CRC16 (Cyclic Redundancy Check: $X^{16}+X^{15}+X^2+X^0$)

(A) General concept

(B) Emulation by C language (Omitted)
By the C emulation, I found that MMC (Multi Media Card) and SD (Secure Digital) flash memory card and the controller have implemented the CRC16 logic wrongly. It is not true CRC16 that performs $X^{16}+X^{15}+X^2+X^0$ unlike their specification. Accordingly, the error detection rate is miserably low.

(C) Computer simulation by Verilog HDL

(a) Verilog source code
(1) Test bench (crc16_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 ("crc16_di"), CRC enable timing signal ("crc16_en"), reset timing signal ("reset_l").
module crc16_sys;

// Regs/wires/integers
reg clk, reset_l, crc16_di, crc16_en;
wire [15:0] crc16_o;
integer i, mask;
reg [7:0] crc16_di_8[15:0];

// Simulation target
crc16 (.crc16_di(crc16_di), .crc16_en(crc16_en), .crc16_o(crc16_o),
reset_l(reset_l), .clk(clk));

//--------------------
// Simulation vector
//--------------------
initial
begin
$readmemh("crc16_di.dat", crc16_di_8);
clk <= 1'b1;
reset_l <= 1'b0;
crc16_en<='0b0;
crc16_di<='0b0;
#110 reset_l <= 1'b1;
#10 crc16_di<= 1'b0;
for(i = 0; i < 16; i = i + 1)
begin
  for(mask = 8'h80; mask > 0; mask = (mask >> 1))
  begin
    crc16_di_gen;
  end
end
#100 $finish;
end

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

//--
// Tasks
//--
task crc16_di_gen;
begin
  if(((i % 8) == 1) && (mask == 8)) reset_l <= 1'b0;
  else reset_l <= 1'b1;
  if((crc16_di_8[i] & mask) == 0) #50 crc16_di <= 1'b0;
  else #50 crc16_di <= 1'b1;
  if(((i % 8) > 1) && ((i % 8) < 6)) crc16_en <= 1'b1;
  else crc16_en <= 1'b0;
end
taskendmodule

(2) Target module (crc16.v)
Making a simple block diagram is recommended.

//~~~~~~~~~~~~~~~
// CRC16 (X16+X15+X2+X0)
// crc16.v
//~~~~~~~~~~~~~~~
module crc16(crc16_di, crc16_en, crc16_o, reset_l, clk);

// I/O definition
input crc16_di;  // CRC16 data (H)
input crc16_en;  // CRC16 enable (H)

CRC16 (Cyclic Redundancy Check: X^16+X^15+X^2+X^0) 2  (c) 2002 Oguchi R&D
input  reset_l; // Reset        (L)
input  clk;  // Clock  (X)
output [15:0] crc16_o; // CRC16 out  (H)

// Regs for random logic
reg  crc16_0_in, crc16_2_in, crc16_15_in;

// Regs for DFFs
reg [15:0] crc16_o;

//------------------
// Random logic
//------------------
always @(crc16_di or crc16_o or crc16_0_in)
begin
  crc16_0_in  <=  (crc16_di   ^ crc16_o[15]);
  crc16_2_in  <= ~(crc16_0_in ^ crc16_o[1]);
  crc16_15_in <= ~(crc16_0_in ^ crc16_o[14]);
end

//------------------
// DFFs
//------------------
always @(posedge(clk) or negedge(reset_l))
begin
  if(~reset_l)     crc16_o[15:0] <= 1'b0;
  else
  begin
    if(~crc16_en)  crc16_o[15:0] <= crc16_o[15:0];
    else
    begin
      crc16_o[15]   <= crc16_15_in;
      crc16_o[14:3] <= crc16_o[13:2];
      crc16_o[2]    <= crc16_2_in;
      crc16_o[1]    <= crc16_o[0];
      crc16_o[0]    <= crc16_0_in;
    end
  end
end

endmodule

(b) Verilog simulation result

"crc16_di" is an input to the CRC16 generator. This contains USB SYNC code, command code, data, and original CRC16 data to be compared with this CRC16 generator result.

"reset_l" resets the contents of CRC16 registers to prepare CRC calculation.

"crc16_en" is a timing signal to enable CRC16 calculation that applies to data field only.

"crc16_o[15:0]" is a result of CRC16 calculation. After the original CRC16 data is received in full, they are compared to see if they match. CRC error is identified by the mismatch.