```
----------------------------------------------------------------------------------
-- Company:
-- Engineer:
--
-- Create Date:    10:36:54 04/05/2009
-- Design Name:
-- Module Name:    dft_core - Behavioral
-- Project Name:
-- Target Devices:
-- Tool versions:
-- Description:
--
-- Dependencies:
--
-- Revision:
-- Revision 0.01 - File Created
-- Additional Comments:
--
----------------------------------------------------------------------------------
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;
use IEEE.NUMERIC_STD.ALL;

---- Uncomment the following library declaration if instantiating
---- any Xilinx primitives in this code.
--library UNISIM;
--use UNISIM.VComponents.all;

entity DFT_CORE is
   generic
      (
      SLV_DWIDTH: integer := 32
      );
    port
      (
      Clk : in std_logic;
      Reset : in std_logic;
      data_written : in  std_logic_vector(0 to 2);
      data_read : in  std_logic_vector(0 to 2);
      command_bit : in  std_logic;
      value_reg : in  std_logic_vector (0 to SLV_DWIDTH-1);
      dft_command : out  std_logic_vector (0 to SLV_DWIDTH-1);
      dft_index : out  std_logic_vector (0 to SLV_DWIDTH-1);
      dft_value : out  std_logic_vector (0 to SLV_DWIDTH-1)
      );
end DFT_CORE;

architecture Behavioral of DFT_CORE is

   type dft_states is (s0, s1, s2, s3, s4);
   signal current_state, next_state : dft_states;

begin

   DFT_SEQ : process( Clk, Reset ) is
      begin
      if ( Clk'event and Clk = '1' ) then
         if ( Reset = '1' ) then
            current_state <= s0;
         else
            current_state <= next_state;
         end if;
      end if;
   end process;

   DFT_PROCESS : process( current_state, command_bit, data_written ) is
      begin

-- Set the default values for the three registers to keep this combinational.
-- Note, none of the registers are updated unless the appropriate bit is set
-- in the dft_command string. Bit 31 MUST always be set -- it allows this state
-- machine to update the command register. Bit 30 is the get_bit -- when set,
-- the state machine is requesting a value from the PPC. Next clock writes both
-- the command register and value register. Bit 29 is the put_bit -- when set,
-- the state machine is requesting the value in the value register to be written
```

```
-- to the index in the index register -- so all three registers are updated when
-- put_bit is set. Bit 28 is the done_bit -- signals to the PPC that all is
-- completed.
      dft_command <= "00000000000000000000000000000001";
      dft_value <= "00000000000000000000000000000000";
      dft_index <= "00000000000000000000000000000000";
      case current_state is
         when s0 =>

-- Wait for PPC to issue the 'go' signal.
            if ( command_bit = '1' ) then
               next_state <= s1;

-- Read from PPC operation, latch these values into command reg and index reg.
               dft_command <= "00000000000000000000000000000011";
               dft_index   <= "00000000000000000000000000000001";
            else
               next_state <= s0;
            end if;
         when s1 =>

-- Wait for the data_written flag -- indicating that the PPC has written
-- the register with the desired value.
            if ( data_written = "001" ) then
               next_state <= s2;
            else
               next_state <= s1;
            end if;
         when s2 =>

-- Wait for the data_written flag to be unset (should happen in next cycle)
-- Add 1 to the dft_value register and issue a write.
            if ( data_written = "000" ) then
               next_state <= s3;
               dft_value <= value_reg + "00000000000000000000000000000001";
               dft_command <= "00000000000000000000000000000101";
               dft_index   <= "00000000000000000000000000000010";
            else
               next_state <= s2;
            end if;

-- Wait for the PPC to read the value and then instruct it to exit.
         when s3 =>
            if ( data_read = "001" ) then
               next_state <= s4;
               dft_command <= "00000000000000000000000000001001";
            else
               next_state <= s3;
            end if;

-- Loop forever.
         when s4 =>
            next_state <= s4;
            dft_command <= "00000000000000000000000000001001";
         when others =>
            next_state <= s0;
      end case;
   end process;

end Behavioral;
```