

PROJECT APB UART

Prepared by :
mohamed khames

Prepared for :
Eng. mohamed tareq



Design code for TX :-

```
2 module uart_tx (
3     input wire      clk,
4     input wire      rst,          // synchronous active-high
5     input wire [31:0] baud_div,
6     input wire      tx_start,    // 1-cycle pulse
7     input wire [7:0] tx_data,
8     output reg      tx,
9     output reg      tx_busy,
10    output reg      tx_done
11 );
12
13 // states
14 localparam S_IDLE = 3'd0;
15 localparam S_START = 3'd1;
16 localparam S_DATA = 3'd2;
17 localparam S_STOP = 3'd3;
18 localparam S_DONE = 3'd4;
19
20 reg [2:0] state, nxt_state;
21 reg [31:0] baud_cnt;
22 reg [2:0] bit_cnt;
23 reg [7:0] shift_reg;
24
25 // sequential
26 always @(posedge clk) begin
27     if (rst) begin
28         state <= S_IDLE;
29         baud_cnt <= 32'd0;
30         bit_cnt <= 3'd0;
31         shift_reg <= 8'd0;
32         tx <= 1'b1;
33         tx_busy <= 1'b0;
34         tx_done <= 1'b0;
35     end else begin
36         state <= nxt_state;
37
38         // default tx_done low; pulse one cycle when entering DONE
39         if (state == S_DONE)
40             tx_done <= 1'b1;
41         else
42             tx_done <= 1'b0;
43
44         case (state)
45             S_IDLE: begin
46                 tx <= 1'b1;
47                 tx_busy <= 1'b0;
48                 if (tx_start) begin
49                     shift_reg <= tx_data;
50                     // load counter for one bit: use baud_div-1 so it counts down to 0 inclusive
51                     baud_cnt <= (baud_div == 0) ? 32'd1 : baud_div - 1;
52                     bit_cnt <= 3'd0;
53                     tx_busy <= 1'b1;
54                 end
55             end
56
57             S_START: begin
58                 tx <= 1'b0; // start bit
59                 tx_busy <= 1'b1;
60                 if (baud_cnt == 0) begin
61                     // finished start bit; reload for data bit
62                     baud_cnt <= (baud_div == 0) ? 32'd1 : baud_div - 1;
63                 end else begin
64                     baud_cnt <= baud_cnt - 1;
65                 end
66             end
67
68             S_DATA: begin
69                 tx <= shift_reg[0];
70                 tx_busy <= 1'b1;
71                 if (baud_cnt == 0) begin
72                     // shift after bit time
73                     shift_reg <= {1'b0, shift_reg[7:1]};
74                     bit_cnt <= bit_cnt + 1;
75                     baud_cnt <= (baud_div == 0) ? 32'd1 : baud_div - 1;
76                 end else begin
77                     baud_cnt <= baud_cnt - 1;
78                 end
79             end
80
81             S_STOP: begin
82                 tx <= 1'b1; // stop bit
83                 tx_busy <= 1'b1;
84                 if (baud_cnt == 0) begin
85                     // finished
86                 end else begin
87                     baud_cnt <= baud_cnt - 1;
88                 end
89             end
90
91             S_DONE: begin
```

```

92         tx <= 1'b1;
93         tx_busy <= 1'b0;
94     end
95
96     default: begin
97         tx <= 1'b1;
98         tx_busy <= 1'b0;
99     end
100 endcase
101 end
102 end
103
104 // combinational next-state
105 always @(*) begin
106     nxt_state = state;
107     case (state)
108         S_IDLE: if (tx_start) nxt_state = S_START;
109         S_START: if (baud_cnt == 32'd0) nxt_state = S_DATA;
110         S_DATA: if ((baud_cnt == 32'd0) && (bit_cnt == 3'd7)) nxt_state = S_STOP;
111         S_STOP: if (baud_cnt == 32'd0) nxt_state = S_DONE;
112         S_DONE: nxt_state = S_IDLE;
113     endcase
114 end
115
116 endmodule
117

```

Test bunch for TX :-

```

2 module tb_uart_tx;
3     reg clk;
4     reg rst;
5     reg [31:0] baud_div;
6     reg tx_start;
7     reg [7:0] tx_data;
8     wire tx;
9     wire tx_busy;
10    wire tx_done;
11
12    // Clock 100MHz = 10ns period
13    initial clk = 0;
14    always #5 clk = ~clk;
15
16    // DUT
17    uart_tx uut (
18        .clk(clk),
19        .rst(rst),
20        .baud_div(baud_div),
21        .tx_start(tx_start),
22        .tx_data(tx_data),
23        .tx(tx),
24        .tx_busy(tx_busy),
25        .tx_done(tx_done)
26    );
27
28    initial begin
29        $dumpfile("tb_uart_tx.vcd");
30        $dumpvars(0, tb_uart_tx);
31
32        // init
33        rst = 1;
34        baud_div = 32'd4;
35        tx_start = 0;
36        tx_data = 8'h00;
37
38        // reset pulse
39        #20 rst = 0;
40        #20;
41    end
42

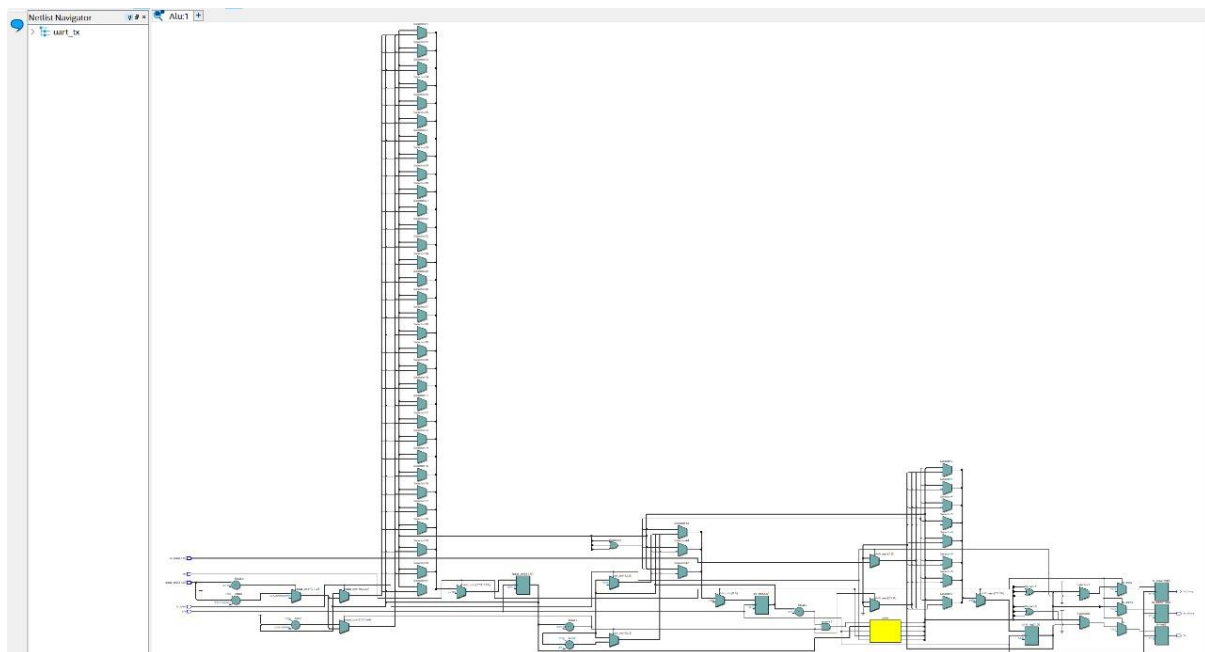
```

```

41
42 // send 0x55
43 tx_data = 8'h55;
44 @(posedge clk);
45 tx_start = 1; // pulse
46 @(posedge clk);
47 tx_start = 0;
48
49 wait (tx_done);
50 $display("Finished sending 0x55 at time %0t", $time);
51
52 // delay then send 0xA3
53 #50;
54 tx_data = 8'hA3;
55 @(posedge clk);
56 tx_start = 1;
57 @(posedge clk);
58 tx_start = 0;
59
60 wait (tx_done);
61 $display("Finished sending 0xA3 at time %0t", $time);
62
63 #200;
64 $finish;
65
66 end
endmodule

```

RTL RESULTS FOR TX :-



DESIGN CODE FOR RX :-

```
2 module uart_rx (
3     input wire      clk,
4     input wire      rst,
5     input wire [31:0] baud_div,
6     input wire      rx_pin,
7     output reg [7:0] rx_data,
8     output reg      rx_done,
9     output reg      rx_busy,
10    output reg      rx_error
11 );
12
13 // states
14 localparam R_IDLE   = 3'd0;
15 localparam R_START  = 3'd1;
16 localparam R_DATA   = 3'd2;
17 localparam R_STOP   = 3'd3;
18 localparam R_DONE   = 3'd4;
19
20 reg [2:0] state, nxt_state;
21 reg [31:0] baud_cnt;
22 reg [2:0] bit_cnt;
23 reg [7:0] shift_reg;
24
25 // synchronizer for rx_pin (2-flop)
26 reg rx_sync1, rx_sync2;
27 always @(posedge clk) begin
28     if (rst) begin
29         rx_sync1 <= 1'b1;
30         rx_sync2 <= 1'b1;
31     end else begin
32         rx_sync1 <= rx_pin;
33         rx_sync2 <= rx_sync1;
34     end
35 end
36 wire rx_synced = rx_sync2;
37
38 // sequential FSM
39 always @(posedge clk) begin
40     if (rst) begin
41         state <= R_IDLE;
42         baud_cnt <= 32'd0;
43         bit_cnt <= 3'd0;
44         shift_reg <= 8'd0;
45         rx_data <= 8'd0;
46         rx_done <= 1'b0;
47         rx_busy <= 1'b0;
48         rx_error <= 1'b0;
49     end else begin
50         state <= nxt_state;
51
52         // Default outputs
53         rx_done <= 1'b0;
54
55         case (state)
56             R_IDLE: begin
57                 rx_busy <= 1'b0;
58                 rx_error <= 1'b0;
59                 if (rx_synced == 1'b0) begin // Start bit detected
60                     baud_cnt <= (baud_div >> 1) - 1; // Sample at middle of start bit
61                     rx_busy <= 1'b1;
62                 end
63             end
64
65             R_START: begin
66                 if (baud_cnt == 32'd0) begin
67                     // Verify start bit is still low
68                     if (rx_synced == 1'b0) begin
69                         baud_cnt <= baud_div - 1; // Full bit period for data bits
70                         bit_cnt <= 3'd0;
71                     end
72                 end else begin
73                     baud_cnt <= baud_cnt - 1;
74                 end
75             end
76
77             R_DATA: begin
```



```

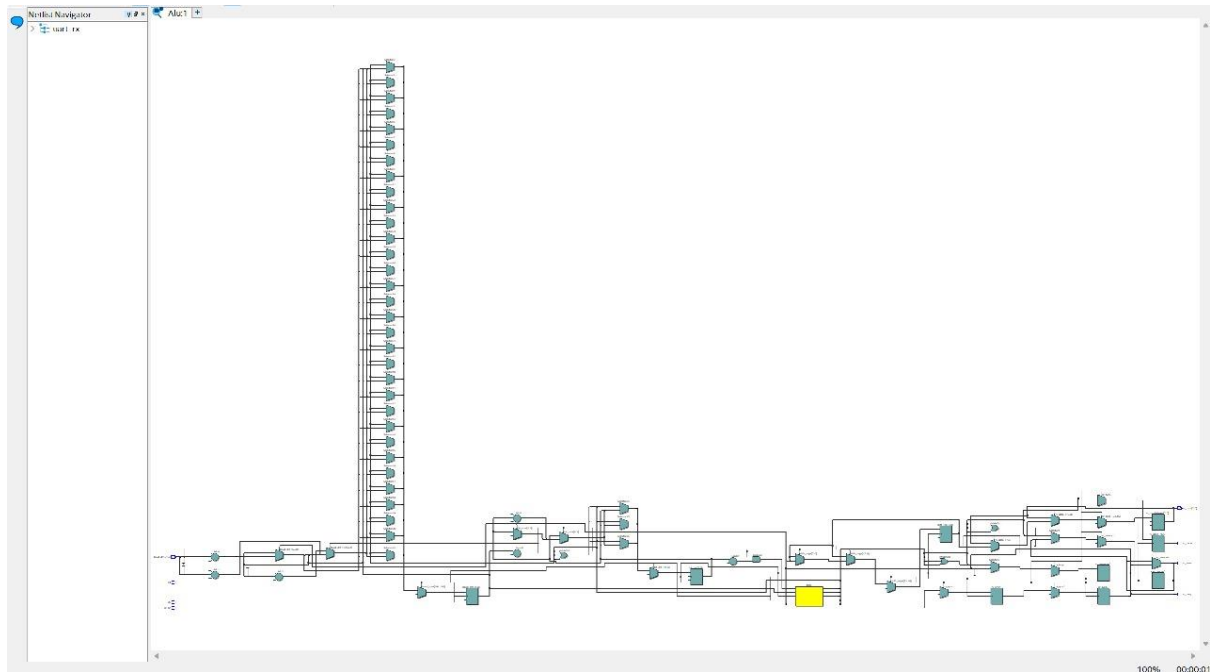
78         if (baud_cnt == 32'd0) begin
79             // Shift in LSB first
80             shift_reg <= {rx_synced, shift_reg[7:1]};
81             bit_cnt <= bit_cnt + 1;
82             baud_cnt <= baud_div - 1;
83         end else begin
84             baud_cnt <= baud_cnt - 1;
85         end
86     end
87
88     R_STOP: begin
89         if (baud_cnt == 32'd0) begin
90             // Check stop bit
91             if (rx_synced == 1'b1) begin
92                 rx_data <= shift_reg;
93                 rx_error <= 1'b0;
94             end else begin
95                 rx_data <= shift_reg;
96                 rx_error <= 1'b1; // Framing error
97             end
98         end else begin
99             baud_cnt <= baud_cnt - 1;
100         end
101     end
102
103     R_DONE: begin
104         rx_done <= 1'b1;
105         rx_busy <= 1'b0;
106     end
107 endcase
108 end
109
110 // next-state combinational logic
111 always @(*) begin
112     nxt_state = state;
113     case (state)
114         R_IDLE: begin
115             if (rx_synced == 1'b0) // Start bit detected
116                 nxt_state = R_START;
117         end
118
119         R_START: begin
120             if (baud_cnt == 32'd0) begin
121                 if (rx_synced == 1'b0) // Valid start bit
122                     nxt_state = R_DATA;
123                 else // False start
124                     nxt_state = R_IDLE;
125             end
126         end
127
128         R_DATA: begin
129             if ((baud_cnt == 32'd0) && (bit_cnt == 3'd7))
130                 nxt_state = R_STOP;
131         end
132
133         R_STOP: begin
134             if (baud_cnt == 32'd0)
135                 nxt_state = R_DONE;
136         end
137
138         R_DONE: begin
139             nxt_state = R_IDLE;
140         end
141
142         default: nxt_state = R_IDLE;
143     endcase
144 end
145 endmodule
146
147
148

```


TEST BUNCH FOR RX :-

```
2 module tb_uart_rx;
3     reg clk = 0;
4     always #5 clk = ~clk; // 100MHz
5
6     reg rst;
7     reg [31:0] baud_div;
8     reg rx_pin;
9     wire [7:0] rx_data;
10    wire rx_done;
11    wire rx_busy;
12    wire rx_error;
13
14    uart_rx uut (
15        .clk(clk),
16        .rst(rst),
17        .baud_div(baud_div),
18        .rx_pin(rx_pin),
19        .rx_data(rx_data),
20        .rx_done(rx_done),
21        .rx_busy(rx_busy),
22        .rx_error(rx_error)
23    );
24
25    // helper task: send serial byte LSB-first with start/stop
26    task send_byte;
27        input [7:0] b;
28        integer i;
29        reg [31:0] bit_ticks;
30        begin
31            bit_ticks = baud_div;
32            // start bit
33            rx_pin = 0;
34            repeat(bit_ticks) @(posedge clk);
35            // data bits LSB first
36            for (i=0;i<8;i=i+1) begin
37                rx_pin = b[i];
38                repeat(bit_ticks) @(posedge clk);
39            end
40            // stop bit
41            rx_pin = 1;
42            repeat(bit_ticks) @(posedge clk);
43        end
44    endtask
45
46    initial begin
47        $dumpfile("tb_uart_rx.vcd");
48        $dumpvars(0,tb_uart_rx);
49        rst = 1; rx_pin = 1; baud_div = 32'd868;
50        #20;
51        rst = 0;
52        #20;
53
54        // send 0xA5
55        send_byte(8'hA5);
56
57        // wait for rx_done
58        wait(rx_done);
59        $display("RX got %02h at time %0t (err=%0b)", rx_data, $time, rx_error);
60
61        #200;
62        $finish;
63    end
64 endmodule
65
```


RTL FOR RX :-



DESIGN CODE FOR APB :-

```

11 module apb_uart(
12     input wire      pclk,
13     input wire      presetn,    // active-low reset (APB style)
14     input wire [31:0] paddr,
15     input wire      psel,
16     input wire      penable,
17     input wire      pwrite,
18     input wire [31:0] pwrdata,
19     output reg [31:0] prdata,
20     output reg      pready,
21     // to UART modules
22     output wire [31:0] baud_div,
23     output reg      tx_start,    // single-cycle pulse
24     output reg [7:0] tx_data,
25     input wire      tx_busy,
26     input wire      tx_done,
27     output reg      rx_reset,    // optional resets to rx/t x
28     input wire [7:0] rx_data,
29     input wire      rx_done,
30     input wire      rx_busy,
31     input wire      rx_error
32 );
33
34 // internal reset active-high for our logic
35 wire rst = ~presetn;
36
37 // registers
38 reg [31:0] ctrl_reg;
39 reg [31:0] status_reg;
40 reg [31:0] tx_data_reg;
41 reg [31:0] rx_data_reg;
42 reg [31:0] bauddiv_reg;
43
44 // address mapping (word-aligned addresses)
45 localparam ADDR_CTRL = 32'h0000;
46 localparam ADDR_STATUS = 32'h0001;
47 localparam ADDR_TX = 32'h0002;
48 localparam ADDR_RX = 32'h0003;
49 localparam ADDR_BAUD = 32'h0004;
50
51
52 always @(posedge pclk) begin
53     if (rst) begin
54         pready <= 1'b0;
55         prdata <= 32'd0;
56         ctrl_reg <= 32'd0;
57         status_reg <= 32'd0;
58         tx_data_reg <= 32'd0;
59         rx_data_reg <= 32'd0;
60         bauddiv_reg <= 32'd868; // default 115200 @100MHz approx
61         tx_start <= 1'b0;
62         tx_data <= 8'd0;
63         rx_reset <= 1'b0;
64     end else begin
65         // default outputs
66         pready <= 1'b0;
67         tx_start <= 1'b0;
68         rx_reset <= 1'b0;
69
70         // update status_reg from UART live signals (bits placed accordingly)
71         status_reg[0] <= rx_busy;
72         status_reg[1] <= tx_busy;
73         status_reg[2] <= rx_done;
74         status_reg[3] <= tx_done;
75         status_reg[4] <= rx_error;
76
77         if (psel & penable) begin
78             pready <= 1'b1; // we complete transaction in this cycle
79             if (pwrite) begin
80                 // write transaction
81                 case (paddr)
82                     ADDR_CTRL: begin
83                         ctrl_reg <= pwrdata;
84                         // if tx_en bit set => generate tx_start pulse & load tx_data_reg to tx_data
85                         if (pwrdata[0]) begin
86                             tx_data <= tx_data_reg[7:0]; // assume previously written
87                             tx_start <= 1'b1;
88                         end
89                         // rx_rst or tx_rst handling (bits 2 & 3)
90                         if (pwrdata[2]) begin
91
92                             end
93                         if (pwrdata[3]) begin
94                             rx_reset <= 1'b1; // pulse reset to RX if desired
95                         end
96                     end
97                 end
89 end
90
91
92
93
94
95
96

```

```

97
98
99         ADDR_TX: begin
100             tx_data_reg <= pwrite;
101
102         end
103
104         ADDR_BAUD: begin
105             bauddiv_reg <= pwrite;
106         end
107
108         default: begin
109             // ignore
110         end
111     endcase
112 end else begin
113     // read transaction
114     case (paddr)
115         ADDR_CTRL: prdata <= ctrl_reg;
116         ADDR_STATUS: prdata <= status_reg;
117         ADDR_TX: prdata <= tx_data_reg;
118         ADDR_RX: prdata <= [24'd0, rx_data]; // low byte = rx_data
119         ADDR_BAUD: prdata <= bauddiv_reg;
120         default: prdata <= 32'hDEAD_BEEF;
121     endcase
122 end
123
124 end
125
126 assign baud_div = bauddiv_reg;
127
128 endmodule
129

```

TEST BUNCH FOR APB :-

```

2 module tb_apb_uart;
3     reg pclk = 0;
4     always #5 pclk = ~pclk; // 100MHz
5
6     reg presetn;
7
8     reg [31:0] paddr;
9     reg psel;
10    reg penable;
11    reg pwrite;
12    reg [31:0] pwrite;
13    wire [31:0] prdata;
14    wire pready;
15
16    // regs for status and rx
17    reg [31:0] s;
18    reg [31:0] rxd;
19
20    // wires to UART modules
21    wire [31:0] baud_div;
22    wire tx_start;
23    wire [7:0] tx_data;
24    wire tx_busy;
25    wire tx_done;
26    wire [7:0] rx_data;
27    wire rx_done;
28    wire rx_busy;
29    wire rx_error;
30
31    // instantiate apb_uart
32    apb_uart uut_apb (
33        .pclk(pclk),
34        .presetn(presetn),
35        .paddr(paddr),
36        .psel(psel),
37        .penable(penable),
38        .pwrite(pwrite),
39        .pwrite(pwrite),
40        .prdata(prdata),
41        .pready(pready),
42        .baud_div(baud_div),
43        .tx_start(tx_start),
44        .tx_data(tx_data),
45        .tx_busy(tx_busy),
46        .tx_done(tx_done).

```

```

47     .rx_reset(),
48     .rx_data(rx_data),
49     .rx_done(rx_done),
50     .rx_busy(rx_busy),
51     .rx_error(rx_error)
52 );
53
54 // loopback
55 wire tx_line;
56 uart_tx UTX (
57     .clk(pclk),
58     .rst(~presetn),
59     .baud_div(baud_div),
60     .tx_start(tx_start),
61     .tx_data(tx_data),
62     .tx(tx_line),
63     .tx_busy(tx_busy),
64     .tx_done(tx_done)
65 );
66
67 uart_rx URX (
68     .clk(pclk),
69     .rst(~presetn),
70     .baud_div(baud_div),
71     .rx_pin(tx_line),
72     .rx_data(rx_data),
73     .rx_done(rx_done),
74     .rx_busy(rx_busy),
75     .rx_error(rx_error)
76 );
77
78 // APB tasks
79 task apb_write;
80     input [31:0] addr;
81     input [31:0] data;
82     begin
83         @(posedge pclk);
84         paddr <= addr;
85         pwrite <= data;
86         pwrite <= 1;
87         psel <= 1;
88         penable <= 1;
89         @(posedge pclk);
90         while (!pready) @(posedge pclk);
91         usel <= 0;
92
93         penable <= 0;
94         pwrite <= 0;
95         @(posedge pclk);
96     end
97 endtask
98
99 task apb_read;
100     input [31:0] addr;
101     output [31:0] data;
102     begin
103         @(posedge pclk);
104         paddr <= addr;
105         pwrite <= 0;
106         psel <= 1;
107         penable <= 1;
108         @(posedge pclk);
109         while (!pready) @(posedge pclk);
110         data = prdata;
111         psel <= 0;
112         penable <= 0;
113         @(posedge pclk);
114     end
115 endtask
116
117 initial begin
118     $dumpfile("tb_apb_uart.vcd");
119     $dumpvars(0,tb_apb_uart);
120     presetn = 0;
121     psel = 0; penable = 0; pwrite = 0; paddr = 0; pwdata = 0;
122     #40;
123     presetn = 1;
124     #40;
125
126     // write TX_DATA = 0x5A
127     apb_write(32'h0002, 32'h0000005A);
128
129     // pulse CTRL.tx_en = bit0
130     apb_write(32'h0000, 32'h1);
131
132     // poll STATUS until tx_done
133     repeat (2000) begin
134         apb_read(32'h0001, s);
135         if (s[3]) begin
136             $display("APB saw tx_done at time %0t", $time);
137             disable poll loop;
138         end
139         #100;
140     end
141     poll_loop ;
142
143     // read RX_DATA
144     apb_read(32'h0003, rxd);
145     $display("APB read RX_DATA = %02h", rxd[7:0]);
146     #200;
147     $finish;
148 end
149 endmodule

```

RTL RESULTS FOR APB :-

