```vhdl
----------------------------------------------------------------------------
--
--  SoPC Oscilloscope VRAM Controller
--
--  Implementation of the VRAM Controller for the SoPC Oscilloscope project.
--  The state machine generates the necessary timing signals for the VRAM
--  based on the needs of the CPU and display controller. Additionally,
--  it refreshes the VRAM as necessary whenever no other cycle is being
--  performed.
--  The inputs to the system determine what action needs to be performed:
--  cs+we requests a read or a write, while ureq requests a SAM row update.
--  The system then outputs the necessary timing signals, and a rdy/uack
--  signal to notify the sender of the end of the cycle.
--  The state machine is implemented using a Moore state machine and a state
--  assignment architecture.
--
--
--  Revision History:
--     13 Apr 04  Glen George       Initial template.
--     20 Feb 14  Santiago Navonne  Initial revision.
--
----------------------------------------------------------------------------


-- bring in the necessary packages
library  ieee;
use  ieee.std_logic_1164.all;


--
--  Oscilloscope VRAM Controller entity declaration
--

entity  VRAMCtrl  is
    port (
        we       :  in  std_logic;        -- read / not write
        cs       :  in  std_logic;        -- chip select
        ureq     :  in  std_logic;        -- serial row update request
        clk      :  in  std_logic;        -- clock
        Reset    :  in  std_logic;        -- reset the system
        ras      :  out  std_logic;       -- RAS timing signal
        cas      :  out  std_logic;       -- CAS timing signal
        trg      :  out  std_logic;       -- transfer/read signal
        welu     :  out  std_logic;       -- write signal
        asrc     :  out  std_logic;       -- address source selection
        arow     :  out  std_logic;       -- address row/column selection
        uack     :  out  std_logic;       -- serial row update acknowledge
        rdy      :  out  std_logic        -- read/write acknowledge
    );
end  VRAMCtrl;



--
--  Oscilloscope VRAM Controller Moore State Machine
--

architecture  assign_statebits  of  VRAMCtrl  is

    subtype  states  is  std_logic_vector(10 downto 0);     -- state type

    -- define the actual states as constants

    -- bits are: RAS CAS TRG WE ASRC AROW UACK RDY ID[2..0]
    constant  IDLE      : states := "11111100000";  -- waiting for events

    constant  READ1     : states := "01111100000";  -- read state 1
    constant  READ2     : states := "01011000000";  -- read state 2
    constant  READ3     : states := "00011000000";  -- read state 3
    constant  READ4     : states := "00011101000";  -- read state 4
    constant  READ5     : states := "11111100001";  -- read state 5
    constant  READ6     : states := "11111100010";  -- read state 6

    constant  WRITE1    : states := "01111100001";  -- write state 1
    constant  WRITE2    : states := "01101000000";  -- write state 2
    constant  WRITE3    : states := "00101001000";  -- write state 3
    constant  WRITE4    : states := "11111100011";  -- write state 4
    constant  WRITE5    : states := "11111100100";  -- write state 5

    constant  SERIAL1   : states := "11010100000";  -- serial transfer state 1
    constant  SERIAL2   : states := "01010100000";  -- serial transfer state 2
    constant  SERIAL3   : states := "01110000000";  -- serial transfer state 3
    constant  SERIAL4   : states := "00110000000";  -- serial transfer state 4
    constant  SERIAL5   : states := "11111110000";  -- serial transfer state 5
    constant  SERIAL6   : states := "11111100101";  -- serial transfer state 6

    constant  REFRESH1  : states := "10111100000";  -- refresh state 1
    constant  REFRESH2  : states := "00111100001";  -- refresh state 2
    constant  REFRESH3  : states := "00111100010";  -- refresh state 3
    constant  REFRESH4  : states := "00111100011";  -- refresh state 4
    constant  REFRESH5  : states := "11111100110";  -- refresh state 5
    constant  REFRESH6  : states := "11111100111";  -- refresh state 6

    signal  CurrentState  :  states;    -- current state
    signal  NextState     :  states;    -- next state

begin


    -- the output is always the 8 highest bits of the encoding
    ras <= CurrentState(10);
     cas <= CurrentState(9);
     trg <= CurrentState(8);
     welu <= CurrentState(7);
     asrc <= CurrentState(6);
     arow <= CurrentState(5);
     uack <= CurrentState(4);
     rdy <= CurrentState(3);


    -- compute the next state (function of current state and inputs)

    transition:  process (Reset, ureq, we, cs, CurrentState)
    begin

        case  CurrentState  is          -- do the state transition/output

        -- transition from idle
            when  IDLE =>                 -- in idle state, do transition
                if  (ureq = '1')  then
                    NextState <= SERIAL1;      -- serial update request has priority
                elsif  (cs = '0' and we = '1')  then
                    NextState <= READ1;        -- read request
                elsif  (cs = '0' and we = '0')  then
                    NextState <= WRITE1;        -- write request
                else
                    NextState <= REFRESH1;     -- nothing to do; refresh
                end if;

        -- read cycle
            when  READ1 =>             -- continue read cycle
                NextState <= READ2;

            when  READ2 =>             -- continue read cycle
                NextState <= READ3;

            when  READ3 =>           -- continue read cycle
                NextState <= READ4;

            when  READ4 =>           -- continue read cycle
                NextState <= READ5;

            when  READ5 =>           -- continue read cycle
                NextState <= READ6;

            when  READ6 =>           -- end read cycle
                NextState <= IDLE;

        -- write cycle
            when  WRITE1 =>          -- continue write cycle
                NextState <= WRITE2;

            when  WRITE2 =>          -- continue write cycle
                NextState <= WRITE3;

            when  WRITE3 =>          -- continue write cycle
                NextState <= WRITE4;

            when  WRITE4 =>          -- continue write cycle
                NextState <= WRITE5;

            when  WRITE5 =>          -- end write cycle
                NextState <= IDLE;

        -- serial update cycle
            when  SERIAL1 =>         -- continue serial cycle
                NextState <= SERIAL2;

            when  SERIAL2 =>         -- continue serial cycle
                NextState <= SERIAL3;

            when  SERIAL3 =>         -- continue serial cycle
                NextState <= SERIAL4;

            when  SERIAL4 =>         -- continue serial cycle
                NextState <= SERIAL5;

            when  SERIAL5 =>         -- continue serial cycle
                NextState <= SERIAL6;

            when  SERIAL6 =>         -- end serial cycle
                NextState <= IDLE;

            -- refresh cycle
            when  REFRESH1 =>         -- continue refresh cycle
                NextState <= REFRESH2;

            when  REFRESH2 =>         -- continue refresh cycle
                NextState <= REFRESH3;

            when  REFRESH3 =>         -- continue refresh cycle
                NextState <= REFRESH4;

            when  REFRESH4 =>         -- continue refresh cycle
                NextState <= REFRESH5;

            when  REFRESH5 =>         -- continue refresh cycle
                NextState <= REFRESH6;

            when  REFRESH6 =>         -- end refresh cycle
                NextState <= IDLE;

                when  OTHERS   =>          -- default; needed for compilation
                   NextState <= IDLE;
        end case;

        if  Reset = '1'  then              -- reset overrides everything
            NextState <= IDLE;             --   go to idle on reset
        end if;

    end process transition;


    -- storage of current state (loads the next state on the clock)

    process (clk)
    begin

        if  clk = '1'  then                -- only change on rising edge of clock
            CurrentState <= NextState;     -- save the new state information
        end if;

    end process;


end  assign_statebits;
```