```
------------------------------------------------------------------------------------
-- $Archive:: /VHDL/product/c6713dsk/c6713dsk.vhd                                 $
-- $Revision:: 3                                                                  $
-- $Date:: 5/08/03 6:49a                                                          $
-- $Author:: Tonyc                                                                $
--
--
-- Copyright (c) 2002-2003, Spectrum Digital Incorporated
-- All rights reserved
------------------------------------------------------------------------------------
------------------------------------------------------------------------------------
-- Start the real code
------------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;

entity c6713dsk is
  port
  (
    CLKIN        : in    std_logic; -- 12 MHz clock in
    EMU_RSTn     : in    std_logic; -- Emulator reset from USB block
    PONRSn       : in    std_logic; -- Power on reset from voltage supervisor
    PUSHBRS      : in    std_logic; -- Push button reset
    HPIRSn       : in    std_logic; -- HPI reset from optional DC

    -- DSP Memory interface signals for internal CPLD registers.  From C64xx B port.
    DSP_DQ       : inout std_logic_vector( 7 downto 0 );    -- DSP Data bus
    DSP_ADDR     : in    std_logic_vector( 2 downto 0 );    -- DSP Address bus
    DSP_CSn      : in    std_logic;                         -- DSP Chip select
    DSP_RSn      : out   std_logic;                         -- DSP Reset

    CPLD_FLASHn  : in    std_logic;                         -- CPLD or FLASH select

    -- DSP Memory interface signals that control the daughter card. From C6713 emif
    DSP_DC_CS0n  : in    std_logic;                 -- DSP DC Chip select (ACE2#)
    DSP_DC_CS1n  : in    std_logic;                 -- DSP DC Chip select (ACE3#)

    DSP_DC_WEn   : in    std_logic;                 -- DSP DC Write strobe
    DSP_DC_REn   : in    std_logic;                 -- DSP DC Read strobe
    DSP_DC_OEn   : in    std_logic;                 -- DSP DC Output enable

    -- User/Board Support
    USER_SW      : in    std_logic_vector( 3 downto 0 );  -- User swtiches
    USER_LED     : out   std_logic_vector( 3 downto 0 );  -- Uwer led
    PWB_REV      : in    std_logic_vector( 2 downto 0 );  -- PWB revision

    -- Daughter Card Support
    DC_STAT      : in    std_logic_vector( 1 downto 0 );   -- DC Status
    DC_CNTL      : out   std_logic_vector( 1 downto 0 );   -- DC Control
    DC_DBUF_DIR  : out   std_logic;                 -- DC Data buffer direction
    DC_DBUF_OEn  : out   std_logic;                 -- DC Data buffer output enable
    DC_CNTL_OEn  : out   std_logic;                 -- DC Control buffer enable
    DC_DETn      : in    std_logic;                 -- DC Detect
    DC_RESETn    : out   std_logic;                 -- DC Reset

    -- McBSP Multiplexer Control
    MCBSP_SEL0   : out   std_logic;                 -- Codec/DC McBsp 1 mux cntl
    MCBSP_SEL1   : out   std_logic;                 -- Codec/DC McBsp 1 mux cntl

    FLSH_PAGE    : out   std_logic;                 -- Flash Address 19
    FLSH_CEn     : out   std_logic;                 -- Flash chip enable
    FLSH_WEn     : out   std_logic;                 -- Flash write enable
    FLSH_OEn     : out   std_logic;                 -- Flash output enable

    -- Misc. Stuff
    BRD_RSn      : out   std_logic;                 -- Board reset
    DSP_RSn_LED  : out   std_logic;                 -- DSP reset led
    SPARE        : out   std_logic_vector( 3 downto 0 );   -- Spare register bits
    CPLD_CLK_OUT : out   std_logic );               -- Place holder
end c6713dsk;

-------------------------------------------------------------------------------------
-- Include standard librariess
-------------------------------------------------------------------------------------
library IEEE;
use IEEE.std_logic_1164.all;
-- use work.std_arith.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_unsigned.all;

-------------------------------------------------------------------------------------
-- Include fpga specifics here if required.
-- act3  is for Actel 54sx devices.  We normally use this for the hardwired
-- clock definition.
-------------------------------------------------------------------------------------
-- library act3;
-- use act3.components.all;

architecture behavior_c6713dsk of c6713dsk is

constant CPLD_VERSION         : std_logic_vector(3 downto 0)  := "0010";

-------------------------------------------------------------------------------------
-- Add local components in here
-------------------------------------------------------------------------------------
--component MyComponent
--port
--(
--);
--end component;

-------------------------------------------------------------------------------------
-- Add signals
-------------------------------------------------------------------------------------

-- CPLD Register signals
signal    CpldReg0        : std_logic_vector( 7 downto 0 );
signal    CpldReg1        : std_logic_vector( 7 downto 0 );
signal    CpldReg2        : std_logic_vector( 7 downto 0 );
signal    CpldReg3        : std_logic_vector( 7 downto 0 );
signal    CpldReg4        : std_logic_vector( 7 downto 0 );
signal    CpldReg5        : std_logic_vector( 7 downto 0 );
signal    CpldReg6        : std_logic_vector( 7 downto 0 );
signal    CpldReg7        : std_logic_vector( 7 downto 0 );
signal    MuxD            : std_logic_vector( 7 downto 0 );

signal    ChipEnables     : std_logic_vector( 7 downto 0 );
signal    CpldRegCs0      : std_logic;
signal    CpldRegCs1      : std_logic;
signal    CpldRegCs2      : std_logic;
signal    CpldRegCs3      : std_logic;
signal    CpldRegCs4      : std_logic;
signal    CpldRegCs5      : std_logic;
signal    CpldRegCs6      : std_logic;
signal    CpldRegCs7      : std_logic;

signal    SystemResetn    : std_logic;
signal    CpldClkOut      : std_logic;

signal    RsClkEn         : std_logic;
signal    RsTimer         : std_logic_vector( 5 downto 0 );
signal    RsSync          : std_logic;

signal    DSP_WEn         : std_logic;
signal    DSP_REn         : std_logic;
signal    DSP_OEn         : std_logic;

signal    DspAddr         : std_logic_vector( 3 downto 0 );

signal    DcCntlOeOff     : std_logic;
signal    DcAllowBusConflict : std_logic;
-----------------------------------------------------------------------------------
-- The implementation
-----------------------------------------------------------------------------------

begin
-----------------------------------------------------------------------------------
-- Map the other components
-----------------------------------------------------------------------------------

-----------------------------------------------------------------------------------
-- Now define the logic
-----------------------------------------------------------------------------------

  -- Map C6713 memory control to CPLD control signals.
  DSP_WEn  <= DSP_DC_WEn;
  DSP_REn  <= DSP_DC_REn;
  DSP_OEn  <= DSP_DC_OEn;

  -- Merge the low order dsp addresses with CPLD_FLASHn( DSP EA21) signal.
  -- CPLD_FLASHn == 0 then FLASH
  -- CPLD_FLASHn == 1 then CPLD registers
  --
  DspAddr <= CPLD_FLASHn & DSP_ADDR( 2 downto 0 );


  -- Generate a reset from the three sources.
  --
  SystemResetn <= '0' when     EMU_RSTn  = '0'
                            or PONRSn    = '0'
                            or PUSHBRS   = '1'
                            else '1';

  -- Sync up and delay the DSP reset.
  process(SystemResetn, CLKIN  )
  begin
    if( SystemResetn = '0' ) then
      RsTimer <= "000000";
      RsSync  <= '0';
    elsif (CLKIN'event and CLKIN = '1')  then
      RsTimer <= RsTimer + '1';

      if( RsTimer = "111111" ) then
          RsSync <= '1';
      end if;

    end if;
  end process;

  BRD_RSn     <= '0' when SystemResetn = '0' else '1';
  DSP_RSn     <= '0' when RsSync = '0' or HPIRSn = '0' else '1';
  DSP_RSn_LED <= '0' when RsSync = '0' or HPIRSn = '0' else '1';

  -- Generate a CPLD clockout from clock input.  This is a place holder just in
  -- case we need it later.
  --
  process( SystemResetn, CLKIN )
  begin
    if SystemResetn = '0' then
        CpldClkOut <= '0';
    elsif CLKIN'event and CLKIN = '1' then
        CpldClkOut <= not CpldClkOut;
    end if;
  end process;

  CPLD_CLK_OUT <= CpldClkOut;

  -- ##########################################################################
  -- Generic register address decode and register chip select generation.
  -- VHDL compiler will reduce any unused logic so we can be verbose.
  --
  process(  DspAddr )
  begin
    case DspAddr( 3 downto 0) is
      when "1000"  => ChipEnables  <= "00000001";
      when "1001"  => ChipEnables  <= "00000010";
      when "1010"  => ChipEnables  <= "00000100";
      when "1011"  => ChipEnables  <= "00001000";
      when "1100"  => ChipEnables  <= "00010000";
      when "1101"  => ChipEnables  <= "00100000";
      when "1110"  => ChipEnables  <= "01000000";
      when "1111"  => ChipEnables  <= "10000000";
      when others  => ChipEnables  <= "00000000";
    end case;
  end process;

  CpldRegCs0 <= '1' when ChipEnables(0) = '1' and DSP_CSn = '0' else '0';
  CpldRegCs1 <= '1' when ChipEnables(1) = '1' and DSP_CSn = '0' else '0';
  CpldRegCs2 <= '1' when ChipEnables(2) = '1' and DSP_CSn = '0' else '0';
  CpldRegCs3 <= '1' when ChipEnables(3) = '1' and DSP_CSn = '0' else '0';
  CpldRegCs4 <= '1' when ChipEnables(4) = '1' and DSP_CSn = '0' else '0';
  CpldRegCs5 <= '1' when ChipEnables(5) = '1' and DSP_CSn = '0' else '0';
  CpldRegCs6 <= '1' when ChipEnables(6) = '1' and DSP_CSn = '0' else '0';
  CpldRegCs7 <= '1' when ChipEnables(7) = '1' and DSP_CSn = '0' else '0';

  -- ##########################################################################
  -- Generate logic for each CPLD register and assign it's write, read, and
  -- pin values if necessary.
  --
```

```
-- All CPLD register writes occur on the rising edge DSP write strobe.
--

-- ============================================================================
-- REG 0: User Register
-- Bit 3-0   Led 3-0
-- Bit 7-4   Switch 3-0
process( SystemResetn, DSP_WEn, CpldRegCs0, DSP_DQ )
begin
  if SystemResetn = '0' then
      CpldReg0(3 downto 0 ) <= "0000";
  elsif DSP_WEn'event and DSP_WEn = '1' then
    if( CpldRegCs0 = '1' ) then
      CpldReg0( 3 downto 0 ) <= DSP_DQ( 3 downto 0 );
    end if;
  end if;
end process;

CpldReg0(7 downto 4 )  <= USER_SW( 3 downto 0 );
USER_LED( 3 downto 0 ) <= not CpldReg0(3 downto 0 );

-- ============================================================================
-- REG 1: DC Register
-- Bit 1-0   DC_CNTL 1-0
-- Bit 2     NU  read 0
-- Bit 3     DC_RESET
-- Bit 5-4   DC_STAT 1-0
-- Bit 6     NU  read 0
-- Bit 7     DC_DETECT
--
process( SystemResetn, DSP_WEn, CpldRegCs1, DSP_DQ )
begin
  if SystemResetn = '0' then
      CpldReg1(1 downto 0 ) <= "00";
      CpldReg1(3)           <= '0';  -- not Reset by default
  elsif DSP_WEn'event and DSP_WEn = '1' then
    if( CpldRegCs1 = '1' ) then
      CpldReg1( 1 downto 0 ) <= DSP_DQ( 1 downto 0 );
      CpldReg1(3)             <= DSP_DQ(3);
    end if;
  end if;
end process;


CpldReg1(2)            <= '0';
CpldReg1(5 downto 4 )  <= DC_STAT( 1 downto 0 );
CpldReg1(6)            <= '0';
CpldReg1(7)            <= not DC_DETn;

DC_CNTL( 1 downto 0 )  <= CpldReg1( 1 downto 0 );

-- HPIRSn not included in the DC_RESETn equation.  This should prevent the
-- DC from holding itself in reset if HPIRSn is active.
DC_RESETn              <= '0' when    CpldReg1(3)  = '1'
                                   or SystemResetn = '0' else '1';

-- ============================================================================
-- REG 4: Version Register
-- Bit 2-0   PWB Revision 2-0
-- Bit 3     NU read 0
-- Bit 7-4   CPLD version
CpldReg4(7 downto 0 ) <= CPLD_VERSION(3 downto 0 ) & '0' & PWB_REV(2 downto 0 );

-- ============================================================================
-- REG 6: Misc. Register
-- Bit 0     McBSP1 select
-- Bit 1     McBsp2 select
-- Bit 2     Flash Page/Flash Address 19
-- Bit 5-3   Scratch (temp location for user)/SPARE 0-2
-- Bit 6     DcAllowBusConflict/SPARE3
-- Bit 7     DC CNTL OE Override
process( SystemResetn, DSP_WEn, CpldRegCs6, DSP_DQ )
begin
  if SystemResetn = '0' then
      CpldReg6(7 downto 0 ) <= "00000000";
  elsif DSP_WEn'event and DSP_WEn = '1' then
    if( CpldRegCs6 = '1' ) then
      CpldReg6(7 downto 0 ) <= DSP_DQ( 7 downto 0);
    end if;
  end if;
end process;

-- Low  = DC
-- High = Codec
MCBSP_SEL0  <= '1' when CpldReg6(0) = '0' else '0';

-- Low  = DC
-- High = Codec
--
MCBSP_SEL1  <= '1' when CpldReg6(1) = '0' else '0';

FLSH_PAGE   <=     CpldReg6(2);    -- Flash address A19
SPARE(0)    <=     CpldReg6(3);
SPARE(1)    <=     CpldReg6(4);
SPARE(2)    <=     CpldReg6(5);
SPARE(3)    <=     CpldReg6(6);

DcAllowBusConflict <= CpldReg6(6);

DcCntlOeOff <=     '1' when    CpldReg6(7) = '1'
                            or RsSync      = '0'
                            or HPIRSn      = '0' else '0';



-- ===========================================================================
-- Mux the read data from all the registers and output for reads
--
process(  DspAddr,CpldReg0,CpldReg1,CpldReg4,CpldReg6 )
begin
  case DspAddr( 3 downto 0) is
    when "1000"  => MuxD  <= CpldReg0;
    when "1001"  => MuxD  <= CpldReg1;
    when "1100"  => MuxD  <= CpldReg4;
    when "1110"  => MuxD  <= CpldReg6;
    when others  => MuxD  <= "00000000";
  end case;
end process;

DSP_DQ <= MuxD  when    DSP_CSn      = '0'
```

```
                        and  CPLD_FLASHn  = '1'
                        and  DSP_REn      = '0'
                        and  DSP_OEn      = '0'
                     else "ZZZZZZZZ";

  -- ##########################################################################
  -- Generate the Daughter card buffer control signals. DC buffers are
  -- enabled if a daughter card is plugged in to mininize EMI.  DC OE
  -- can also be enabled by DcCntlOeOff = 0.  This is added to allow
  -- the DC_RDY signal to flow through to the DSP on startup.  It also
  -- allows the DC control signals to be enabled for testing/probing when
  -- a daughter card is not plugged in.
  --
  -- DSP OE signal is low for read and high for write.   The DSP OE signal
  -- overlaps the RE/WE signals so the direction is stable before the enable.
  -- This protects DSK and DC from bus conflicts in both directions.
  -- DcAllowBusConflict has been defined to override this protection.
  -- When set the bus buffer is enabled when a DC CS is active.  Simply
  -- changing the direction points the buffer to read or write.  This
  -- should only be done if user sets the EMIF TA paramter to 1 so that
  -- EMIF inserts a dead cycle between reads and writes and CSes.
  --
  -- The default state of the DC buffer is in the write direction. This
  -- adds a little extera protection to the DSP/DSK with respect to bus
  -- conflict.  In theory the daughter card would have to absorb most of
  -- the bus conflict if it occurs.
  --
  --
  DC_DBUF_DIR <= DSP_DC_OEn;      -- high for write, low for read

  DC_DBUF_OEn <= '0' when      DC_DETn    = '0'
                          and ( DSP_DC_CS0n = '0' or  DSP_DC_CS1n = '0' )
                          and (     DSP_DC_REn  = '0'
                                or  DSP_DC_WEn  = '0'
                                or  DcAllowBusConflict = '1'
                              )  else '1';

  DC_CNTL_OEn <= '0' when DC_DETn = '0' or DcCntlOeOff = '1' else '1';

  -- ==========================================================================
  -- Generate the FLASH control signals
  --
  FLSH_CEn <= '0' when DSP_CSn = '0' and CPLD_FLASHn = '0'                  else '1';
  FLSH_WEn <= '0' when DSP_CSn = '0' and CPLD_FLASHn = '0' and DSP_WEn = '0' else '1';
  FLSH_OEn <= '0' when DSP_CSn = '0' and CPLD_FLASHn = '0' and DSP_OEn = '0' else '1';

end behavior_c6713dsk;
```