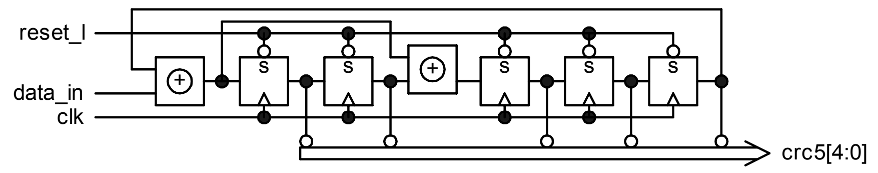
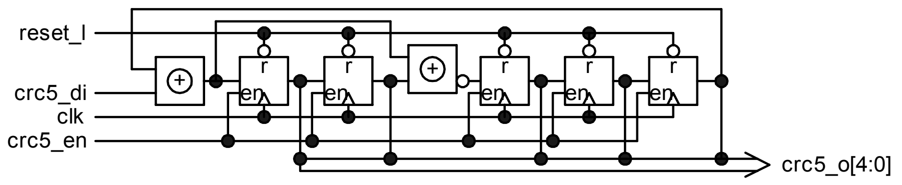
CRC5 (Cyclic Redundancy Check: $X^5+X^2+X^0$)

(A) General concept



(B) Emulation by C language (Omitted)

(C) Computer simulation by Verilog HDL



In general, CRC calculation is applied to a predetermined serial data field only. Accordingly, a certain timing signal to enable the CRC logic, “crc5_en”, must be made and provided.

In case of USB, CRC5 calculation is applied to only USB address and end point data field except SYNC, USB command, and, of course, CRC data field as below.

```
USB SETUP addr=15 endp=e CRC5[4:0]
                  |←-------→|
  SYNC     SETUP    addr  endp CRC5  Dummy
00000001-10110100-10101000-11110111-11111111

USB  OUT  addr=3a endp=a CRC5[4:0]
                  |←-------→|
  SYNC      OUT     addr  endp CRC5  Dummy
00000001-10000111-01011100-10111100-11111111

USB  IN   addr=70 endp=4 CRC5[4:0]
                  |←-------→|
  SYNC      IN      addr  endp CRC5  Dummy
00000001-10010110-00001110-01001110-11111111
```

(a) Verilog source code

(1) Test bench (crc5_sys.v)

Test bench generates a simulation vector that should be flexible. To make good test bench, some sort of programming skill backed up by polished sense is required.

The test bench makes 20MHz clock (“clk”), a serial data stream (“crc5_di”), CRC enable timing signal (“crc5_en”), reset timing signal (“reset_l”).

```
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
// Test bench for CRC5 (X5+X2+X0)
//  crc5_sys.v
//~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
`timescale     1ns / 1ns

module         crc5_sys;
```

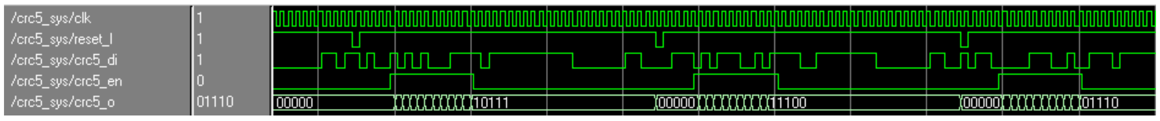
```
// Regs/wires/integers
reg            clk, reset_l, crc5_di, crc5_en;
wire    [4:0]  crc5_o;
integer        i, mask;
reg     [7:0]  crc5_di_8[14:0];

// Simulation target
crc5    crc5(.crc5_di(crc5_di), .crc5_en(crc5_en), .crc5_o(crc5_o), .reset_l(reset_l),
.clk(clk));

//--------------------
// Simulation vector
//--------------------
initial
begin
  $readmemh("crc5_di.dat", crc5_di_8);

     clk     <= 1'b1;
     reset_l <= 1'b0;
     crc5_en <= 1'b0;
     crc5_di <= 1'b0;
#110 reset_l <= 1'b1;
#10  crc5_di <= 1'b0;

  for(i = 0;  i < 15;  i = i + 1)
  begin
    for(mask = 8'h80;  mask > 0;  mask = (mask >> 1))
    begin
      crc5_di_gen;
    end
  end

#100  $finish;
end

// 20MHz clock generator
always  #25    clk <= ~clk;

//--------
// Tasks
//--------
task  crc5_di_gen;
begin
  if(((i % 5) == 1) && (mask == 8))      reset_l <= 1'b0;
  else                                   reset_l <= 1'b1;
  if((crc5_di_8[i] & mask) == 0)   #50  crc5_di <= 1'b0;
  else                             #50  crc5_di <= 1'b1;
  if(((i % 5) == 2) || (((i % 5) == 3) && (mask > 8'h10)))
  begin
        crc5_en <= 1'b1;
  end
  else  crc5_en <= 1'b0;
end
endtask

endmodule
```

(2) Target module (crc5.v)
Making a simple block diagram is recommended.

```
//~~~~~~~~~~~~~~~~~~~
// CRC5 (X5+X2+X0)
//  crc16.v
//~~~~~~~~~~~~~~~~~~~
module crc5(crc5_di, crc5_en, crc5_o, reset_l, clk);

// I/O definition
input          crc5_di;       // CRC5 data           (H)
```

```
input          crc5_en;       // CRC5 enable         (H)
input          reset_l;       // Reset               (L)
input          clk;           // Clock               (X)

output [4:0]   crc5_o;        // CRC5 out            (H)


// Regs for random logic
reg            crc5_0_in, crc5_2_in;

// Regs for DFFs
reg     [4:0]  crc5_o;


//---------------
// Random logic
//---------------
always @(crc5_di or crc5_o or crc5_0_in)
begin
  crc5_0_in  <=  (crc5_di   ^ crc5_o[4]);
  crc5_2_in  <= ~(crc5_0_in ^ crc5_o[1]);
end


//-------
// DFFs
//-------
always @(posedge(clk) or negedge(reset_l))
begin
  if(~reset_l)    crc5_o[4:0] <= 1'b0;
  else
  begin

    if(~crc5_en)  crc5_o[4:0] <= crc5_o[4:0];
    else
    begin
      crc5_o[4:3] <= crc5_o[3:2];
      crc5_o[2]   <= crc5_2_in;
      crc5_o[1]   <= crc5_o[0];
      crc5_o[0]   <= crc5_0_in;
    end
  end
end


endmodule
```

(b) Verilog simulation result



“crc5_di” is an input to the CRC5 generator. This contains USB SYNC code, command code, USB address, end point, and original CRC5 data to be compared with this CRC5 generator result. 3 USB command packets are transmitted.

“reset_l” resets the contents of CRC5 registers to prepare CRC calculation.

“crc5_en” is a timing signal to enable CRC5 calculation that applies to USB address and end point data field only.

“crc5_o[4:0]” is a result of the CRC5 calculation. After the original CRC5 data is received in full, they are compared to see if they match. CRC5 error is identified by the mismatch.