APB Master:

```
module apb_add_master (
  input logic                 pclk,
  input logic                 preset_n,      // Active low reset

  inputlogic[1:0]             add_i,
// 2'b00 - NOP, 2'b01 - READ, 2'b11 - WRITE

  output logic                      psel_o,
  output logic                      penable_o,
  output logic [31:0]         paddr_o,
  output logic                      pwrite_o,
  output logic [31:0]         pwdata_o,
  input  logic [31:0]         prdata_i,
  input  logic                pready_i

);

  typedef enum logic[1:0] {ST_IDLE, ST_SETUP, ST_ACCESS}
apb_state_t;

  apb_state_t state_q;              // Current state
  apb_state_t nxt_state;            // Next state

  logic apb_state_setup;
  logic apb_state_access;

  logic nxt_pwrite;
  logic pwrite_q;

  logic [31:0] nxt_rdata;
  logic [31:0] rdata_q;

  always_ff @(posedge pclk or negedge preset_n)
    if (~preset_n)
      state_q <= ST_IDLE;
          else
      state_q <= nxt_state;

  always_comb begin
    nxt_pwrite = pwrite_q;
    nxt_rdata = rdata_q;
    case (state_q)
```

```
    ST_IDLE:
      if (add_i[0]) begin     //01 read, 11 write
        nxt_state = ST_SETUP;
        nxt_pwrite = add_i[1];
      end else begin
        nxt_state = ST_IDLE;
      end
    ST_SETUP: nxt_state = ST_ACCESS;
    ST_ACCESS:
      if (pready_i) begin
        if (~pwrite_q)
          nxt_rdata = prdata_i;
        nxt_state = ST_IDLE;
      end else
        nxt_state = ST_ACCESS;
    default: nxt_state = ST_IDLE;
  endcase
end

assign apb_state_access = (state_q == ST_ACCESS);
assign apb_state_setup = (state_q == ST_SETUP);

assign psel_o = apb_state_setup | apb_state_access;
assign penable_o = apb_state_access;

// APB Address
assign paddr_o = {32{apb_state_access}} & 32'hA000;

// APB PWRITE control signal
always_ff @(posedge pclk or negedge preset_n)
  if (~preset_n)
    pwrite_q <= 1'b0;
        else
    pwrite_q <= nxt_pwrite;

assign pwrite_o = pwrite_q;

// APB PWDATA data signal
// ADDER
// Read a value from the slave at address 0xA000
// Increment that value
// Send that value back during the write operation to address 0xA000
```

```
assign pwdata_o = {32{apb_state_access}} & (rdata_q + 32'h1);


  always_ff @(posedge pclk or negedge preset_n)
    if (~preset_n)
      rdata_q <= 32'h0;
         else
      rdata_q <= nxt_rdata;

Endmodule
```

APB Slave:

```
`define CLK @(posedge pclk)

module apb_slave_tb ();

        logic                   pclk;
        logic                   preset_n;      // Active low reset

  logic[1:0]            add_i;          // 2'b00 - NOP, 2'b01 - READ, 2'b11 -
WRITE

        logic                   psel_o;
        logic                   penable_o;
        logic [31:0]  paddr_o;
        logic                   pwrite_o;
        logic [31:0]  pwdata_o;
        logic [31:0]  prdata_i;
        logic                   pready_i;

  // Implement clock
  always begin
    pclk = 1'b0;
    #5;
    pclk = 1'b1;
    #5;
  end

  // Instantiate the RTL
  apb_add_master APB_MASTER (.*);
```

```
// Drive stimulus
initial begin
  preset_n = 1'b0;
  add_i = 2'b00;
  repeat (2) `CLK;
  preset_n = 1'b1;
  repeat (2) `CLK;

  // Initiate a read transaction
  add_i = 2'b01;
  `CLK;
  add_i = 2'b00;
  repeat (4) `CLK;

  // Initiate a write transaction
  add_i = 2'b11;
  `CLK;
  add_i = 2'b00;
  repeat (4) `CLK;
  $finish();
end

// APB Slave
always_ff @(posedge pclk or negedge preset_n) begin
  if (~preset_n)
    pready_i <= 1'b0;
  else begin
  if (psel_o && penable_o) begin
    pready_i <= 1'b1;
    prdata_i <= $urandom%32'h20;
  end else begin
    pready_i <= 1'b0;
    prdata_i <= $urandom%32'hFF;
  end
  end
end

// VCD Dump
initial begin
  $dumpfile("dump.vcd");
  $dumpvars(1);
end
endmodule
```