```
---------------------------------------------------------------------------------
      -
-- Company: 
-- Engineer: David Paquette
-- 
-- Create Date:    16:49:31 11/19/2015 
-- Design Name: 
-- Module Name:    PIDController - 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.numeric_std.all;

-- Uncomment the following library declaration if using
-- arithmetic functions with Signed or Unsigned values
--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 PIDController is
      Port (      samplingRateClock : in std_logic;
                        reset : in std_logic;
                        setpoint: in integer range 0 to 100;
                        sensorFeedbackValue : in integer range 0 to 100;
                        controlOutput : out integer range 0 to 100
            );
end PIDController;

architecture Behavioral of PIDController is
      signal controllerOutput : integer range 0 to 100:=0;
      constant kp : integer range -1000 to 0:=-300;
      constant ki : integer range -1000 to 0:=-2;
      constant kd : integer range -1000 to 0:=0;
begin
      process(samplingRateClock, reset)
            variable error :  integer range -1000 to 1000:=0;
            variable previousError : integer range -1000 to 1000:=0;
            variable errorSum:  integer range -100000 to 100000:=0;
            variable errorChange:  integer range -1000 to 1000:=0;
            variable output:  integer range -1000 to 1000:=0;
      begin
```

```
            if(reset='0') then
                  error:=0;
                  previousError:=0;
                  errorSum:=0;
                  errorChange:=0;
                  output:=0;
                  controllerOutput<= 0;
            elsif(samplingRateClock'event and samplingRateClock='1') then
                  error := (setpoint - sensorFeedbackValue);
                  errorSum    := errorSum + error;
                  if(errorSum > 10000) then
                        errorSum := 10000;
                  elsif(errorSum < -10000) then
                        errorSum := -10000;
                  end if;
                  errorChange := error - previousError;
                  output := (kp*error + ki*errorSum + kd*errorChange)/100;
                  previousError := error;
                  if(output>100) then
                        output := 100;
                  elsif(output<0)then
                        output:=0;
                  end if;
                  controllerOutput<= output;
            end if;
      end process;

      controlOutput<=controllerOutput;

end Behavioral;
```