```
//*****************************************************************************
//                              INTERFACE UNIT DRIVER
//*****************************************************************************
/*****************************************************************************

Driver for the INTERFACE module
file:       interface_drv.v
authors:    Bernardo Mota and Antonio J. de Parga
creation:   25.04.01
last update 04.12.01
version:    1.0
*****************************************************************************/

`timescale 1ns/1ps
`include "interface_rtl.v"


/* This alias module is for use internal to the netlister only.
   Please do not use the same name for modules or
   assume the existence of this module. */

module cds_alias( cds_alias_sig, cds_alias_sig);

parameter width = 1;
input [width:1] cds_alias_sig;

endmodule


module sim();

//inputs assynchronous

reg         rstb,
            write,
            l2y,
            trg;

//inputs with clk
reg         clk;

//inputs with clk2
reg         clk2,
            cstb;


//Bidirectional Bus
wire  [39:0]      bd;
reg   [39:0]      io_bd;


//outputs with clk
wire        tstoutb;



//outputs with clk2
```

```
wire        ack, ackb,
            trsf, trsfb,
            ack_en, ackb_en,
            dolob_en,
            trsf_en, trsfb_en,
            dstb,
            errorb,
            writeb;


// Inout assignments
cds_alias #40 cds_alias_inst1(bd, io_bd);

initial
   begin
     $shm_open("simulation.shm");
     $shm_probe(sim,"AS");
     #900000 $stop;
     $shm_close;
   end


altro a1(clk, clk2, rstb, trg, l2y, cstb, writeb, ackb, ackb_en, dolob_en,
trsfb, trsfb_en, dstb, errorb, tstoutb, bd);


integer j;
integer dummy;


assign writeb = !write;
assign ack = !ackb;
assign ack_en = !ackb_en;
assign trsf = !trsfb;
assign trsf_en = !trsfb_en;


// Default verilog stimulus.

initial
begin

   io_bd[39:0] = 40'b0000000000000000000000000000000000000000;

   clk = 1'b0;
   clk2 = 1'b0;
   cstb = 1'b0;

   rstb = 1'b0;
   trg = 1'b0;
   l2y = 1'b0;
   write = 1'b0;

end

parameter T1 = 100;     // Period of ADC clock
parameter T2 = 24;      // Period of Bus clock
```

```
parameter dchip = 2;    // Output delay of ALTRO
parameter drcu = 4;     // Output delay of RCU
parameter ack_to = 20;  // ACK timeout in clk2 cycles


task Sync_CK;           // Syncronize with clock 1
 begin
  if (clk) wait(!clk);
  wait(clk);
 end
endtask


task Sync_CK2;          // Syncronize with clock 2
 begin
  if (clk2) wait(!clk2);
  wait(clk2);
 end
endtask


task Wait_ACK;          // Wait for acknowledgement
integer ack_r;
 begin
  if (ack) wait(~ack);
  ack_r = ack_to;
  while ((ack_r>0) && !ack)
   begin
    Sync_CK2;
    ack_r = ack_r -1;
   end
 end
endtask



task Parity;
 begin
      io_bd[39] = io_bd[38] ^ io_bd[37] ^ io_bd[36] ^ io_bd[35] ^ io_bd[34] ^
io_bd[33] ^ io_bd[32] ^
      io_bd[31] ^ io_bd[30] ^ io_bd[29] ^ io_bd[28] ^ io_bd[27] ^ io_bd[26] ^
io_bd[25] ^
      io_bd[24] ^ io_bd[23] ^ io_bd[22] ^ io_bd[21] ^ io_bd[20];
 end
endtask


task Wait_Transfer;           // Wait for Transfer
integer trs_r;
 begin
  if (trsf_en) wait(~trsf_en);
  trs_r = ack_to;
  while ((trs_r>0) && !trsf_en)
   begin
    Sync_CK2;
    trs_r = trs_r -1;
   end
```

```
  wait(!trsf_en);
 end
endtask



task Write_CSR;
  input [4:0]  csr;
  input [3:0]  chadd;
  input [19:0] data;
 begin
      wait(~ack_en);
      wait(~trsf_en);               // Check that the bus is free
      Sync_CK2;
   #drcu;
      io_bd[38:37] = 2'b00;
      io_bd[36:29] = 1;             // Chip Address
      io_bd[28:25] = 0;             // No channel address
      io_bd[24:20] = csr;                 // Register Addr / Command
      io_bd[19:0] = data;                 // Data
      Parity;
      write = 1;
      cstb = 1;
      Wait_ACK;
      if (a1.i1.i2.instr_err)
         $display("error caused by instruction %x at time %d",csr,$stime);
      Sync_CK2;
   #drcu;
      cstb = 0;
      write = 0;
      io_bd = 40'hzzzzzzzzzz;
 end
endtask


task Read_CSR;
  input  [4:0]  csr;
  input  [3:0]  chadd;
  output [19:0] data;
 begin
      wait(~ack_en);
      wait(~trsf_en);
      Sync_CK2;
   #drcu;
      io_bd[38:37] = 2'b00;
      io_bd[36:29] = 1;             // Chip Address
      io_bd[28:25] = 0;             // No channel address
      io_bd[24:20] = csr;                 // Register Addr / Command
      Parity;
      write = 0;
      cstb = 1;
      Wait_ACK;
      if (a1.i1.i2.instr_err)
         $display("error caused by instruction %x at time %d",csr,$stime);
      Sync_CK2;
    #drcu;
      data = io_bd[19:0];
```

```
      cstb = 0;
      io_bd = 40'hzzzzzzzzzz;
 end
endtask


always #(T1*0.5) clk = ~clk;  // Generate Sampling Clock
always #(T2*0.5) clk2 = ~clk2;      // Generate Bus Clock



//--------- Read, Write Registers and send all possible commands including
addressing inexistent instructions-------//

initial
begin

 #T2;
 #drcu;
      rstb = 0;
 #50  rstb = 1;


 #1000;

       for (j=0; j<26; j=j+1)
        begin
         Write_CSR(j,0,5*j);
        end

      //-----Readout Command------//
            Write_CSR(26,0,1);
            Wait_Transfer;
      //-------------------------//

       for (j=27; j<32; j=j+1)
        begin
         Write_CSR(j,0,5*j);
        end

 #1000;

       for (j=0; j<32; j=j+1)
        begin
         Read_CSR(j,0,dummy);
        end
 #2000 $finish;

end


endmodule
```