[[CRC] Bài 5 - Các thông số ảnh hưởng đến kết quả tính CRC](https://nguyenquanicd.blogspot.com/2019/09/crc-tap-hop-cac-van-e-ve-viec-tinh-toan.html)

Tác giả [Nguyễn Quân](https://www.blogger.com/profile/11739939602738380602) at 08:18  [background](https://nguyenquanicd.blogspot.com/search/label/background?&max-results=5), [CRC](https://nguyenquanicd.blogspot.com/search/label/CRC?&max-results=5), [Kiến Thức Cơ Bản](https://nguyenquanicd.blogspot.com/search/label/Ki%E1%BA%BFn%20Th%E1%BB%A9c%20C%C6%A1%20B%E1%BA%A3n?&max-results=5)  [1 bình luận](https://nguyenquanicd.blogspot.com/2019/09/crc-tap-hop-cac-van-e-ve-viec-tinh-toan.html#comment-form)

*Bài viết này trình bày khái quát lại về các vấn đề liên quan đến việc tính CRC (Cyclic Redundancy Code) bao gồm các thông số tính CRC, giá trị khởi tạo và các cách biểu diễn đa thức sinh. Bài viết sẽ giải thích chi tiết sự ảnh hưởng của các thông số này đến việc tính CRC. Đồng thời, thực hiện 1 bộ tính CRC hỗ trợ đầy đủ việc thiết lập các thông số tính CRC bằng Verilog HDL.*

**1) Đặt vấn đề**

Trong [các bài viết trước](http://nguyenquanicd.blogspot.com/search/label/CRC?&max-results=5" \t "_blank), tác giả có trình bày về lý thuyết cơ bản và các cách thiết kế bộ tính CRC:

[Bài 1 - Lý thuyết CRC và mạch tính CRC nối tiếp](http://nguyenquanicd.blogspot.com/2017/10/basic-knowledgecrc-bai-1-ly-thuyet-ve.html)

[Bài 2 - Bộ tính CRC nối tiếp với một giá trị khởi tạo](http://nguyenquanicd.blogspot.com/2019/08/crc-bai-2-logic-tinh-crc-noi-tiep-voi.html)

[Bài 3 - Bộ tính CRC song song toàn phần](http://nguyenquanicd.blogspot.com/2019/08/crc-bai-3-bo-tinh-crc-song-song.html)

[Bài 4 - Bộ tính CRC song song từng phần](http://nguyenquanicd.blogspot.com/2019/09/crc-bai-4-bo-tinh-crc-song-song-tung.html)

Trong các bài viết trên, các bộ tính CRC có đặc điểm như sau:

* Chuỗi bit đa thức sinh được biểu diễn theo cách thông thường (Normal representation)
* Giá trị khởi tạo ban đầu của thanh ghi CRC trong các bộ tính CRC là 0

Trên thực tế, kết quả tính toán CRC sẽ phụ thuộc vào một số thông số cấu hình. Với cùng một đa thức sinh, nếu các thông số cấu hình khác nhau, kết quả tính CRC sẽ khác nhau. Tùy vào giao thức truyền thông, việc tính toán CRC theo cấu hình nào sẽ được quy định cụ thể.

**2) Các thông số ảnh hưởng đến việc tính toán CRC**

**2.1) Cách biểu diễn chuỗi bit đa thức sinh (Polynomial Generator representation)**

Như đã nói ở phần 1, các bộ tính CRC đang sử dụng chuỗi bit đa thức sinh theo cách biểu diễn thông thường. Một đa thức sinh có nhiều cách biểu diễn chuỗi bit khác nhau. Mỗi cách biểu diễn sẽ cho giá trị chuỗi bit đa thức sinh khác nhau.

Đa thức sinh tổng quát của bộ tính CRC-n như sau:

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEgNBAo05x69WrPI_JE0hNA3ZU0utgAk4O5OqtvYjI2FBmov-lGzIhsjOGRKB4yjDl4YCHu6EvvKDQJKbncMPFiWDnL_c4OQGrE4CCFrHCBn0jNavxZYu51qKlCxclv_wf2LVWjCXDt8EZ0e/s1600/image.png)

Trong đó, *an* là 1 hoặc 0.

Chuỗi bit của đa thức sinh có các cách biểu diễn sau:

* Cách biểu diễn thông thường (normal representation)
* Cách biểu diễn đảo ngược (reversed representation)
* Cách biểu diễn Koopman (Koopman representation)
* Cách biểu diễn nghịch đảo (Reciprocal representation)

Xét đa thức sinh của một CRC-8 sau đây:

*Ví dụ 1 - Đa thức sinh CRC-8*

[](https://blogger.googleusercontent.com/img/b/R29vZ2xl/AVvXsEhI2nSZ-JKyBvGHUF1zPcW-xErPV77CrZjesZhlKRb0l1mdA75b25ZQ0w7fP0y06krxfLfbZ5pomaykVCgMa5dWxUtW3snHdejuxF-z46fVFwKsc5fk_4Xuw1Q_xqjl-z9pNjPHzxK-KXwp/s1600/image.png)

Đa thức sinh này sẽ được sử dụng để minh họa cho các cách biểu diễn chuỗi bit đa thức sinh.

**Cách biểu diễn thông thường** sẽ loại bỏ bit phần tử có trọng số cao nhất *x^n*. Trong một đa thức sinh, *an* luôn bằng 1. Chuỗi bit đa thức sinh, tính từ trái qua phải, sẽ chứa các bit từ n-1 đến 0. Áp dụng vào ví dụ 1, *x^8* sẽ được loại bỏ, chuỗi bit đa thức sinh sẽ từ *x^7* đến *x^0* là:

1101\_0101 (0xD5)

**Cách biểu diễn đảo ngược** sẽ loại bỏ bit phần tử có trọng số cao nhất *x^n*. Chuỗi bit đa thức sinh, tính từ trái qua phải, sẽ chứa các bit từ 0 đến n-1. Thứ tự này ngược với cách biểu diễn thông thường. Áp dụng vào ví dụ 1, *x^8* sẽ được loại bỏ, chuỗi bit đa thức sinh sẽ từ *x^0* đến *x^7* là:

1010\_1011 (0xAB)

**Cách biểu diễn Koopman** sẽ loại bỏ bit phần tử có trọng số thấp nhất *x^0*. Trong một đa thức sinh, *a0* luôn bằng 1 giống như *an*. Chuỗi bit đa thức sinh, tính từ trái qua phải, sẽ chứa các bit từ **n**đến **1**. Áp dụng vào ví dụ 1, *x^0* sẽ được loại bỏ, chuỗi bit đa thức sinh sẽ từ *x^8* đến *x^1* là:

1110\_1010 (0xEA)

**Các biểu diễn nghịch đảo** sẽ đảo lại toàn bộ thứ tự bit của đa thức sinh, x^0 thành x^n, x^1 thành x^n-1, ... sau đó, bit đầu tiên bên trái sẽ bị loại bỏ. Chuỗi bit đa thức sinh, tính từ trái qua phải, sẽ chứa các bit từ **1**đến **n**. Áp dụng vào ví dụ 1, *x^0* sẽ được loại bỏ, chuỗi bit đa thức sinh sẽ từ *x^1* đến *x^8* là:

0101\_0111 (0x57)

Như vậy, cùng với một chuỗi dữ liệu, giá trị CRC sẽ khác nhau nếu chuỗi bit đa thức sinh được biểu diễn theo các cách khác nhau.

**2.2) Giá trị khởi tạo ban đầu (Initial Value)**

Trong các thiết kế trình bày ở những bài trước, giá trị khởi tạo của các bộ tính là 0.  
Giá trị khởi tạo là giá trị sẽ được XOR với những bit đầu tiên của chuỗi dữ liệu trước khi tính CRC, bước này còn gọi là "Initial XOR". Giá trị khởi tạo có số bit bằng với số bit của chuỗi CRC.

Ví dụ, xét chuỗi dữ liệu như sau:

10011010\_10111100

Nếu chuỗi trên được đưa vào bộ tính CRC-8 với giá trị khởi tạo là 0 thì chuỗi bit dùng để tính toán CRC chính là chuỗi bit nguyên thủy trên. Nếu việc tính CRC yêu cầu giá trị khởi tạo là *8'hff*thì byte đầu tiên của chuỗi bit đầu vào sẽ được XOR với *8'hff* trước khi dùng để tính toán CRC. Chuỗi bit đầu vào trở thành:

**01100101**\_10111100

Như vậy, với các giá trị khởi tạo khác nhau, kết quả tính CRC sẽ khác nhau.  
**2.3) Thứ tự bit của các byte dữ liệu đầu vào**  
Trong các thiết kế ở những bài trước, các bit của chuỗi dữ liệu đầu vào được xử lý theo thứ tự thông thường. Thứ tự bit thông thường là thứ tự bit được đưa vào bộ tính CRC và bộ tính CRC không thay đổi thứ tự này.  
Tùy giao thức, thứ tự bit trong từng byte dữ liệu (8 bit) có thể bị đảo ngược trước khi tính CRC. Việc đảo ngược thứ tự bit trong từng byte gọi là "Reflected Input".  
Ví dụ, xét chuỗi dữ liệu như sau:

10011010\_10111100

Nếu tính CRC với cơ chế "Reflected input", thì chuỗi bit đầu vào sẽ được biến đổi thành:

**01011001\_00111101**

Từng nhóm 8 bit sẽ bị đảo ngược vị trí bit. Như vậy, tính CRC với "normal input" và "reflected input" sẽ cho kết quả chuỗi CRC khác nhau.  
**2.4) Thứ tự bit của kết quả tính CRC**  
Trong các thiết kế ở những bài trước, thứ tự bit của chuỗi CRC (kết quả tính CRC) là thứ tự thông thường. Thứ tự thông thường là thứ tự tự nhiên của chuỗi CRC sau khi tính toán. Trong thứ tự này, không bit nào bị thay đổi vị trí hiện tại của nó.  
Tùy giao thức, thứ tự bit của toàn bộ chuỗi CRC có thể bị đảo ngược, "MSB đến LSB" bị đảo thành "LSB đến MSB". Việc đảo ngược thứ tự bit của chuỗi CRC gọi là "Reflected Output". Lưu ý, việc đảo ngược thực hiện trên toàn bộ chuỗi CRC chứ không phải theo từng byte như "Reflected input". Ví dụ, nếu ta có một chuỗi kết quả tính CRC thông thường như sau:

10011010\_10111100

Nếu tính CRC với cơ chế "Reflected output", thì giá trị tính toán CRC sẽ là:

**00111101\_01011001**

Như vậy, tính CRC với "normal output" và "reflected output" sẽ cho kết quả chuỗi CRC khác nhau.

**2.5) XOR ngõ ra**  
Trong các thiết kế ở những bài trước, kết quả tính CRC được lấy trực tiếp từ thanh ghi tính toán CRC mà không có bất kỳ sự điều chỉnh giá trị nào. Kết quả này tạm gọi là kết quả thông thường (normal result).  
Tùy giao thức, kết quả thông thường có thể bị điều chỉnh bằng cách XOR với một giá trị cố định, gọi là "Final XOR hay XOR output". Ví dụ, nếu ta có một chuỗi kết quả tính CRC thông thường như sau:

10011010\_10111100

Nếu tính CRC dùng cơ chế "Final XOR" với giá trị được XOR là:

11111111\_11111111

thì giá trị tính toán CRC sẽ là:

**01100101\_0100011**

Kết quả thông thường ứng với trường hợp XOR với giá trị toàn bit 0. Như vậy, hằng số XOR ngõ ra khác nhau sẽ tạo ra chuỗi bit CRC khác nhau.  
**3) Thứ tự áp dụng các thông số tính toán trong quá trình tính CRC**  
Vấn đề tiếp theo cần quan tâm là khi có nhiều thông số cùng được áp dụng vào phép tính CRC thì thông số nào sẽ được áp dụng trước, thông số nào sẽ áp dụng sau? Ví dụ, một giao thức vừa yêu cầu có "Initial Value", vừa có "Reflected input" thì dữ liệu ngõ vào sẽ được XOR với giá trị khởi tạo trước rồi mới đảo bit hay sẽ được đảo bit trước rồi mới XOR?  
Sau đây là thứ tự áp dụng các thông số trong tính toán CRC:

1. Dữ liệu đầu vào
2. Đảo thứ tự bit trong từng byte (Reflected input)
3. XOR với giá trị khởi tạo (Initial XOR)
4. Tính toán CRC với chuỗi đa thức sinh
5. Đảo thự tự bit chuỗi CRC (Reflected output)
6. XOR với giá trị kết thúc (Final XOR)
7. Kết quả chuỗi CRC

|  |
| --- |
|  |
| Hình 1: Thứ tự áp dụng các thông số tính CRC |

**4) Bộ tính CRC song song hỗ trợ đầy đủ các thông số tùy chọn**  
Trong [bài 3](http://nguyenquanicd.blogspot.com/2019/08/crc-bai-3-bo-tinh-crc-song-song.html" \t "_blank), tác giả đã trình bày về cách thực hiện một bộ tính CRC song song toàn phần. Thiết kế nào sẽ được điều chỉnh lại hỗ trợ đầy đủ các tùy chọn tính CRC như đã trình bày.  
Các port được thêm vào gồm:

* *initXorValue*: Nạp giá trị cho bước "Initial XOR". Khi giá trị toàn bit 0 thì ngõ vào dữ liệu xem như không XOR với giá trị khởi tạo.
* *refInEn*: Cho phép đảo thứ tự bit trong từng byte của dữ liệu ngõ vào. Tích cực mức 1. Khi*ctrlEn=1* và *refInEn=1*thì chuỗi dữ liệu ngõ vào sẽ được đảo thứ tự bit.
* *refOutEn*: Cho phép đảo thứ tự bit chuỗi CRC ngõ ra. Tích cực mức 1. Khi *ctrlEn=1* và *refOutEn=1* thì chuỗi CRC sẽ được đảo thứ tự bit sau khi tính toán.
* *finalXorValue*: Nạp giá trị cho bước "Final XOR". Khi giá trị toàn bit 0 thì chuỗi CRC (ngõ ra) xem như không XOR.

*Ví dụ 2 - RTL code của "Reflected input"*

generate

genvar j;

for (j = 0; j < (DWIDTH/8); j=j+1) begin

assign refInput[j\*8+0] = dataIn[j\*8+7];

assign refInput[j\*8+1] = dataIn[j\*8+6];

assign refInput[j\*8+2] = dataIn[j\*8+5];

assign refInput[j\*8+3] = dataIn[j\*8+4];

assign refInput[j\*8+4] = dataIn[j\*8+3];

assign refInput[j\*8+5] = dataIn[j\*8+2];

assign refInput[j\*8+6] = dataIn[j\*8+1];

assign refInput[j\*8+7] = dataIn[j\*8+0];

end

endgenerate

*Ví dụ 3 - RTL code của "Initial XOR" sau bước "Reflected Input"*

assign xorInitInput = (refInEn?

refInput[DWIDTH-1:0]: dataIn)

^ {initXorValue[CRC\_WIDTH-1:0],

{DWIDTH-CRC\_WIDTH{1'b0}}};

*refInEn*được sử dụng để xác định xem bước "Reflected Input" có được áp dụng hay không. Nếu không, dữ liệu không đảo *dataIn*sẽ được XOR.  
  
*Ví dụ 4 - RTL code của "Reflected Output"*

generate

genvar k;

for (k = 0; k < CRC\_WIDTH; k=k+1) begin

assign refOutput[k] = crcSeq[CRC\_WIDTH-1-k];

end

endgenerate

*crcSeq*là thanh ghi chứa giá trị chuỗi CRC trước khi đảo.  
  
*Ví dụ 5 - RTL code của "Final XOR"*

always @ (posedge clk) begin

if (~rstN)

invOutEn <= 1'b0;

else if (ctrlEn)

invOutEn <= refOutEn;

end

assign crcOut[CRC\_WIDTH-1:0] =

(invOutEn? refOutput: crcSeq)

^ finalXorValue[CRC\_WIDTH-1:0];

*incOutEn*là bit sẽ cập nhật giá trị điều khiển từ *refOutEn*. Nếu *incOutEn=1* thì chuỗi kết quả CRC trước khi XOR sẽ bị đảo. Nếu *incOutEn=1* thì chuỗi kết quả CRC trước khi XOR sẽ lấy từ thanh ghi tính toán thông thường *crcSeq*.  
Các bạn hãy tải code về để xem chi tiết.  
**5) Kết quả mô phỏng**  
Sau đây là một kết quả mô phỏng minh họa với chuỗi đa thức sinh được chọn là 0x1021

* Trường hợp 1, tính CRC thông thường, không áp dụng Initial XOR, Reflected Input, Reflected Output và Final XOR

# --GenPoly:     **1021**  
# Initial value:     0000  
# Final XOR value:     0000  
# Data input: **9abcdef0**  
# refInEn: 0  
# refOutEn: 0  
# CRC result:     **fc9d**

* Trường hợp 2, tính CRC áp dụng Initial XOR khác 0

# --GenPoly:     **1021**  
# Initial value:     1d0f   
# Final XOR value:     0000  
# Data input: **9abcdef0**  
# refInEn: 0  
# refOutEn: 0  
# CRC result:     **f28d**  
#

* Trường hợp 3, tính CRC áp dụng Initial XOR và Final XOR khác 0

# --GenPoly:     **1021**  
# Initial value:     ffff   
# Final XOR value:     ffff   
# Data input: **9abcdef0**  
# refInEn: 0  
# refOutEn: 0  
# CRC result:     **87a2**

* Trường hợp 4, tính CRC áp dụng Initial XOR và Final XOR khác 0, Reflected input và Reflected output

# --GenPoly:     **1021**  
# Initial value:     ffff   
# Final XOR value:     ffff   
# Data input: **9abcdef0**  
# refInEn: 1   
# refOutEn: 1  
# CRC result:     d3fa  
  
Các bạn hãy dùng testbench đơn giản sau đây để thử:

**module**tbCrcParallelFull;  
  parameter CRC\_WIDTH = 16;  
  parameter DWIDTH    = 32; //Must be n byte with n=1,2,3,4,...  
  parameter TMP\_WIDTH = DWIDTH\*CRC\_WIDTH;  
  //  
  //Inputs  
  //  
  reg clk;    //System clock  
  reg rstN;   //System Reset  
  reg ctrlEn; //CRC enable  
  reg [DWIDTH-1:0] dataIn; //Data input  
  reg [CRC\_WIDTH-1:0] genPoly; //Generator Polynominal  
  reg [CRC\_WIDTH-1:0] initXorValue; //Initial value, set 0 if don't use  
  reg refInEn;  //Reverse bit order in each input byte  
  reg refOutEn; //Reverse bit order of CRC output  
  reg [CRC\_WIDTH-1:0] finalXorValue; //Final XOR value, set 0 if don't use  
  //  
  //Outputs  
  //  
  wire [CRC\_WIDTH-1:0] crcOut;  
  wire crcReady;  
  //  
  crcParallelFull crcParallelFull (  
   clk,  
   rstN,  
   ctrlEn,  
   dataIn,  
   genPoly,  
   initXorValue,  
   refInEn,  
   refOutEn,  
   finalXorValue,  
   crcOut,  
   crcReady  
   );  
  //  
  initial begin  
    clk = 0;  
    forever #5 clk = ~clk;  
  end  
  
  initial begin  
    rstN = 1'b0;  
    #20  
    rstN = 1'b1;  
  end  
  
  initial begin  
    ctrlEn = 1'b0;  
    genPoly = 16'h1021;  
    #26  
    initXorValue = 16'h0000;  
    finalXorValue = 16'h0000;  
    dataIn = 32'h9abc\_def0;  
    ctrlEn = 1'b1;  
    refInEn  = 1'b0;  
    refOutEn = 1'b0;  
    #10  
    ctrlEn = 1'b0;  
    refInEn  = 1'b0;  
    refOutEn = 1'b0;  
    #20  
    initXorValue = 16'h1D0F;  
    finalXorValue = 16'h0000;  
    dataIn = 32'h9abc\_def0;  
    ctrlEn = 1'b1;  
    refInEn  = 1'b0;  
    refOutEn = 1'b0;  
    #10  
    ctrlEn = 1'b0;  
    refInEn  = 1'b0;  
    refOutEn = 1'b0;  
    #20  
    initXorValue = 16'hffff;  
    finalXorValue = 16'hffff;  
    dataIn = 32'h9abc\_def0;  
    ctrlEn = 1'b1;  
    refInEn  = 1'b0;  
    refOutEn = 1'b0;  
    #10  
    ctrlEn = 1'b0;  
    refInEn  = 1'b0;  
    refOutEn = 1'b0;  
    #20  
    initXorValue = 16'hffff;  
    finalXorValue = 16'hffff;  
    dataIn = 32'h9abc\_def0;  
    ctrlEn = 1'b1;  
    refInEn  = 1'b1;  
    refOutEn = 1'b1;  
    #10  
    ctrlEn = 1'b0;  
    refInEn  = 1'b0;  
    refOutEn = 1'b0;  
    #200  
    $stop;  
  end  
  reg crcReadyReg;  
  wire risingCrcReady;  
  reg refInEnReg, refOutEnReg;  
  always @ (posedge clk) begin  
    if (ctrlEn) begin  
      refInEnReg <= refInEn;  
      refOutEnReg <= refOutEn;  
    end  
  end  
  always @ (posedge clk, negedge rstN) begin  
    if (~rstN) crcReadyReg <= 1;  
    else crcReadyReg <= crcReady;  
  end  
  assign risingCrcReady = ~crcReadyReg & crcReady;  
  always @ (posedge clk) begin  
   if (rstN & risingCrcReady)  
    $display ("--GenPoly: %8h \nInitial value: %8h \nFinal XOR value: %8h \nData input: %8h \nrefInEn: %b \nrefOutEn: %b\nCRC result: %8h\n",  
    genPoly, initXorValue, finalXorValue, dataIn, refInEnReg, refOutEnReg, crcOut);  
  end  
**endmodule**

Tác giả chọn các phép thử trên để bạn đọc có thể so sánh trên website tính CRC online sau:  
<https://crccalc.com/>  
  
Trong đó:

* Trường hợp 1 là CRC-16/XMODEM
* Trường hợp 2 là CRC-16/AUG-CCITT
* Trường hợp 3 là CRC-16/GENIBUS
* Trường hợp 4 là CRC-16/X-25

Như vậy, việc tính CRC phụ thuộc vào nhiều thông số cấu hình khác nhau. Mỗi giao thức (chuẩn) sẽ quy định cụ thể các thông số này.  
  
*Dữ liệu có thể tải:*  
[Source code trên Github](https://github.com/nguyenquanicd/crcCalculator)