Altera’s Avalon Communication Fabric

Prof. Stephen A. Edwards
sedwards@cs.columbia.edu
Columbia University
Spring 2007

Altera’s Avalon Bus

Something like “PCI on a chip”
Described in Altera’s Avalon Memory-Mapped Interface Specification document.
Protocol defined between peripherals and the “bus” (actually a fairly complicated circuit).

Masters and Slaves

Most bus protocols draw a distinction between
Masters: Can initiate a transaction, specify an address, etc. E.g., the Nios II processor
Slaves: Respond to requests from masters, can generate return data. E.g., a video controller
Most peripherals are slaves.
Masters speak a more complex protocol
Bus arbiter decides which master gains control

The Simplest Slave Peripheral

Avalon-MM Interface
(Avalon-MM Slave Port)
Application-Specific Interface
write[15..0] writedata[15..0]

Basically, “latch when I’m selected and written to.”

Slave Signals

For a 16-bit connection that spans 32 halfwords,

Avalon

Avalon Slave Signals

clk | Master clock
reset | Reset signal to peripheral
chipselect | Asserted when bus accesses peripheral
address[...] | Word address (data-width specific)
read | Asserted during peripheral—bus transfer
write | Asserted during bus—peripheral transfer
writedata[...] | Data from bus to peripheral
byteenable[...] | Indicates active bytes in a transfer
readdata[...] | Data from peripheral to bus
irq | peripheral—processor interrupt request

All are optional, as are many others for, e.g., flow-control and burst transfers.

Naming Conventions

Used by the SOPC Builder’s New Component Wizard to match up VHDL entity ports with Avalon bus signals.
type_interface__signal
type is typically avs for Avalon-MM Slave
interface is the user-selected name of the interface, e.g., s1.
signal is chipselect, address, etc.
Thus, avs_s1_chipselect is the chip select signal for a slave port called “s1.”

Bytes, Bits, and Words

The Nios II and Avalon bus are little-endian:
31 is the most significant bit, 0 is the least
Bytes and halfwords are right-justified:

<table>
<thead>
<tr>
<th>Byte</th>
<th>3</th>
<th>2</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bit</td>
<td>31</td>
<td>24</td>
<td>16</td>
<td>8</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Word</th>
<th>31</th>
</tr>
</thead>
<tbody>
<tr>
<td>Halfword</td>
<td>15</td>
</tr>
<tr>
<td>Byte</td>
<td>7</td>
</tr>
</tbody>
</table>
In VHDL

Entity Declaration

library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

entity led_flasher is
  port ( 
    avs_s1_clk : in std_logic;
    avs_s1_reset_n : in std_logic;
    avs_s1_read : in std_logic;
    avs_s1_write : in std_logic;
    avs_s1_chipselect : in std_logic;
    avs_s1_address : in std_logic_vector(4 downto 0);
    avs_s1_readdata : out std_logic_vector(15 downto 0);
    avs_s1_writedata : in std_logic_vector(15 downto 0);
  );
end led_flasher;

Architecture (1)

architecture rtl of led_flasher is
  signal clk, reset_n : std_logic;
  type ram_type is array(15 downto 0) of std_logic_vector(15 downto 0);
  signal ram : ram_type;
  signal ram_address, display_address : std_logic_vector(3 downto 0);
  signal counter_delay : std_logic_vector(15 downto 0);
  signal counter : std_logic_vector(31 downto 0);
  begin
    clk <= avs_s1_clk;
    reset_n <= avs_s1_reset_n;
    ram_address <= avs_s1_address(3 downto 0);
    leds : out std_logic_vector(15 downto 0) <= (others => '0');
  end rtl;

Architecture (2)

process (clk)
begin
  if clk'event and clk = '1' then
    if reset_n = '0' then
      ram_address <= (others => '0');
      display_address <= (others => '0');
      counter <= (others => '0');
      counter_delay <= (others => '1');
    else
      if avs_s1_chipselect = '1' then
        if avs_s1_address(4) = '0' then -- Memory region
          if avs_s1_read = '1' then
            avs_s1_readdata <= ram(conv_integer(ram_address));
          else
            avs_s1_writedata <= value;
          end if;
        else
          if avs_s1_write = '1' then -- Linger register
            counter_delay <= avs_s1_writedata;
          end if;
        end if;
      end if;
    end if;
  end if;
end process;
else  -- No chip select; display memory
        leds <= RAM(case_integer(display_address));
        if counter = x"00000000" then  -- Reset counter
            display_address <= display_address + 1;  -- Next address
            counter <= counter_delay & x"0000";
        else
            counter <= counter - 1;
        end if;
    end if;
end if;
end process;
end if;