```
--------------------------------------------------------------------------------
--
--  Oscilloscope Digital Trigger
--
--  This is an implementation of a trigger for a digital oscilloscope in
--  VHDL.  There are three inputs to the system, one selects the trigger
--  slope and the other two determine the relationship between the trigger
--  level and the signal level.  The only output is a trigger signal which
--  indicates a trigger event has occurred.
--
--  The file contains multiple architectures for a Moore state machine
--  implementation to demonstrate the different ways of building a state
--  machine.
--
--
--  Revision History:
--     13 Apr 04  Glen George       Initial revision.
--      4 Nov 05  Glen George       Updated comments.
--     17 Nov 07  Glen George       Updated comments.
--     13 Feb 10  Glen George       Added more example architectures.
--     01 Mar 14  Santiago Navonne  Removed unnecessary architectures.
--
--------------------------------------------------------------------------------


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


--
--  Oscilloscope Digital Trigger entity declaration
--

entity  ScopeTrigger  is
    port (
        TS         :  in  std_logic;      -- trigger slope (1 -> negative, 0 -> positive)
        TEQ        :  in  std_logic;      -- signal and trigger levels equal
        TLT        :  in  std_logic;      -- signal level < trigger level
        clk        :  in  std_logic;      -- clock
        Reset      :  in  std_logic;      -- reset the system
        TrigEvent  :  out  std_logic      -- a trigger event has occurred
    );
end  ScopeTrigger;


--
--  Oscilloscope Digital Trigger Moore State Machine
--     State Assignment Architecture
--
--  This architecture just shows the basic state machine syntax when the state
--  assignments are made manually.  This is useful for minimizing output
--  decoding logic and avoiding glitches in the output (due to the decoding
--  logic).
--

architecture  assign_statebits  of  ScopeTrigger  is

    subtype  states  is  std_logic_vector(2 downto 0);     -- state type

    -- define the actual states as constants
    constant  IDLE      : states := "000";  -- waiting for start of trigger event
    constant  WAIT_POS  : states := "001";  -- waiting for positive slope trigger
    constant  WAIT_NEG  : states := "010";  -- waiting for negative slope trigger
    constant  TRIGGER   : states := "100";  -- got a trigger event


    signal  CurrentState  :  states;    -- current state
```

```
    signal  NextState     :  states;    -- next state

begin


    -- the output is always the high bit of the state encoding
    TrigEvent <= CurrentState(2);


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

    transition:  process (Reset, TS, TEQ, TLT, CurrentState)
    begin

        case  CurrentState  is          -- do the state transition/output

            when  IDLE =>               -- in idle state, do transition
                if  (TS = '0' and TLT = '1' and TEQ = '0')  then
                    NextState <= WAIT_POS;      -- below trigger and + slope
                elsif  (TS = '1' and TLT = '0' and TEQ = '0')  then
                    NextState <= WAIT_NEG;      -- above trigger and - slope
                else
                    NextState <= IDLE;          -- trigger not possible yet
                end if;

            when  WAIT_POS =>           -- waiting for positive slope trigger
                if  (TS = '0' and TLT = '1')  then
                    NextState <= WAIT_POS;      -- no trigger yet
                elsif  (TS = '0' and TLT = '0')  then
                    NextState <= TRIGGER;       -- got a trigger
                else
                    NextState <= IDLE;          -- trigger slope changed
                end if;

            when  WAIT_NEG =>           -- waiting for negative slope trigger
                if  (TS = '1' and TLT = '0' and TEQ = '0')  then
                    NextState <= WAIT_NEG;      -- no trigger yet
                elsif  (TS = '1' and (TLT = '1' or TEQ = '1'))  then
                    NextState <= TRIGGER;       -- got a trigger
                else
                    NextState <= IDLE;          -- trigger slope changed
                end if;

            when  TRIGGER =>            -- in the trigger state
                NextState <= IDLE;      -- always go back to idle

                when others =>
                     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;
```

```
137
138
139 end  assign_statebits;
140
```