```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity plb2bus is
    generic
    (
        C_SPLB_BASEADDR : std_logic_vector := x"FFFFFFFF";
        C_SPLB_HIGHADDR : std_logic_vector := x"00000000";
        C_SPLB_MID_WIDTH : natural := 1;
        C_SPLB_NUM_MASTERS : natural := 1;
        C_AWIDTH : integer := 32;
        C_DWIDTH : integer := 32;
        C_FIFO_DEPTH : natural := 1
    );
    port (
        splb_clk : in std_logic;
        splb_rst : in std_logic;
        plb_abus : in std_logic_vector(0 to C_AWIDTH-1);
        plb_uabus : in std_logic_vector(0 to C_AWIDTH-1);
        plb_pavalid : in std_logic;
        plb_savalid : in std_logic;
        plb_rdprim : in std_logic;
        plb_wrprim : in std_logic;
        plb_masterid : in std_logic_vector(0 to C_SPLB_MID_WIDTH-1);
        plb_abort : in std_logic;
        plb_buslock : in std_logic;
        plb_rnw : in std_logic;
        plb_be : in std_logic_vector(0 to 3);
        plb_msize : in std_logic_vector(0 to 1);
        plb_size : in std_logic_vector(0 to 3);
        plb_type : in std_logic_vector(0 to 2);
        plb_lockerr : in std_logic;
        plb_wrdbus : in std_logic_vector(0 to C_DWIDTH-1);
        plb_wrburst : in std_logic;
        plb_rdburst : in std_logic;
        plb_wrpendreq : in std_logic;
        plb_rdpendreq : in std_logic;
        plb_wrpendpri : in std_logic_vector(0 to 1);
        plb_rdpendpri : in std_logic_vector(0 to 1);
        plb_reqpri : in std_logic_vector(0 to 1);
        plb_tattribute : in std_logic_vector(0 to 15);
        sl_addrack : out std_logic;
        sl_ssize : out std_logic_vector(0 to 1);
        sl_wait : out std_logic;
        sl_rearbitrate : out std_logic;
        sl_wrdack : out std_logic;
        sl_wrcomp : out std_logic;
        sl_wrbterm : out std_logic;
        sl_rddbus : out std_logic_vector(0 to C_DWIDTH-1);
        sl_rdwdaddr : out std_logic_vector(0 to 3);
        sl_rddack : out std_logic;
        sl_rdcomp : out std_logic;
        sl_rdbterm : out std_logic;
        sl_mbusy : out std_logic_vector(0 to C_SPLB_NUM_MASTERS-1);
        sl_mwrerr : out std_logic_vector(0 to C_SPLB_NUM_MASTERS-1);
        sl_mrderr : out std_logic_vector(0 to C_SPLB_NUM_MASTERS-1);
        sl_mirq : out std_logic_vector(0 to C_SPLB_NUM_MASTERS-1);

        request_out : out std_logic;
        endreq_out : out std_logic;
        data_out : out std_logic_vector(C_DWIDTH-1 downto 0);
        valid_out : out std_logic;
        accept_out : in std_logic;

        data_in : in std_logic_vector(C_DWIDTH-1 downto 0);
        valid_in : in std_logic;
```

```
        accept_in : out std_logic
    );
end plb2bus;

architecture rtl of plb2bus is
    constant SIZE_BITS : natural := 5; -- equals dtl_cmd_block_size'length in busmst.vhd

    function swapix(v:std_logic_vector) return std_logic_vector is
        variable s : std_logic_vector(v'reverse_range);
    begin
        for i in 0 to v'length-1 loop s(v'high-i) := v(v'low+i); end loop;
        return s;
    end swapix;

    function addrbits(base, high : std_logic_vector) return natural is
    begin
        for i in base'range loop
            if base(i)/=high(i) then
                if base'ascending then
                    return i;
                else
                    return base'left-i;
                end if;
            end if;
        end loop;
        return base'length;
    end addrbits;

    constant ADDR_MATCH_BITS : natural := addrbits(C_SPLB_BASEADDR,C_SPLB_HIGHADDR);

    signal clk, rstn, plb_pavalid_i : std_logic;
    signal cmd_word, addr_word, data_word : std_logic_vector(data_out'range);
    signal size : natural range 0 to 2**SIZE_BITS-1;
    signal mask : std_logic_vector(plb_be'length-1 downto 0);

    type state_t is (NOP,CMD,ADDR,WRITE,WRITE_LAST,READ,READ_LAST);
    type reg_t is record
        state : state_t;
        addr : std_logic_vector(plb_abus'range);
        size : natural range 0 to 2**SIZE_BITS-1;
        rnw : std_logic;
        mask : std_logic_vector(mask'range);
        beat : natural range 0 to 15;
        mid : std_logic_vector(plb_masterid'range);
    end record;
    signal r, r_in : reg_t;
begin

    clk <= splb_clk;
    rstn <= not splb_rst;

    process(clk)
    begin
        if rising_edge(clk) then
            if rstn/='1' then
                r.state <= NOP;
            else
                r <= r_in;
            end if;
        end if;
    end process;


    process(r,plb_abus,size,plb_rnw,mask,plb_masterid,plb_pavalid_i,plb_abort,accept_out,valid
    _in)
        variable v : reg_t;
    begin
```

```
    v := r;

    case r.state is
    when NOP =>
        v.addr := plb_abus;
        v.size := size;
        v.rnw := plb_rnw;
        v.mask := mask;
        v.beat := 0;
        v.mid := plb_masterid;
        if plb_pavalid_i='1' and plb_abort='0' then
            v.state := CMD;
        end if;
    when CMD =>
        if accept_out='1' then
            v.state := ADDR;
        end if;
    when ADDR =>
        if accept_out='1' then
            if r.rnw='1' then
                if r.size=0 then
                    v.state := READ_LAST;
                else
                    v.state := READ;
                end if;
            else
                if r.size=0 then
                    v.state := WRITE_LAST;
                else
                    v.state := WRITE;
                end if;
            end if;
        end if;
    when WRITE =>
        if accept_out='1' then
            v.beat := r.beat+1;
            if v.beat=r.size then
                v.state := WRITE_LAST;
            end if;
        end if;
    when WRITE_LAST =>
        if accept_out='1' then
            v.state := NOP;
        end if;
    when READ =>
        if valid_in='1' then
            v.beat := r.beat+1;
            if v.beat=r.size then
                v.state := READ_LAST;
            end if;
        end if;
    when READ_LAST =>
        if valid_in='1' then
            v.state := NOP;
        end if;
    end case;

    r_in <= v;
end process;

plb_pavalid_i <=
    plb_pavalid when plb_abus(0 to ADDR_MATCH_BITS-1)=C_SPLB_BASEADDR(0 to
    ADDR_MATCH_BITS-1) else
    '0';

cmd_word <=
    r.rnw &
```

```
    (C_DWIDTH-1-r.mask'length-SIZE_BITS-1 downto 0=>'0') &
    r.mask &
    std_logic_vector(to_unsigned(r.size,SIZE_BITS));
size <=
    0 when plb_size="0000" else
    3 when plb_size="0001" else
    7 when plb_size="0010" else
    15 when plb_size="0011" else
    to_integer(unsigned(plb_be)) when plb_size="1010" else
    0;
mask <=
    (others=>'0') when plb_size="1010" else
    not swapix(plb_be);
addr_word <= swapix(r.addr(0 to C_AWIDTH-3)) & "00";
data_word <= swapix(plb_wrdbus);

with r.state select
    sl_addrack <=
        plb_pavalid_i when NOP,
        '0' when others;
sl_ssize <= "00";
sl_wait <= plb_pavalid_i;
sl_rearbitrate <= '0';
with r.state select
    sl_wrdack <= accept_out when WRITE | WRITE_LAST, '0' when others;
with r.state select
    sl_wrcomp <= accept_out when WRITE_LAST, '0' when others;
sl_wrbterm <= '0';
with r.state select
    sl_rddbus <= swapix(data_in) when READ | READ_LAST, (others=>'0') when others;
sl_rdwdaddr <= std_logic_vector(to_unsigned(r.beat,sl_rdwdaddr'length));
sl_rddack <= valid_in;
with r.state select
    sl_rdcomp <= valid_in when READ_LAST, '0' when others;
sl_rdbterm <= '0';

process(r)
begin
    sl_mbusy <= (others=>'0');
    if r.state /= NOP then
        for i in 0 to 2**C_SPLB_MID_WIDTH-1 loop
            if i=to_integer(unsigned(r.mid)) then
                sl_mbusy(i) <= '1';
            end if;
        end loop;
    end if;
end process;

sl_mwrerr <= (others=>'0');
sl_mrderr <= (others=>'0');
sl_mirq <= (others=>'0');

with r.state select
    request_out <= '1' when CMD | ADDR | WRITE | WRITE_LAST, '0' when others;
with r.state select
    endreq_out <=
        r.rnw when ADDR,
        '1' when WRITE_LAST,
        '0' when others;
with r.state select
    data_out <=
        cmd_word when CMD,
        addr_word when ADDR,
        data_word when WRITE | WRITE_LAST,
        (others=>'0') when others;
with r.state select
    valid_out <=
```

```
            '1' when CMD | ADDR | WRITE | WRITE_LAST,
            '0' when others;
    with r.state select
        accept_in <=
            '1' when READ | READ_LAST,
            '0' when others;

--pragma translate_off
    assert C_DWIDTH=32 report "Unsupported data bus width" severity error;
    assert C_DWIDTH>=1+C_DWIDTH/8+SIZE_BITS report "Packet width to small for command
    parameters" severity error;
    assert not (rising_edge(clk) and plb_pavalid_i='1' and
        plb_size/="0000" and plb_size/="0001" and plb_size/="0010" and plb_size/="0011" and
        plb_size/="1010")
        report "Unsupported PLB transfer size" severity error;
--pragma translate_on
end rtl;
```