دانشکده‌ی مهندسی کامپیوتر

اشکان تاریوردی ۴۰۱۱۰۵۷۵۳ - روژین تقی‌زادگان ۴۰۱۱۰۵۷۷۵

مدارهای واسط

زمستان ۱۴۰۳

گزارش پروژه

۱ انکودینگ 64B/66B

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

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

۲ تبدیل انکودینگ 64B/66B به 8B/10B

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

۳ انکودینگ 8B/10B

۱.۳ پیاده‌سازی

در این انکودینگ، ۸ بیت داده در قالب یک symbol یا character ۱۰ بیتی فرستاده می‌شود. ۵ بیت کم‌ارزش داده با استفاده از انکودینگ 5B/6B به یک گروه ۶ بیتی تبدیل می‌شود. همچنین ۳ بیت پرارزش داده با استفاده از انکودینگ 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.

symbolهای داده به صورت D.x.y نشان داده می‌شوند که x بین صفر تا ۳۱ و y بین صفر تا ۷ است.

---

[۱] Bit Significant Most

[۲] sync header

[۳] payload

استانداردهایی که از انکودینگ 8B/10B استفاده می‌کنند همچنین ۱۲ symbol خاص یا symbol کنترلی تعریف می‌کنند که به جای symbol داده فرستاده می‌شوند. آن‌ها غالبا برای نشان دادن آغاز frame، پایان frame، بیکار بودن لینک و موارد مشابه استفاده می‌شوند. این symbolها به صورت K.x.y نشان داده می‌شوند.

از آن جا که انکودینگ 8B/10B از symbolهای ۱۰ بیتی برای انکودینگ داده‌های ۸ بیتی استفاده می‌کند، برخی از ۱۰۲۴ symbol موجود استفاده نمی‌شوند تا تضمین شود که در این انکودینگ اختلاف تعداد بیت‌های صفر و یک بیشتر از دو نمی‌شود. همچنین برخی از ۲۵۶ داده ۸ بیتی می‌توانند به دو صورت انکودینگ شوند. استفاده از این انکودینگ‌های جایگزین باعث می‌شود که این انکودینگ بتواند DC-balance را به صورت طولانی‌مدت حفظ کند.

۲.۳ Running disparity

انکودینگ 8B/10B بدون نیاز به DC-component عمل می‌کند، به این معنی که در یک بازه طولانی‌مدت تعداد بیت‌های صفر و یک فرستاده‌شده 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+ یا ۱- است. این تقاوت به عنوان running disparity یا RD شناخته می‌شود.

این انکودینگ تنها به دو حالت running disparity ۱+ و ۱- نیاز دارد. کدگذاری از حالت ۱- آغاز می‌شود.

برای هر کد 5B/6B و 3B/4B که تعداد بیت‌های صفر و یک نابرابر دارند، دو الگوی بیتی قابل استفاده وجود دارد که در یک الگو تعداد بیت‌های ۱ بیشتر است و در الگوی دیگر تعداد بیت‌های صفر بیشتر است. با توجه به running disparity کنونی سیگنال، رمزگذار تصمیم می‌گیرد که کدام گروه ۶ بیتی یا ۴ بیتی را برای ارسال داده انتخاب کند. واضح است در صورتی که گروه‌های ۶ بیتی و ۴ بیتی تعداد بیت‌های صفر و یک برابر داشته باشند، تنها از یک الگو استفاده می‌شود.

۴ پیاده‌سازی به کمک زبان توصیف سخت‌افزار

۱.۴ ماژول رمزگذاری 8B/10B

۱.۱.۴ ورودی‌ها

```
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
```

- clk: سیگنال clk به منظور هم‌گام‌سازی
- rst: سیگنال rst به منظور reset کردن رمزگذار
- en: سیگنال en به منظور فعال‌سازی رمزگذار
- kin: سیگنال kin تعیین می‌کند که آیا ورودی کاراکتر کنترلی است یا خیر (K-symbol)
- data-in: ۸ بیت داده‌ی ورودی برای انکودینگ

۲.۱.۴ خروجی‌ها

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

- data-out: ۱۰ بیت داده‌ی خروجی که انکودینگ شده است
- disp: سیگنال disp به منظور نشان دادن Disparity یا DC-Balance
- kin-err: 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

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

- tmp-disp: این ثبات مقدار Disparity انکودینگ فعلی را ذخیره می‌کند
- tmp-k-err: این ثبات بررسی می‌کند که به هنگام انکودینگ K-Symbol، خطا رخ داده است یا خیر
- t: این ثبات موقت ۱۹ بیتی است و مقادیر میانی به هنگام محاسبه انکودینگ را ذخیره می‌کند
- tmp-data-out: این ثبات ۱۰ بیت نهایی انکودینگ شده را ذخیره می‌کند
- tmp-data-in: این سیم داده‌ی ورودی را ذخیره می‌کند
- tmp-kin: این سیم مقدار kin ورودی را ذخیره می‌کند

۴.۱.۴ الگوریتم انکودینگ

```
always @(posedge clk) begin
    if (rst) begin
        tmp_disp <= 1'b0;
        tmp_k_err <= 1'b0;
        tmp_data_out <= 10'b0;
    end else begin
        if (en == 1'b1) begin
            tmp_disp <= ((tmp_data_in[5]&tmp_data_in[6]&tmp_data_in[7])|(!tmp_data_in[5]&!t
            tmp_k_err <= (tmp_kin&(tmp_data_in[0]|tmp_data_in[1]|!tmp_data_in[2]|!tmp_data_
            tmp_data_out[9] <= t[12]^t[0];
            tmp_data_out[8] <= t[12]^(t[1]|t[2]);
            tmp_data_out[7] <= t[12]^(t[3]|t[4]);
            tmp_data_out[6] <= t[12]^t[5];
            tmp_data_out[5] <= t[12]^(t[6]&t[7]);
            tmp_data_out[4] <= t[12]^(t[8]|t[9]|t[10]|t[11]);
            tmp_data_out[3] <= t[13]^(t[15]&!t[14]);
            tmp_data_out[2] <= t[13]^t[16];
            tmp_data_out[1] <= t[13]^t[17];
            tmp_data_out[0] <= t[13]^(t[18]|t[14]);
        end
    end
end
```

```
always @(posedge clk) begin
    if(rst) begin
        t <= 0;
    end else begin
        if (en == 1'b1) begin
            t[0] <= tmp_data_in[0];
            t[1] <= tmp_data_in[1]&!(tmp_data_in[0]&tmp_data_in[1]&tmp_data_in[2]&tmp_data_in[3]);
            t[2] <= (!tmp_data_in[0]&!tmp_data_in[1]&!tmp_data_in[2]&!tmp_data_in[3]);
            t[3] <= (!tmp_data_in[0]&!tmp_data_in[1]&!tmp_data_in[2]&!tmp_data_in[3])|tmp_data_in[2];
            t[4] <= tmp_data_in[4]&tmp_data_in[3]&!tmp_data_in[2]&!tmp_data_in[1]&!tmp_data_in[0];
            t[5] <= tmp_data_in[3]&!(tmp_data_in[0]&tmp_data_in[1]&tmp_data_in[2]);
            t[6] <= tmp_data_in[4]|((!((tmp_data_in[0]&tmp_data_in[1]))|(!tmp_data_in[0]&!tmp_data_in[1
            t[7] <= !(tmp_data_in[4]&tmp_data_in[3]&!tmp_data_in[2]&!tmp_data_in[1]&!tmp_data_in[0]);
            t[8] <= (((tmp_data_in[0]&tmp_data_in[1]&!tmp_data_in[2]&!tmp_data_in[3])|(tmp_data_in[2]&t
            t[9] <= tmp_data_in[4]&!tmp_data_in[3]&!tmp_data_in[2]&!(tmp_data_in[0]&tmp_data_in[1]);
            t[10] <= tmp_kin&tmp_data_in[4]&tmp_data_in[3]&tmp_data_in[2]&!tmp_data_in[1]&!tmp_data_in[
            t[11] <= tmp_data_in[4]&!tmp_data_in[3]&tmp_data_in[2]&!tmp_data_in[1]&!tmp_data_in[0];
            t[12] <= (((tmp_data_in[4]&tmp_data_in[3]&!tmp_data_in[2]&!tmp_data_in[1]&!tmp_data_in[0])|
            t[13] <= ((!tmp_data_in[5]&!tmp_data_in[6])|(tmp_kin&((tmp_data_in[5]&!tmp_data_in[6])|(!t
            t[14] <= tmp_data_in[5]&tmp_data_in[6]&tmp_data_in[7]&(tmp_kin|(tmp_disp?(!tmp_data_in[4]&t
            t[15] <= tmp_data_in[5];
            t[16] <= tmp_data_in[6]|(!tmp_data_in[5]&!tmp_data_in[6]&!tmp_data_in[7]);
            t[17] <= tmp_data_in[7];
            t[18] <= !tmp_data_in[7]&(tmp_data_in[6]^tmp_data_in[5]);
        end
    end
end
```

- Disparity: در هر چرخه، مقدار Disparity به‌روزرسانی می‌شود.

```
    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 می‌کند که تفاضل بین تعداد یک‌ها و صفرها در داده‌ی انکودینگ شده، از یک بیش‌تر نشود.

- تشخیص خطا K-Symbol: ابتدا بررسی می‌شود که ورودی، یک K-Symbol است یا خیر.

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

کدهای کنترلی مخصوصی K-Symbol به صورت رزرو ذخیره شده‌اند و با پترن‌های 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 انکودینگ ۱۰ بیتی را محاسبه می‌کند:
  - مقادیر t[0] تا t[4] از بیت‌های کم‌ارزش data-in بدست می‌آیند.
  - مقادیر t[5] تا t[7] از بیت‌های پرارزش data-in به کمک منطق برای اصلاح Disparity بدست می‌آیند.
  - مقادیر t[8] تا t[12] با ترکیب کردن بیت‌های مختلف بدست می‌آیند.
  - مقادیر t[13] تا t[18] برای کنترل Disparity و هندل کردن K-Symbolها است.
- نگاشت ۱۰ بیت خروجی: هر بیت tmp-data-out توسط ترکیبی از درایه‌ها ثبات t بدست می‌آید.

```
    tmp_data_out[9] <= t[12] ^ t[0];
    tmp_data_out[8] <= t[12] ^ (t[1] | t[2]);
    tmp_data_out[7] <= t[12] ^ (t[3] | t[4]);
    tmp_data_out[6] <= t[12] ^ t[5];
    tmp_data_out[5] <= t[12] ^ (t[6] & t[7]);
    tmp_data_out[4] <= t[12] ^ (t[8]|t[9] | t[10] | t[11]);
```

```
    tmp_data_out[3] <= t[13] ^ (t[15] & !t[14]);
    tmp_data_out[2] <= t[13] ^ t[16];
    tmp_data_out[1] <= t[13] ^ t[17];
    tmp_data_out[0] <= t[13] ^ (t[18] | t[14]);
```

۲.۴ ماژول Converter

۱.۲.۴ ورودی‌ها

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

- clk: سیگنال clk به منظور هم‌گام‌سازی
- rst: سیگنال rst به منظور reset کردن مبدل
- en: سیگنال en به منظور فعال‌سازی مبدل
- din-66b: ۶۶ بیت داده‌ی ورودی برای تبدیل به انکودینگ 8B/10B
  - ۲ بیت برای هم‌گام‌سازی din-66b[65:64]
  - ۶۴ بیت داده din-66b[63:0]
- kin: سیگنال kin تعیین می‌کند که آیا ورودی کاراکتر کنترلی است یا خیر (K-symbol)

۲.۲.۴ خروجی‌ها

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

- dout-8b10: ۸۰ بیت خروجی که ۸ تا ۱۰ بیتی است
- disp-err: سیگنال disp-err به منظور نشان دادن خطا در محاسبه Disparity است
- kin-err: سیگنال kin-err به منظور نشان دادن اینکه آیا K-Symbol نامعتبر تشخیص داده شده است یا خیر

۳.۲.۴ ثبات‌ها و متغیرهای موقت

```
// 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[7:0];
wire [9:0] encoded_chunk[7:0];
wire disparity[7:0];
wire kin_err_chunk[7:0];
```

- sync-bits: ۲ بیت مربوط به هم‌گام‌سازی را از ورودی din-66b استخراج می‌کند
- data-64b: ۶۴ بیت دیتا را از ورودی din-66b استخراج می‌کند
- data-chunk: هشت تکه‌ی ۸ بیتی که از جداسازی data-64b بدست می‌آید.
- encoded-chunk: هشت تکه‌ی ۱۰ 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

۴.۲.۴ الگوریتم مبدل

```
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]};
```

- تقسیم‌بندی ۶۴ بیت به تکه‌های ۸ بیتی

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

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

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

```
genvar j;
generate
    for (j = 0; j < 8; j = j + 1) begin : ENCODER_INSTANCES
        encoder_8b10 encoder_inst (
            .clk(clk),
            .rst(rst),
            .en(en),
            .kin(kin),
            .data_in(data_chunk[j]),
            .data_out(encoded_chunk[j]),
            .disp(disparity[j]),
            .kin_err(kin_err_chunk[j])
        );
    end
endgenerate
```

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

- ترکیب تکه‌های انکودینگ شده به خروجی نهایی

```
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 خطا
  - disp-err

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

عملگر |، disparityهای هر تکه را با یکدیگر OR منطقی می‌کند. در صورتی که حتی یک تکه disparity[i] == 1 داشته باشد، disp-err برابر یک می‌شود

  - 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 قبل، از عملگر | یا همان OR منطقی استفاده می‌کنیم.

۳.۴ ماژول آزمون

در این ماژول ورودی‌های مختلف به عنوان ورودی به ماژول converter داده می‌شوند و خروجی ماژول بررسی می‌شود تا از بتوان از صحت عملکرد ماژول مبدل اطمینان حاصل کرد.

```
Test 1 – All zeros: dout_8b10 = 10011101001001110100100111010010011101001001110100100111010010011101001001110100100111010010011101001001110100, disp_err = 0, kin_err = 0
Test 2 – All ones: dout_8b10 = 10101100011010110001101011000110101100011010110001101011000110101100011010110001, disp_err = 0, kin_err = 0
Test 3 – Alternate bits: dout_8b10 = 01010110100101011010010101101001010110100101011010010101101001010110100101011010, disp_err = 0, kin_err = 0
Test 4 – Single control char: dout_8b10 = 10101110001001110100100111010010011101001001110100100111010010011101001001110100, disp_err = 0, kin_err = 1
Test 5 – Mixed data/control: dout_8b10 = 10101001101010100101101010011010101001010101101001010111010010101101001010111010, disp_err = 0, kin_err = 1
Test 6 – MSB sync bits: dout_8b10 = 10011101001001110100100111010010011101001001110100100111010010011101001001110100, disp_err = 0, kin_err = 0
Test 7 – LSB sync bits: dout_8b10 = 10011101001001110100100111010010011101001001110100100111010010011101001100011011, disp_err = 0, kin_err = 0
Test 8 – Random data with control: dout_8b10 = 01110101001100011001101001010111100000111001010010110100101010110001100101111000, disp_err = 0, kin_err = 1
Test 9 – Random data no control: dout_8b10 = 01011101001011001110110100011010010110101110001101101001110011000101010111011001, disp_err = 0, kin_err = 0
```

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

۴.۴ ماژول تزریق خطا

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)
```

- clk: سیگنال clk به منظور هم‌گام‌سازی
- rst: سیگنال rst به منظور reset کردن ماژول
- en: سیگنال en به منظور فعال‌سازی ماژول
- din: ۸۰ بیت ورودی برای تزریق خطا

۲.۴.۴ خروجی‌ها

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

- dout: ۸۰ بیت داده‌ی خروجی که خطا 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
```

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

۴.۴.۴ الگوریتم تزریق خطا

```
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
```

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

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

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 data-out قرار می‌گیرد (یعنی در حالت عادی خروجی بدون خطا خواهد بود). اگر مقدار lfsr %ERROR-RATE == 0 باشد، یک خطا ایجاد می‌شود.

۵.۴ تزریق خطا در داده‌های ۸ بیتی

برای تزریق خطا در داده‌های ۸ بیتی، لازم است ماژول converter را کمی تغییر دهیم:

- تعریف ماژول converter

```
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,
    output wire [63:0] chunk_corrupted,
    output wire [79:0] dout_original,   // 8 x 10-bit output
    output wire [79:0] dout_corrupted,  // 8 x 10-bit corrupted output
    output wire disp_err_original,      // Disparity error original data
    output wire disp_err_corrupted,     // Disparity error cottupted data
    output wire kin_err_original,       // Original control character error
    output wire kin_err_corrupted       // Corrupted control character error
);
```

- تعریف داده‌های میانی

```
// 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;
```

- تقسیم‌بندی داده ۶۴ بیتی به ۸ بخش ۸ بیتی

```
// 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
```

- تعریف ماژول‌های مورد نیاز

```
// 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
```

- محاسبه سیگنال‌های خروجی

```
// 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]};
```

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

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

```
input: 0000000000000000000000000000000000000000000000000000000000000000
chunk original:
0000000000000000000000000000000000000000000000000000000000000000
chunk corrupted:
0001000000000010000000100000100000100000001000000010000000000100
dout original:
1001110100100111010010011101001001110100100111010010011101001001110100
dout corrupted:
0110110100101101010010110101001110010100100111100110011110011001111001110101010100
chunk sub:
1110111111111110111111101111110111111011111110111111110111111111111100
dout sub:
0010111111111001111111100111111011011111111111101011111110101111111010110010000 0
disparity error original: 0 | disparity error corrupted: 0
kin error original: 0 | kin error corrupted: 0
--------------------------------------------------------------------------------
input: 0011111111111111111111111111111111111111111111111111111111111111
chunk original:
1111111111111111111111111111111111111111111111111111111111111111
chunk corrupted:
0100000000000010000100000010000000100000001000000000010000000000010
dout original:
1010110001101011000110101100011010110001101011000110101100011010110001
dout corrupted:
1001110101101101010001101101001001111001100111100110011101011001111001101101010 0
chunk sub:
1011111111111101111011111101111111011111101111111110111111111111101
dout sub:
0000111011111101110100111111010000111000000011100000001111000000110111111111011101
disparity error original: 0 | disparity error corrupted: 0
kin error original: 0 | kin error corrupted: 0
--------------------------------------------------------------------------------
```

```
input: 0000010001000100010001000100010001000100010001000100010001000100010001
chunk original:
0001000100010001000100010001000100010001000100010001000100010001000100010001
chunk corrupted:
1111111111111111111111111111111111111111111111111111111111111111111111111111
dout original:
1000111011100011101110001110111000111011100011101110001110111000111011100011
dout corrupted:
1010110001101011000110101100011010110001101011000110101100011010110000110101
chunk sub:
0001000100010001000100010001000100010001000100010001000100010001000100010010
dout sub:
1110001001111000100111100010011110001001111000100111100010011110001001111000
disparity error original: 0 | disparity error corrupted: 0
kin error original: 0 | kin error corrupted: 0
--------------------------------------------------------------------------------
input: 0000000001001000110100010101100111100010011010101111001101111011111
chunk original:
0000000100100011010001010110011111000100110101011110011011110111111
chunk corrupted:
0001010100011001001100010011000100110001000100010011000100010001
dout original:
0111010100110001100110100101011110001100100101110111010010101011000110010111000
dout corrupted:
1010101011100110101110001110011000111001100011100110001110111000111001100011101
chunk sub:
1110110000001010000101000011011001011000100110101001110011011110
dout sub:
1100101001001010111000010111000101010011000010010001000011110010001100110011010
disparity error original: 0 | disparity error corrupted: 0
kin error original: 0 | kin error corrupted: 0
--------------------------------------------------------------------------------
input: 0000000001001000110100010101100111100010011010101111001101111011111
chunk original:
0000000100100011010001010110011111000100110101011110011011110111111
chunk corrupted:
0000000100100011010001010110011111100100110101011111011011110111111
dout original:
0111010100110001100110100101011110001100100101110111010010101011000110010111000
dout corrupted:
0111010100110001100110100101011110001100100101011011010010101011001110010111000
chunk sub:
1111111111111111111111111111111111011111111111111111100000000000000
dout sub:
0000000000000000000000000000000000000000000000011011111111111111111000000000000
disparity error original: 0 | disparity error corrupted: 0
kin error original: 0 | kin error corrupted: 0
--------------------------------------------------------------------------------
```

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

۶.۴ تزریق خطا در خروجی‌های ۸۰ بیتی

نوع دیگری از خطا ممکن است در خروجی مبدل پس از پایان تبدیل انکودینگ‌ها 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:

```
module top (
    input wire clk,                // Clock signal
    input wire rst,                // Reset signal
    input wire en,                 // Enable signal
    input wire [65:0] din_66b,     // 66-bit input (64-bit data + 2-bit sync)
    input wire kin,                // Control character
    input wire error_injection_enable,
    output wire [79:0] dout_8b10, // 8 x 10-bit output
    output wire disp_err,          // Disparity error
    output wire kin_err,           // Control character error
    output wire [79:0] corrupted_data
);

    // Instantiate the converter module
    converter converter_inst (
        .clk(clk),
        .rst(rst),
        .en(en),
        .din_66b(din_66b),
        .kin(kin),
        .dout_8b10(dout_8b10),
        .disp_err(disp_err),
        .kin_err(kin_err)
    );

    // Instantiate the error injection module
    error_injection #(
        .ERROR_RATE(1),  // 1 in 100 chance of error
        .NUM_BITS(1)       // Flip 1 bit
    ) error_injection_inst (
        .clk(clk),
        .rst(rst),
        .en(error_injection_enable),
        .din(dout_8b10),
        .dout(corrupted_data)
    );

endmodule
```

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

```
input: 0000000000000000000000000000000000000000000000000000000000000000
original output:
9d2749d2749d2749d274
corrupted output:
9d2749d2759d2749d274
sub:
1111111111111111111111111111111111111111000000000000000000000000000000000000
--------------------------------------------------------------------------------
input: 0011111111111111111111111111111111111111111111111111111111111111
original output:
ac6b1ac6b1ac6b1ac6b1
corrupted output:
ac6b1ac6b1ac6b1ac4b1
sub:
0000000000000000000000000000000000000000000000000000000001000000000
--------------------------------------------------------------------------------
input: 0000010001000100010001000100010001000100010001000100010001000100010001
original output:
8ee3b8ee3b8ee3b8ee3b
corrupted output:
8ee3b8ee3b8ee3b8ea3b
sub:
0000000000000000000000000000000000000000000000000000000010000000000
--------------------------------------------------------------------------------
input: 0000000001001000110100010101100111100010011010101111001101111011111
original output:
75319a578c9774ab1971
corrupted output:
75359a578c9774ab1971
sub:
1111111111111100000000000000000000000000000000000000000000000000000000000000
--------------------------------------------------------------------------------
input: 0011111110110111001011101010011000011101100101010000110010000010000
original output:
784e65ab326b0b54e5b4
corrupted output:
7a4e65ab326b0b54e5b4
sub:
1111111000000000000000000000000000000000000000000000000000000000000000000000
--------------------------------------------------------------------------------
```

همان‌طور که در خروجی‌های ماژول آزمون مشخص است، در صورتی که خطای ایجاد شده، running disparity کدهای 8B/10B را در حالت غیر مجاز یعنی ± 2 ببرد، خطا تشخیص داده می‌شود. اما در صورتی که خطا running disparity را در حالت غیرمجاز ببرد، تشخیص خطا سخت‌تر می‌شود اما همچنان ممکن است بتوان خطا را تشخیص داد. تشخیص خطا در این مرحله به صورت قطعی انجام نمی‌شود و نیاز به بررسی خطا در لایه‌های بالاتر مانند استفاده از CRC و 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:

```
Test 1: input: 00000000000000000000000000000000000000000000000000000000000000000000
original output:
10011101001001110100100111010010011101001001110100100111010010011101001001110100
corrupted output:
10011101001001110100100111010010011101011001110100100111010010011101001001110100
sub:
11111111111111111111111111111111111111110000000000000000000000000000000000000000
error: 1
--------------------------------------------------------------------------
Test 2: input: 00111111111111111111111111111111111111111111111111111111111111111111
original output:
10101100011010110001101011000110101100011010110001101011000110101100011010110001
corrupted output:
10101100011010110001101011000110101100011010110001101011000110101100010010110001
sub:
00000000000000000000000000000000000000000000000000000000000000000001000000000000
error: 1
--------------------------------------------------------------------------
Test 3: input: 00000100010001000100010001000100010001000100010001000100010001000100
original output:
10001110111000111011100011101110001110111000111011100011101110001110111000111011
corrupted output:
10001110111000111011100011101110001110111000111011100011101110001110101000111011
sub:
00000000000000000000000000000000000000000000000000000000000000000010000000000000
error: 0
--------------------------------------------------------------------------
Test 4: input: 00000000010010001101000101011001111000100110101011110011011110111
original output:
01110101001100011001101001010111100011001001011101110100101010110001100101110001
corrupted output:
01110101001101011001101001010111100011001001011101110100101010110001100101110001
sub:
11111111111111000000000000000000000000000000000000000000000000000000000000000000
error: 0
--------------------------------------------------------------------------
Test 5: input: 00111111101101110010111010100110000111011001010100001100100001000
original output:
01111000010011100110010110101011001100100110101100001011010101001110010110110100
corrupted output:
01111010010011100110010110101011001100100110101100001011010101001110010110110100
sub:
11111110000000000000000000000000000000000000000000000000000000000000000000000000
error: 0
--------------------------------------------------------------------------
```

```
Test 6: input: 0001110110110101000101011111101101010001100010110111110111110001100
original output:
0110101100001011011011101001011011001110011001010110110010011111010000100110111101
corrupted output:
0110101100001011111011101001011011001110011001010110110010011111010000100110111101
sub:
1111111111111111100000000000000000000000000000000000000000000000000000000000000000
error: 1
--------------------------------------------------------------------------
Test 7: input: 0011100010111101111000010011000101110101010001001111010010101010101
original output:
1011010001111010000111010100101010010110101010011011001010110100110110010101101010
corrupted output:
1011010001111010100111010100101010010110101010011011001010110100110110010101101010
sub:
1111111111111111100000000000000000000000000000000000000000000000000000000000000000
error: 0
--------------------------------------------------------------------------
Test 8: input: 0010001001001100101101011000010010010001111110110011011011100011111
original output:
1001011101010011100101101001100100111011111000010100110111101101100110010111001010
corrupted output:
1001011101010011100101101001100100111011111000010100110101101101100110010111001010
sub:
0000000000000000000000000000000000000000000000000000000010000000000000000000000000
error: 1
--------------------------------------------------------------------------
Test 9: input: 0011110100000000000111101011101000111000101100101001001110110001010
original output:
0010110111100111010001011011001110010001101101000101010101100111000101101001011010
corrupted output:
0011110111100111010001011011001110010001101101000101010101100111000101101001011010
sub:
1111000000000000000000000000000000000000000000000000000000000000000000000000000000
error: 1
--------------------------------------------------------------------------
Test 10: input: 0010010110101010110101100000101101101100101010011100100110011001010
original output:
0110101101111010010101100110101101100100101001110101110000101001100110011010011100
corrupted output:
0110101101111010010101100110101101100100101001110101110000101101100110011010011100
sub:
1111111111111111111111111111111111111111111111111111111111111111000000000000000000
error: 1
--------------------------------------------------------------------------
```

```
Test 11: input: 0011000000001110110010001010000000001000001100100001000010010000
original output:
1001110110110110100110110110011001110010011011010011010100110111011001100111001
corrupted output:
1001100110110110100110110110011001110010011011010011010100110111011001100111001
sub:
0000010000000000000000000000000000000000000000000000000000000000000000000000000
error: 0
--------------------------------------------------------------------------
Test 12: input: 0011001011001000000011111010010110100010011000001110111000000010011
original output:
1101000110100111100101111010010110101101100101110111000111011100111010110010101 1
corrupted output:
1101000110100111100101111011010110101101100101110111000111011100111010110010101 1
sub:
1111111111111111111111111111000000000000000000000000000000000000000000000000000
error: 1
--------------------------------------------------------------------------
Test 13: input: 0000110101100111111101110101101011111010101010011000101010110101 01
original output:
1010101001101011001010111001101101001100010101111001100110100101011001101010011 0
corrupted output:
1010101001101011001010111001101101001100010101111001100110100101111001101010011 0
sub:
1111111111111111111111111111111111111111111111111111111111111111111100000000000000
error: 1
--------------------------------------------------------------------------
Test 14: input: 0000001110111111111110100100011101111001111100010101110010110011111
original output:
0111001011101011000110010111101011100100111000111010100101100100111100010111011 0
corrupted output:
0111101011101011000110010111101011100100111000111010100101100100111100010111011 0
sub:
1111100000000000000000000000000000000000000000000000000000000000000000000000000
error: 1
--------------------------------------------------------------------------
Test 15: input: 0011100101011100110000101011001010100111100011000101001100001111 00
original output:
1010011110110010110001010110110101010110011111000101000111001001101010100111010 01
corrupted output:
1010011110110010110001010110110101000110011110001010001110010011010101001110100 1
sub:
0000000000000000000000000000000000010000000000000000000000000000000000000000000
error: 1
--------------------------------------------------------------------------
```

```
Test 16: input: 0000100000110001001011001101000001111011000100101100110100110110000
original output:
1001111001110101011011001010100111010101001101111011010001010010111001110011011O
corrupted output:
1001111001110101011011001010100111010101001101111011000001010010111001110011011O
sub:
0000000000000000000000000000000000000000000000000000100000000000000000000000000
error: 1
--------------------------------------------------------------------------
Test 17: input: 0001110101110001010000110111101011010110110000001001100101101101l0
original output:
1010101100101001011010110010111101001110110110010110110101001010011100011010101O
corrupted output:
1010101100101001011010110010111101001110110110010110110101001010011100011000101O
sub:
0000000000000000000000000000000000000000000000000000000000000000000000000100000
error: 1
--------------------------------------------------------------------------
Test 18: input: 0011011110011101010000001010111100000101010000111111011101001010l0
original output:
0111100110101010110010110101000011101010101010101110101110100101110011001010110O1
corrupted output:
0111100110101010110010110101000101110101010101011010111010010111001100101011001
sub:
1111111111111111111111111111110000000000000000000000000000000000000000000000000
error: 1
--------------------------------------------------------------------------
Test 19: input: 0001000010111100100100000110000101001001111111001001010101010011111
original output:
1011010101010011011101110101011010011101111000100101001101111010100101010111010l
corrupted output:
1011010101010011011101110101011010011101111000100101001101111010100101010111000l
sub:
0000000000000000000000000000000000000000000000000000000000000000000000000000100
error: 0
--------------------------------------------------------------------------
Test 20: input: 0010111111001000110011001001111111000001010101010100100101100010101
original output:
1010111010110001100101001110010111100011010101101101010110101101000101101010101l
corrupted output:
1010111010110001100101001110010111100011010101101101010111101000101101010101l
sub:
1111111111111111111111111111111111111111111111111111111111111111111100000000000000000000
error: 1
--------------------------------------------------------------------------
```

همان‌طور که در آزمایش دیده می‌شود، از ۲۰ خطا ۱۴ خطا تشخیص داده شده است. یعنی در این آزمایش ۷۰ درصد خطاها تشخیص داده شده‌اند اما ۳۰ درصد خطاها قابل تشخیص نبوده‌اند.

شکل توزیع خطاها در این 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 0010110001

Test 3:
1000111011 1000111011 1000111011 1000111011 1000111011 1000111011 1000111010 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:
0011110111 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 0110001010

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
```

در این آزمایش برای تشخیص خطاها، امکان ظاهر شدن الگوی داده‌ها در انکودینگ 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.