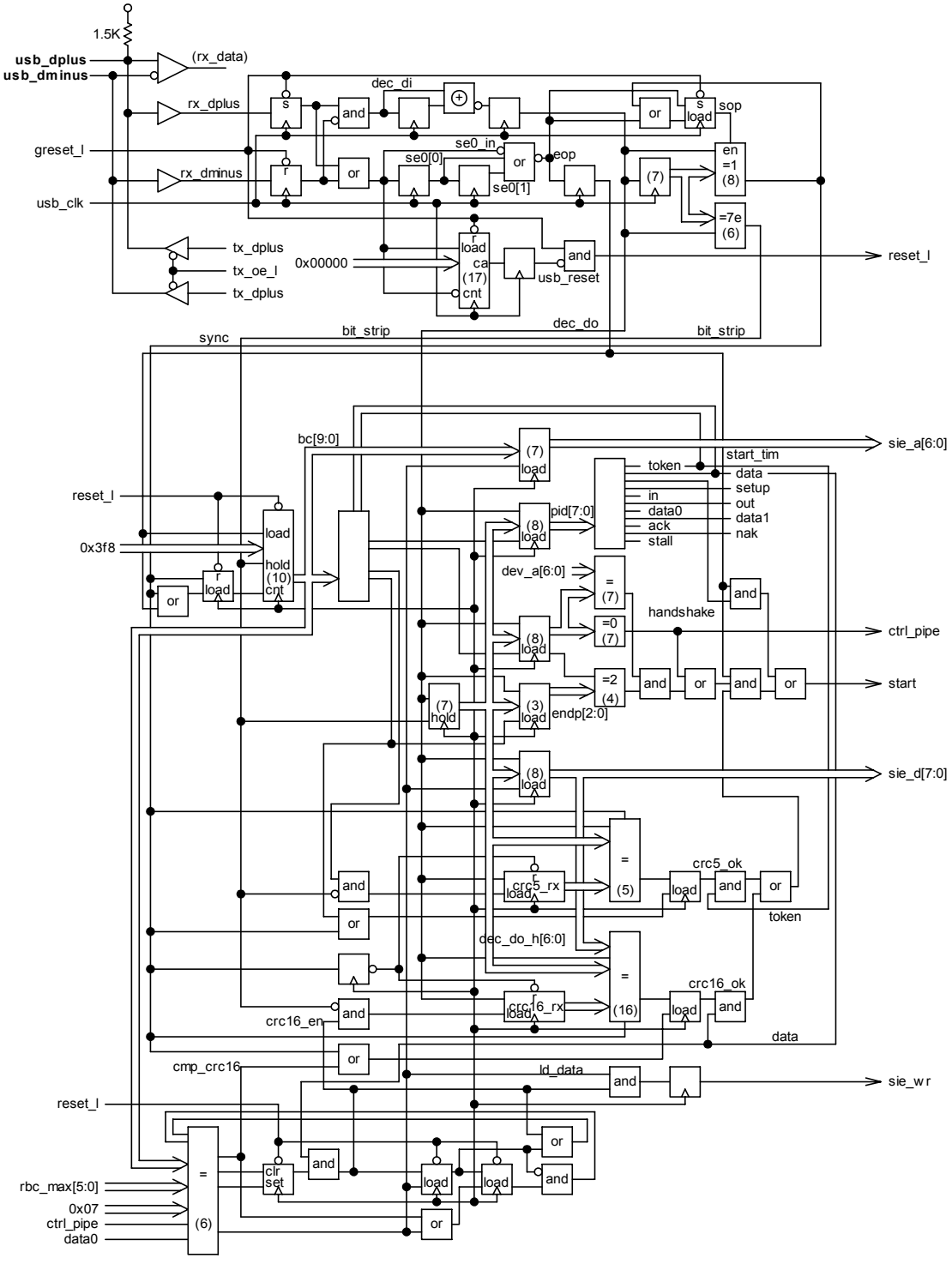
USB SIE Receiver

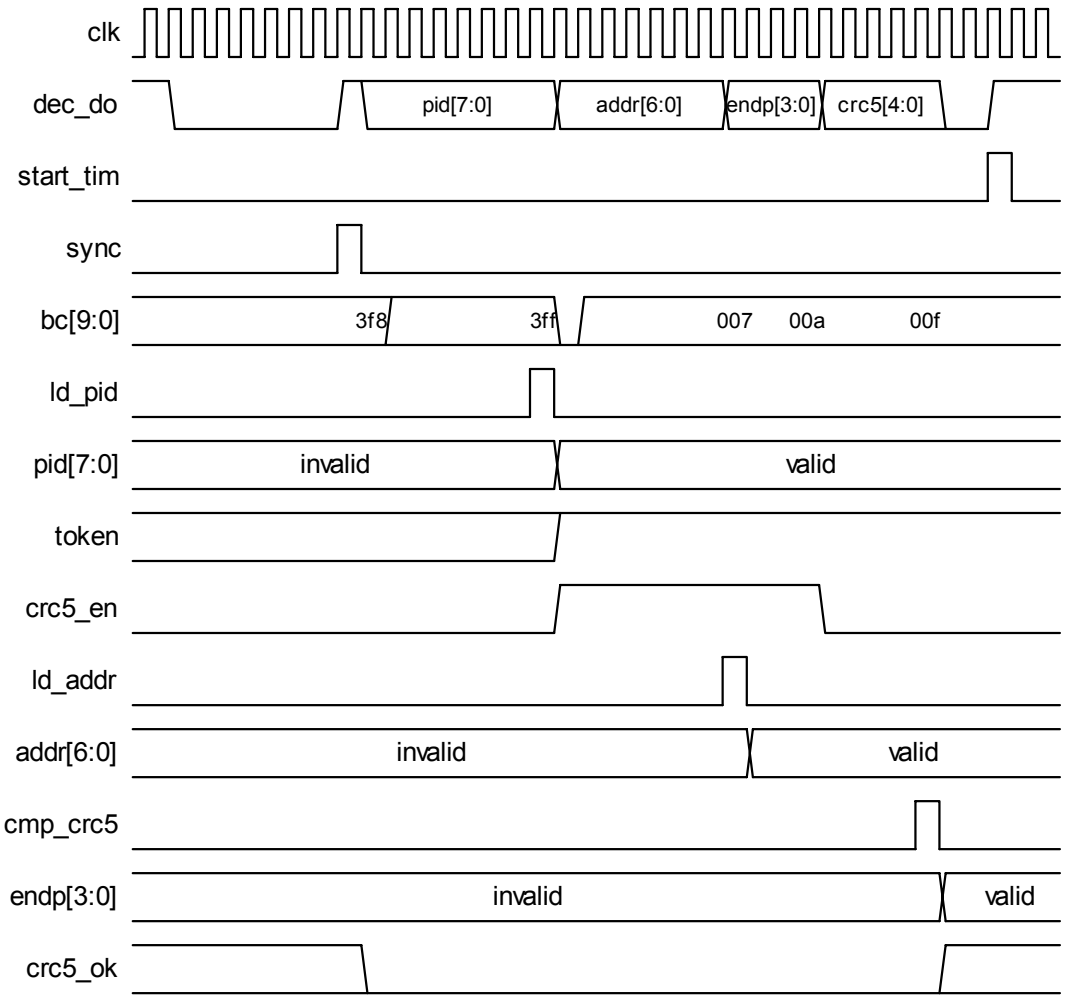
(A) Computer simulation by Verilog HDL

(a) Block Diagram

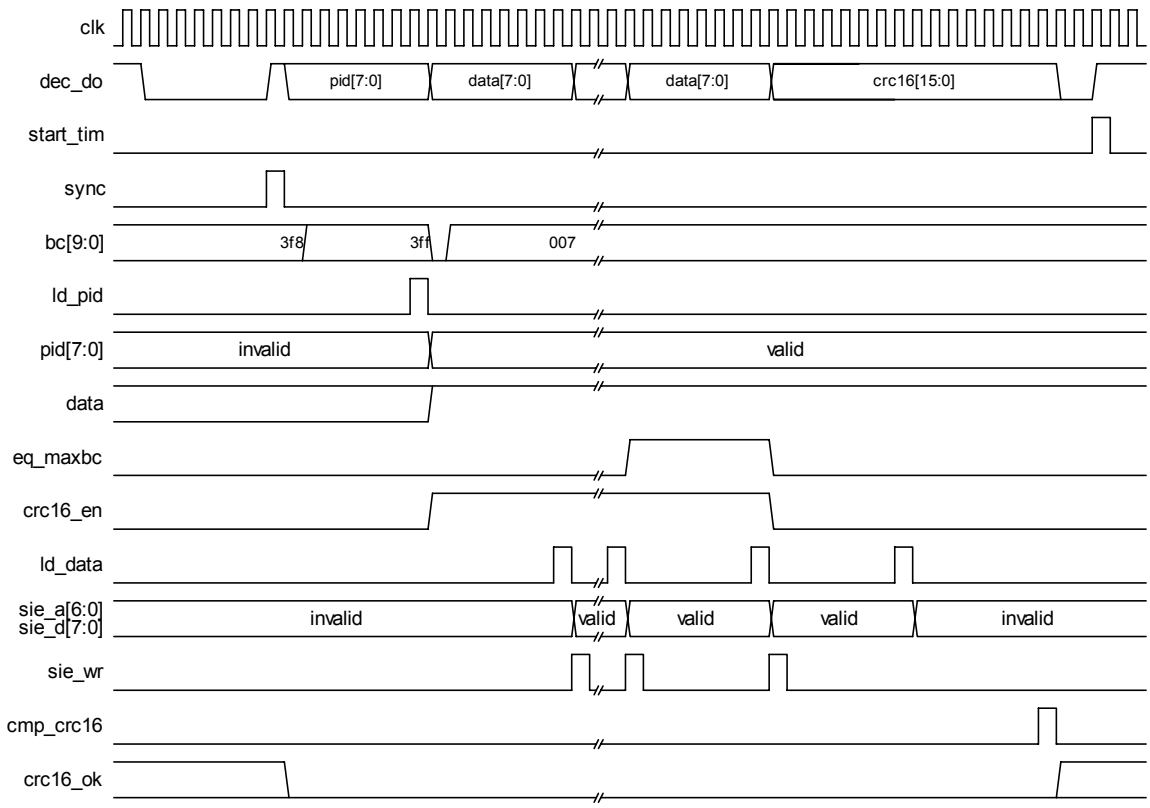


The block diagram is not furnished to make actual LSI or FPGA by making top level portion that defines pads and I/O buffers. Also, a certain signals to be provided by other functional modules in actual design are temporarily connected to fixed value in this example.
However, it is truly applicable to actual design.

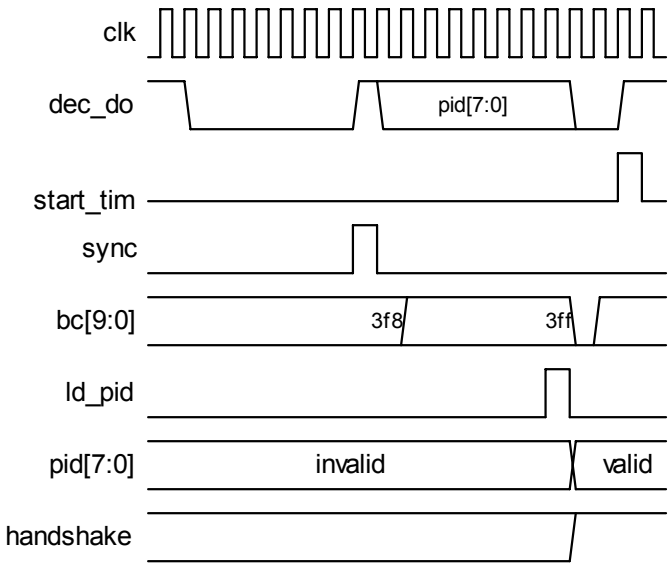
(b) Timing charts



Token packet decoding



Data packet decoding



Handshake packet decoding

(c) Verilog source code

(1) Test bench (usb_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”), bi-directional differential serial data stream (“usb_dplus” and “usb_dminus”), and reset timing signal (“reset_l”).

```
//~~~~~~~~~~~~~~~~~~~~~
// Test bench for USB
//  usb_sys.v
//~~~~~~~~~~~~~~~~~~~~~
`timescale     1ns / 1ns

module         usb_sys;

// Define directives
parameter max_bytes = 26;

// Regs/wires/integers
reg            usb_oe_l, clk, reset_l;
reg            usb_di, enc_do, enc_do_set;
reg            usb_dplus_drv, usb_dminus_drv;
reg            usb_dplus_reg, usb_dminus_reg;
wire           usb_dplus, usb_dminus;
integer        i, j, mask, max_tran, max_wait;
reg    [7:0]   usb_di_8[(max_bytes - 1):0];

// Simulation target
usb     sie_rx(.usb_dplus(usb_dplus), .usb_dminus(usb_dminus), .greset_l(reset_l),
.clk(clk));

//--------------------
// Simulation vector
//--------------------
initial
begin
  $readmemh("usb_di.dat", usb_di_8);

     clk        <= 1'b1;
     reset_l    <= 1'b0;
     usb_oe_l   <= 1'b1;
     usb_di     <= 1'b1;
     enc_do_set <= 1'b0;
#110 reset_l    <= 1'b1;
#10  usb_oe_l   <= 1'b1;

  usb_di_gen;

#100  $finish;
end

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


//-------------------------
// Bidirectional notation
//-------------------------
always @(posedge(clk))
begin
  usb_dplus_reg  <= (~usb_oe_l) ? usb_dplus_drv  : 1'bz;
  usb_dminus_reg <= (~usb_oe_l) ? usb_dminus_drv : 1'bz;
end

assign  usb_dplus  = usb_dplus_reg;
assign  usb_dminus = usb_dminus_reg;
```

```
//-------
// DFFs
//-------
always @(posedge(clk) or negedge(reset_l))
begin
  if(~reset_l)      enc_do <= 1'b1;
  else
  begin
    if(enc_do_set)  enc_do <= 1'b1;
    else            enc_do <= ~(usb_di ^ enc_do);
  end
end


//--------
// Tasks
//--------
task  usb_di_gen;
begin
  i = 0;
  while(i < (max_bytes))
  begin
    max_tran = usb_di_8[i];
    i = i + 1;
    for(j = 0;  j < max_tran;  j = j + 1)
    begin
      for(mask = 8'h01;  mask < 8'h81;  mask = (mask << 1))
      begin
        if((usb_di_8[i] & mask) == 0)
        begin
          usb_di         <= 1'b0;
          usb_dplus_drv  <= ~enc_do;
          usb_dminus_drv <=  enc_do;
        end
        else
        begin
          usb_di         <= 1'b1;
          usb_dplus_drv  <=  enc_do;
          usb_dminus_drv <= ~enc_do;
        end
          usb_oe_l <= 1'b0;
      #50 usb_oe_l <= 1'b0;
      end
    i = i + 1;
    end
      usb_dplus_drv  <= 1'b0;
      usb_dminus_drv <= 1'b0;
  #50 usb_dplus_drv  <= 1'b0;
      usb_dminus_drv <= 1'b0;
  #50 usb_dplus_drv  <= 1'b1;
      usb_dminus_drv <= 1'b0;
  #50 usb_oe_l       <= 1'b1;
    enc_do_set <= 1'b1;
    max_wait = usb_di_8[i];
    i = i + 1;
    for(j = 0;  j < max_wait;  j = j + 1)
    begin
    #50 usb_di <= 1'b1;
      enc_do_set <= 1'b0;
    end
  end
end
endtask

endmodule
```

(2) USB SIE receiver (sie_rx.v)

This module includes all of the functions need to SIE receiver such as NRZI decoder, Stuffed bit stripping, Unpacking packet ID, address, endpoint, CRC5, data, & CRC16, Decoding packet Ids, reset, EOP, & suspend, Real time CRC16 and CRC5 generation & check, and Sequence control of DATA0 & DATA1.
To store the data transmitted by system through USB, the address, data, and memory control signals are provided to bi-directional FIFO RAM physically merged in a unified SRAM.
In this example, DPLL (Digital Phase-Locked Loop) that equips 4 times over-sampling is not discussed.
The combination with USB SIE transmitter is not discussed here as well.

```
//~~~~~~~~~~~~~~~~~~~
// USB SIE receiver
//  sie_rx.v
//~~~~~~~~~~~~~~~~~~~
`timescale     1ns / 1ns

module sie_rx(usb_dplus, usb_dminus, greset_l, clk);

// I/O definition
inout          usb_dplus;     // USB data plus       (H)
inout          usb_dminus;    // USB data minus      (L)

input          greset_l;      // Global reset               (L)
input          clk;           // Clock               (X)


// Regs for random logic
tri1           usb_dplus;
tri0           usb_dminus;
wire    [15:0] crc16_o;
wire    [4:0]  crc5_o;
reg            rx_dplus, rx_dminus;
reg            dec_di, dec_do_in;
reg            se0_in, eop;
reg            sync, bit_strip, sop_load, reset_l;
reg            bc_cnt_en_load, pid_load, addr_load;
reg            cmp_crc5, crc5_en, crc5_load, crc5_ok_load, crc5_ok_in;
reg            setup, in, out, data0, data1, ack, nack, stall;
reg            token, data, handshake;
reg            ctrl_pipe, cs, start, crc16_en_set, crc16_en_clr;
reg            ld_data, crc16_load, cmp_crc16, crc16_ok_in, crc16_ok_load;
reg            crc16_en, sie_wr_in, cmp_crc_t;

// Temporary
wire    [6:0]  dev_a;
wire    [5:0]  rbc_max;

// Regs for DFFs
reg     [17:0] usb_reset_cnt;
reg     [9:0]  bc;
reg     [7:0]  dec_do, pid, sie_d;
reg     [6:0]  dec_do_h, addr, sie_a;
reg     [3:0]  endp;
reg     [1:0]  se0;
reg            rx_dplus_1d, rx_dminus_1d, dec_di_1d;
reg            sop, eop_7d, eop_6d, eop_5d, eop_4d, eop_3d, eop_2d, eop_1d;
reg            bc_cnt_en, sync_1d, crc5_ok, crc16_ok;
reg            sie_wr, crc16_en_ff, crc16_en_1d, crc16_en_2d;

//---------------
// Random logic
//---------------
assign usb_dplus  = 1'bz;
assign usb_dminus = 1'bz;

always @(usb_dplus or usb_dminus)
begin
  rx_dplus  <= usb_dplus;
```

```
  rx_dminus <= usb_dminus;
end

always @(rx_dplus_1d or rx_dminus_1d or dec_di or dec_di_1d)
begin
  dec_di    <= (rx_dplus_1d & ~rx_dminus_1d);
  dec_do_in <= ~(dec_di ^ dec_di_1d);
end

always @(rx_dplus_1d or rx_dminus_1d or se0 or se0_in)
begin
  se0_in <= (rx_dplus_1d | rx_dminus_1d);
  eop    <= ~(se0[1] | se0[0] | ~se0_in);
end

always @(dec_do or sop or sync or eop)
begin
  sync      <= (dec_do[7:0] == 8'h01) & sop;
  bit_strip <= (dec_do[7:0] == 8'h7e);
  sop_load  <= (sync | eop);
end

always @(greset_l or usb_reset_cnt)
begin
  reset_l <= (greset_l & ~usb_reset_cnt[17]);
end

always @(eop_6d or sync)
begin
  bc_cnt_en_load <= (eop_6d | sync);
end

always @(bc or token or crc5_en or bit_strip)
begin
  pid_load  <= (bc[9:0] == 10'h3ff);
  addr_load <= (bc[9:0] == 10'h007) & token;
  cmp_crc5  <= (bc[9:0] == 10'h00f) & token;
  crc5_en   <= (bc[9:4] == 6'h00)   & token &
               ((bc[3] == 1'b0) | ((bc[3:2] == 2'b10) & ~(bc[1:0] == 2'b11)));
  crc5_load <= (crc5_en & ~bit_strip);
end

always @(pid or setup or in or out or data0 or data1 or ack or nack or stall)
begin
  setup     <= (pid[7:0] == 8'h2d);
  in        <= (pid[7:0] == 8'h69);
  out       <= (pid[7:0] == 8'he1);
  data0     <= (pid[7:0] == 8'hc3);
  data1     <= (pid[7:0] == 8'h4b);
  ack       <= (pid[7:0] == 8'hd2);
  nack      <= (pid[7:0] == 8'h5a);
  stall     <= (pid[7:0] == 8'h1e);
  token     <= (setup | in   | out);
  data      <= (data0 | data1);
  handshake <= (ack   | nack | stall);
end

always @(sync or crc5_o or dec_do_h or dec_do or cmp_crc5 or sync or token)
begin
  crc5_ok_in <= (~sync & (crc5_o[4:0] == {dec_do_h[3:0], dec_do[0]}) & token);
  crc5_ok_load <= (cmp_crc5 | sync);
end

always @(addr or dev_a or endp)
begin
  ctrl_pipe <= (addr[6:0] == 7'h00);
  cs        <= (addr[6:0] == dev_a[6:0]) & (endp[3:0] == 4'h2);
end

always @(eop_7d or handshake or ctrl_pipe or cs or crc5_ok or crc16_ok)
begin
```

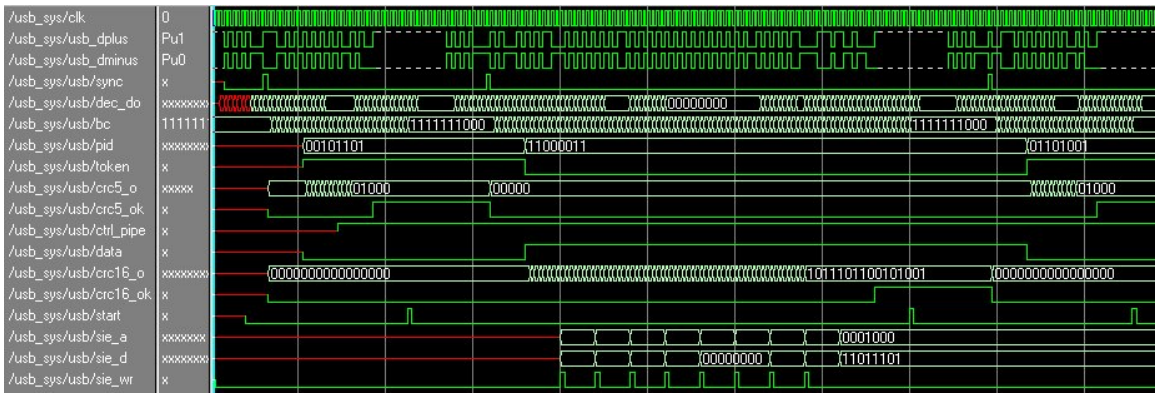
```
  start <= eop_7d & (handshake | ((ctrl_pipe | cs) & (crc5_ok | crc16_ok)));
end

always @(bc or ctrl_pipe or data0 or rbc_max or crc16_en or
crc16_en_1d or crc16_en_2d or crc16_en_ff or data or ld_data)
begin
  crc16_en_set <= (bc[9:0] == 10'h3ff);
  crc16_en_clr <= (ctrl_pipe & data0) ? (bc[9:0] == 10'h03f) :
                  ({rbc_max[5:0], 3'b111} == bc[8:0]);
  ld_data      <= (crc16_en | crc16_en_1d) & (bc[2:0] == 3'h7);
  cmp_crc16    <= (crc16_en_2d & ~crc16_en_1d) & (bc[2:0] == 3'h7);
  crc16_load   <= (crc16_en & ~bit_strip);
  crc16_en     <= (crc16_en_ff & data);
  sie_wr_in    <= (ld_data & crc16_en);
end

always @(sync or crc16_o or sie_d or dec_do_h or dec_do or data or cmp_crc16)
begin
  crc16_ok_in  <= (~sync & (crc16_o[15:0] == {sie_d[0], sie_d[1], sie_d[2], sie_d[3],
  sie_d[4], sie_d[5], sie_d[6], sie_d[7], dec_do_h[6:0], dec_do[0]}) & data);
  crc16_ok_load <= (cmp_crc16 | sync);
end

always @(ld_data or cmp_crc16)
begin
  cmp_crc_t <= (ld_data | cmp_crc16);
end

// Temporary reaction (dev_a[6:0] must be assigned iby system and set by processor.)
assign dev_a[6:0]   = 7'h12;
assign rbc_max[5:0] = 6'h20;

//-------
// DFFs
//-------
always @(posedge(clk) or negedge(greset_l))
begin
  if(~greset_l) rx_dplus_1d <= 1'b1;
  else          rx_dplus_1d <= rx_dplus;
end

always @(posedge(clk) or negedge(greset_l))
begin
  if(~greset_l) rx_dminus_1d <= 1'b0;
  else          rx_dminus_1d <= rx_dminus;
end

always @(posedge(clk) or negedge(greset_l))
begin
  if(~greset_l)   sop <= 1'b1;
  else
  begin
    if(sop_load)  sop <= eop;
    else          sop <= sop;
  end
end

always @(posedge(clk) or negedge(greset_l))
begin
  if(~greset_l) usb_reset_cnt[17:0] <= 18'h00000;
  else
  begin
    if(se0_in)  usb_reset_cnt[17:0] <= 18'h00000;
    else        usb_reset_cnt[17:0] <= usb_reset_cnt[17:0] + 1;
  end
end

always @(posedge(clk))
begin
  dec_do[7:1] <= dec_do[6:0];
  dec_do[0]   <= dec_do_in;
```

```
  dec_di_1d   <= dec_di;
  se0[1]      <= se0[0];
  se0[0]      <= se0_in;
  eop_7d      <= eop_6d;
  eop_6d      <= eop_5d;
  eop_5d      <= eop_4d;
  eop_4d      <= eop_3d;
  eop_3d      <= eop_2d;
  eop_2d      <= eop_1d;
  eop_1d      <= eop;
end

always @(posedge(clk) or negedge(reset_l))
begin
  if(~reset_l) bc_cnt_en <= 1'b0;
  else
  begin
    if(bc_cnt_en_load) bc_cnt_en <= sync;
    else               bc_cnt_en <= bc_cnt_en;
  end
end

always @(posedge(clk) or negedge(reset_l))
begin
  if(~reset_l)       bc[9:0] <= 10'h3f8;
  else
  begin
    if(eop_6d)       bc[9:0] <= 10'h3f8;
    else
    begin
      if(bc_cnt_en)  bc[9:0] <= bc[9:0] + 1;
      else           bc[9:0] <= bc[9:0];
    end
  end
end

always @(posedge(clk))
begin
  if(bit_strip)  dec_do_h[6:0] <= dec_do_h[6:0];
  else
  begin
    dec_do_h[6:1] <= dec_do_h[5:0];
    dec_do_h[0]   <= dec_do[0];
  end
end

always @(posedge(clk))
begin
  if(pid_load)
  begin
    pid[7] <= dec_do[0];
    pid[6] <= dec_do_h[0];
    pid[5] <= dec_do_h[1];
    pid[4] <= dec_do_h[2];
    pid[3] <= dec_do_h[3];
    pid[2] <= dec_do_h[4];
    pid[1] <= dec_do_h[5];
    pid[0] <= dec_do_h[6];
  end
  else  pid[7:0] <= pid[7:0];
end

always @(posedge(clk))
begin
  if(addr_load)
  begin
    addr[6] <= dec_do_h[0];
    addr[5] <= dec_do_h[1];
    addr[4] <= dec_do_h[2];
    addr[3] <= dec_do_h[3];
    addr[2] <= dec_do_h[4];
```

```
    addr[1] <= dec_do_h[5];
    addr[0] <= dec_do_h[6];
    endp[0] <= dec_do[0];
  end
  else
  begin
    addr[6:0] <= addr[6:0];
    endp[0]   <= endp[0];
  end
end

always @(posedge(clk))
begin
  if(cmp_crc5)
  begin
       endp[3] <= dec_do_h[4];
       endp[2] <= dec_do_h[5];
       endp[1] <= dec_do_h[6];
  end
  else endp[3:1] <= endp[3:1];
end

always @(posedge(clk))
begin
  if(crc5_ok_load)  crc5_ok <= crc5_ok_in;
  else              crc5_ok <= crc5_ok;
end

always @(posedge(clk))
begin
  if(crc16_ok_load) crc16_ok <= crc16_ok_in;
  else              crc16_ok <= crc16_ok;
end

always @(posedge(clk))
begin
  if(ld_data)
  begin
    sie_a[6:0] <= bc[9:3];
    sie_d[7:0]  <=  {dec_do[0],  dec_do_h[0],  dec_do_h[1],  dec_do_h[2],  dec_do_h[3],
dec_do_h[4], dec_do_h[5], dec_do_h[6]};
  end
  else
  begin
    sie_a[6:0] <= sie_a[6:0];
    sie_d[7:0] <= sie_d[7:0];
  end
end

always @(posedge(clk) or negedge(reset_l))
begin
  if(~reset_l)         crc16_en_ff <= 1'b0;
  else
  begin
    if(crc16_en_set)   crc16_en_ff <= 1'b1;
    else
    begin
      if(crc16_en_clr) crc16_en_ff <= 1'b0;
      else             crc16_en_ff <= crc16_en_ff;
    end
  end
end

always @(posedge(clk) or negedge(reset_l))
begin
  if(~reset_l)  crc16_en_1d <= 1'b0;
  else
  begin
    if(ld_data) crc16_en_1d <= crc16_en;
    else        crc16_en_1d <= crc16_en_1d;
  end
```

```
end

always @(posedge(clk) or negedge(reset_l))
begin
  if(~reset_l)    crc16_en_2d <= 1'b0;
  else
  begin
    if(cmp_crc_t) crc16_en_2d <= crc16_en_1d;
    else          crc16_en_2d <= crc16_en_2d;
  end
end

always @(posedge(clk))
begin
  sync_1d <= sync;
  sie_wr  <= sie_wr_in;
end

//----------------
// Lower modules
//----------------
crc5          crc5(.crc5_di(dec_do[0]),     .crc5_en(crc5_load),     .crc5_o(crc5_o[4:0]),
.reset_l(~sync_1d), .clk(clk));
crc16    crc16(.crc16_di(dec_do[0]),    .crc16_en(crc16_load),    .crc16_o(crc16_o[15:0]),
.reset_l(~sync_1d), .clk(clk));


endmodule
```

(d) Verilog simulation result



"usb_dplus" and "usb_dminus" are differential bi-directional tri-state USB data stream. In case of Full speed (12MHz) as shown here, "usb-dplus" = 1 and "usb_dminus" = 0 when no signal is transmitted. They are NRZI encoded signals.
EOP (End Of Packet) is asserted at the end of the packets.

Sync code must be detected first. PID (Packet IDentification) must be unpacked and decoded. Real time CRC5 generation must be done and the value must be compared with CRC5 reference data transmitted by system. If they are equivalent, "crc5_ok" goes high if the packet is a token. Then, "start" is issued to inform embedded processor to start processing.

In case of data packet asserted by system, real time CRC16 generation is activated instead of CRC5 and "crc16_ok" must go high. If "crc16_ok" = 0, NACK must be transmitted by this device to request system to resend the same data again. The data must be written into data buffer by providing the address ("sie_a"), data ("sie_d"), and memory write ("sie_wr").