```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.std_logic_unsigned.all;
USE ieee.numeric_std.ALL;


entity TekBot is
    Port ( reset_n     : in  STD_LOGIC; --active low reset
           SensorR     : in  STD_LOGIC; --active low sensor on right side
           SensorL     : in  STD_LOGIC; --active low sensor on left side
           Timer_done   : in  STD_LOGIC; --active high signal that 1/2 second is up
           clk         : in  STD_LOGIC; --
           Left             : out STD_LOGIC; --controls left motor 1=forward 0=backward
           Right            : out STD_LOGIC); --controls right motor 1=forward 0=backward
end TekBot;

architecture behavioral of TekBot is

    type state_type is (FORWARD,BACK,TURNL,TURNR);  --four states for this TekBot
    signal current_state, next_state : state_type;

    begin

       --This process moves the state machine to the next state on the rising edge clock
        sync: process(clk, reset_n) is
            begin
                if (reset_n = '0') then         --always default to moving forward
                    current_state <= FORWARD;
                elsif (clk'event and clk='1') then
                    current_state <= next_state;
                end if;
            end process;

      --This process sets up the next state based on the current state and the inputs
        comb: process(current_state,SensorL,SensorR,timer_done) is
            begin
                case (current_state) is
                    when FORWARD =>
                        if (SensorR = '0' or SensorL = '0') then  -- if either sensor hits, move back
                            next_state <= BACK;
                        else
                            next_state <= FORWARD;
                        end if;
                    when BACK =>
                        if (timer_done = '0') then     --move back for 1/2 second
                            next_state <= back;
                        elsif (SensorL = '0') then     --turn right if left sensor hit
                            next_state <= TurnR;
                        else
                            next_state <= TurnL;        --turn left if right sensor hit
                        end if;
                    when TurnL =>
                        if (timer_done = '0') then     --turn left for 1/2 second
                            next_state <= TurnL;
                        else
                            next_state <= FORWARD;
                        end if;
                    when TurnR =>                     --turn right for 1/2 second
                        if (timer_done = '0') then
                            next_state <= TurnR;
                        else
                            next_state <= FORWARD;
                        end if;
                    when OTHERS =>
                        next_state <= FORWARD;         --default state is forward
                end case;
            end process;

        output_right: process(current_state) is --Moore FSM, output is dependent only on state
            begin
                case (current_state) is
                    when FORWARD | TURNL =>        --right motor goes forward in left turn
                        right <= '1';
                    when OTHERS =>                 --backwards in right turn
                        right <= '0';
                end case;
            end process;

        output_left: process(current_state) is
            begin
                case (current_state) is
                    when FORWARD | TurnR =>        --left motot goes forward in right turn
                        left <= '1';
                    when OTHERS =>                 --backwards in left turn
                        left <= '0';
                end case;
            end process;


    end behavioral;

```