err_chunk_original[7]};

assign kin_err_corrupted = {|kin_err_chunk_corrupted[0], kin_err_chunk_corrupted[1], kin_err_chunk_corrupted[2], kin_err_chunk_corrupted[3],
| kin_err_chunk_corrupted[4], kin_err_chunk_corrupted[5], kin_err_chunk_corrupted[6], kin_err_chunk_corrupted[7]};
```

شکل ۲۰: ماژول top برای تزریق خطا در خورجی

این حالت را با استفاده از یک ماژول آزمون می‌توان بررسی کرد:

شکل ۲۱: خروجی ماذول آزمون تزریق خطا

همان طور که در خروجی های مارژول آزمون مشخص است، در صورتی که خطای ایجاد شده، running disparity کدهای 8B/10B را در حالت غیر مجاز یعنی ± 2 ببرد، خطای تشخیص داده می شود. اما در صورتی که خطای running disparity را در حالت غیر مجاز ببرد، تشخیص خطای سخت تر می شود اما همچنان ممکن است بتوان خطای تشخیص داد. تشخیص خطای در این مرحله به صورت قطعی انجام نمی شود و نیاز به بررسی خطای لایه های بالاتر مانند استفاده از FEC است.

برای بررسی تشخیص خطا پس از تزریق خطا در خروجی‌های ۸۰ بیتی آزمایشی روی ۲۰ داده رندوم انجام می‌دهیم. خروجی آزمایش به صورت زیر است:

شکل ۲۲: خروجی آزمایش تشخیص خطأ

همان طور که در آزمایش دیده می شود، از ۲۰ خطای تشخیص داده شده است. یعنی در این آزمایش ۷۰ درصد خطاهای تشخیص داده شده اند اما ۳۰ درصد خطاهای قابل تشخیص نبوده اند.

شکل توزیع خطاهای آزمایش به صورت زیر بوده است:

Test 1:
 1001110100 1001110100 1001110100 1001110101 1001110100 1001110100 1001110100 1001110100

Test 2:
 1010110001 1010110001 1010110001 1010110001 1010110001 1010110001 1010110001 01010110001

Test 3:
 1000111011 1000111011 1000111011 1000111011 1000111011 1000111011 1000111011 1000111011

Test 4:
 0111010100 1101011001 1010010101 1110001100 1001011101 1101001010 1011000110 0101110001

Test 5:
 0111101001 0011100110 0101101010 1100110010 0110101100 0010110101 0100111001 0110110100

Test 6:
 0110101100 0010111110 1110100101 1011001110 0110010101 1011001001 1110100001 0011011101

Test 7:
 1011010001 1110101001 1101010010 1010010110 1010100110 1100101011 0100110110 0101011010

Test 8:
 1001011101 0100111001 0110100110 0100111011 1110000101 0011010110 1101100110 0101110010

Test 9:
 001110111 1001110100 0101101100 1110010001 1011010001 0101010110 0111000101 1010010110

Test 10:
 0110101101 1101001010 1100110101 1011001001 0100111010 1110001011 0110011001 1010011100

Test 11:
 1001100110 1101101001 1011011001 1001110010 0110110100 1101010011 0111011001 1001111001

Test 12:
 1101000110 1001111001 0111101101 0110101101 1001011101 1100011101 1100111010 1100101011

Test 13:
 1010101001 1010110010 1011100110 1101001100 0101011110 0110011010 0101111001 1010100110

Test 14:
 0111101011 1010110001 1001011110 1011100100 1110001110 1010010110 0100111100 0101110110

Test 15:
 1010011110 1100101100 0101011011 0101000110 0111100010 1000111001 0011010101 0011101001

Test 16:
 1001111001 1101010110 1100101010 0111010101 0011011110 1100000101 0010111001 1100110110

Test 17:
 1010101100 1010010110 1011001011 1101001110 1101100101 1011010100 1010011100 01100001010

Test 18:
 0111100110 1010101100 1011010100 1011101010 1010101011 0101110100 1011100110 0101011001

Test 19:
 1011010101 0100110111 0111010101 1010011101 1110001001 0100110111 1010100101 0101110001

Test 20:
 1010111010 1100011001 0100111001 0111100011 0101011011 0101011011 1101000101 1010101011

شكل ۲۳: توزیع خطاهای

در این آزمایش برای تشخیص خطاهای امکان ظاهر شدن الگوی داده‌ها در رمزگذاری 8B/10B به عنوان معیار در نظر گرفته شده است. اگر امکان دیده شدن الگویی در انکویدینگ 8B/10B وجود نداشته باشد، خطا تشخیص داده می‌شود.