Christopher Spadavecchia
Eli Shtindler
CPE 487

Lab 4

Project hexcalc:

[Original Hex Calculator]

After uploading the original code, the calculator does not stop additional values from being displayed after entering more than four values and it does not have the subtraction operator.

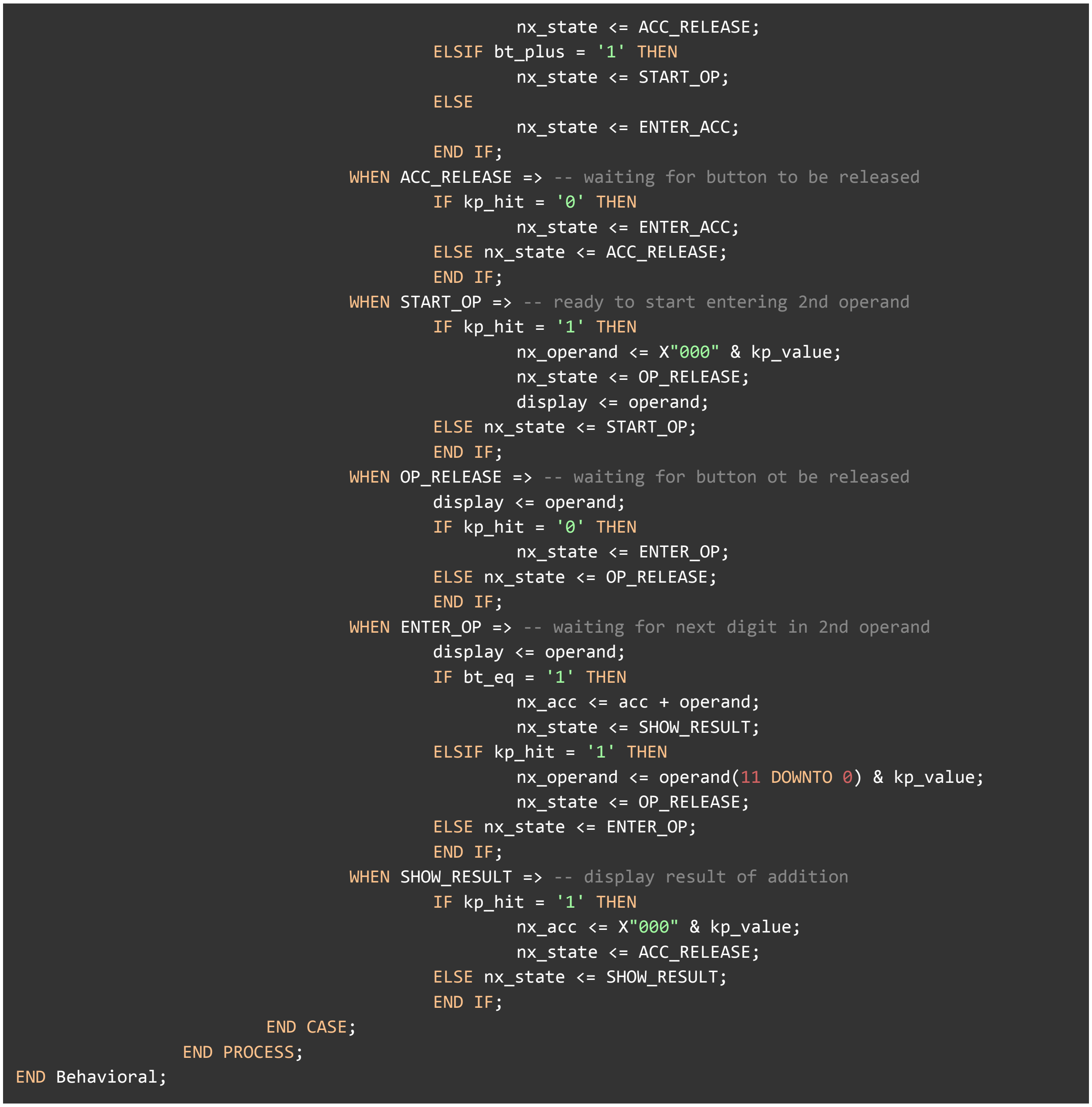
[Modified Hex Calculator]

After modifying the code, the monitor calculator stops additional values from being displayed after entering more than four values and it does have the subtraction operator.

hexcalc.vhd

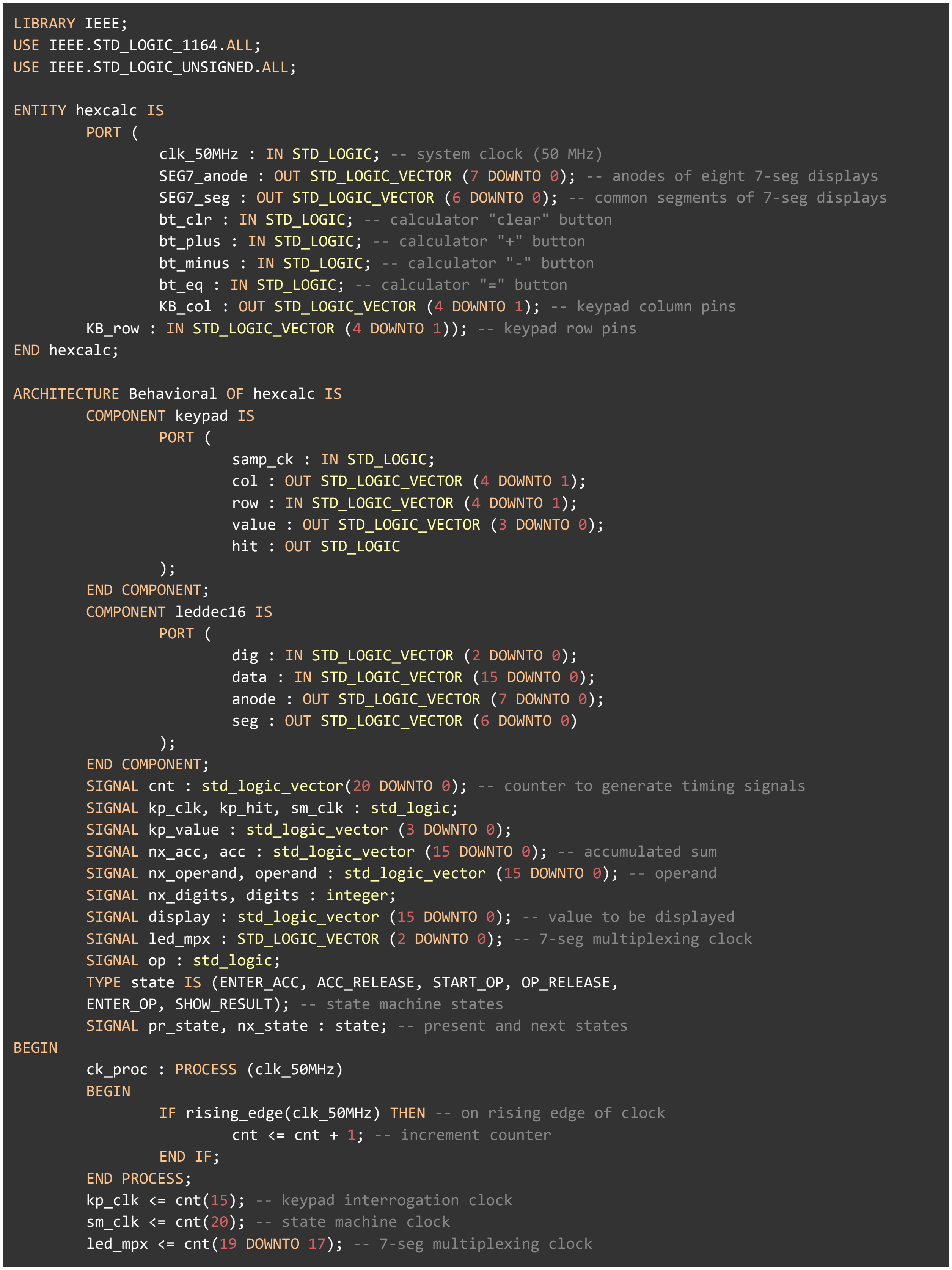
Original:

```
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY hexcalc IS
        PORT (
                clk_50MHz : IN STD_LOGIC; -- system clock (50 MHz)
                SEG7_anode : OUT STD_LOGIC_VECTOR (7 DOWNTO 0); -- anodes of eight 7-seg displays
                SEG7_seg : OUT STD_LOGIC_VECTOR (6 DOWNTO 0); -- common segments of 7-seg displays
                bt_clr : IN STD_LOGIC; -- calculator "clear" button
                bt_plus : IN STD_LOGIC; -- calculator "+" button
                bt_eq : IN STD_LOGIC; -- calculator "=" button
                KB_col : OUT STD_LOGIC_VECTOR (4 DOWNTO 1); -- keypad column pins
        KB_row : IN STD_LOGIC_VECTOR (4 DOWNTO 1)); -- keypad row pins
END hexcalc;

ARCHITECTURE Behavioral OF hexcalc IS
        COMPONENT keypad IS
                PORT (
                        samp_ck : IN STD_LOGIC;
                        col : OUT STD_LOGIC_VECTOR (4 DOWNTO 1);
                        row : IN STD_LOGIC_VECTOR (4 DOWNTO 1);
                        value : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
                        hit : OUT STD_LOGIC
                );
        END COMPONENT;
        COMPONENT leddec16 IS
                PORT (
                        dig : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
                        data : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
                        anode : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
                        seg : OUT STD_LOGIC_VECTOR (6 DOWNTO 0)
                );
        END COMPONENT;
        SIGNAL cnt : std_logic_vector(20 DOWNTO 0); -- counter to generate timing signals
        SIGNAL kp_clk, kp_hit, sm_clk : std_logic;
        SIGNAL kp_value : std_logic_vector (3 DOWNTO 0);
        SIGNAL nx_acc, acc : std_logic_vector (15 DOWNTO 0); -- accumulated sum
        SIGNAL nx_operand, operand : std_logic_vector (15 DOWNTO 0); -- operand
        SIGNAL display : std_logic_vector (15 DOWNTO 0); -- value to be displayed
        SIGNAL led_mpx : STD_LOGIC_VECTOR (2 DOWNTO 0); -- 7-seg multiplexing clock
        TYPE state IS (ENTER_ACC, ACC_RELEASE, START_OP, OP_RELEASE,
        ENTER_OP, SHOW_RESULT); -- state machine states
        SIGNAL pr_state, nx_state : state; -- present and next states
BEGIN
        ck_proc : PROCESS (clk_50MHz)
        BEGIN
                IF rising_edge(clk_50MHz) THEN -- on rising edge of clock
                        cnt <= cnt + 1; -- increment counter
                END IF;
        END PROCESS;
        kp_clk <= cnt(15); -- keypad interrogation clock
        sm_clk <= cnt(20); -- state machine clock
        led_mpx <= cnt(19 DOWNTO 17); -- 7-seg multiplexing clock
        kp1 : keypad
        PORT MAP(
                samp_ck => kp_clk, col => KB_col,
                row => KB_row, value => kp_value, hit => kp_hit
                );
                led1 : leddec16
                PORT MAP(
                        dig => led_mpx, data => display,
                        anode => SEG7_anode, seg => SEG7_seg
                );
                sm_ck_pr : PROCESS (bt_clr, sm_clk) -- state machine clock process
                BEGIN
                        IF bt_clr = '1' THEN -- reset to known state
                                acc <= X"0000";
                                operand <= X"0000";
                                pr_state <= ENTER_ACC;
                        ELSIF rising_edge (sm_clk) THEN -- on rising clock edge
                                pr_state <= nx_state; -- update present state
                                acc <= nx_acc; -- update accumulator
                                operand <= nx_operand; -- update operand
                        END IF;
                END PROCESS;
                -- state maching combinatorial process
                -- determines output of state machine and next state
                sm_comb_pr : PROCESS (kp_hit, kp_value, bt_plus, bt_eq, acc, operand, pr_state)
                BEGIN
                        nx_acc <= acc; -- default values of nx_acc, nx_operand & display
                        nx_operand <= operand;
                        display <= acc;
                        CASE pr_state IS -- depending on present state...
                                WHEN ENTER_ACC => -- waiting for next digit in 1st operand entry
                                        IF kp_hit = '1' THEN
                                                nx_acc <= acc(11 DOWNTO 0) & kp_value;
```

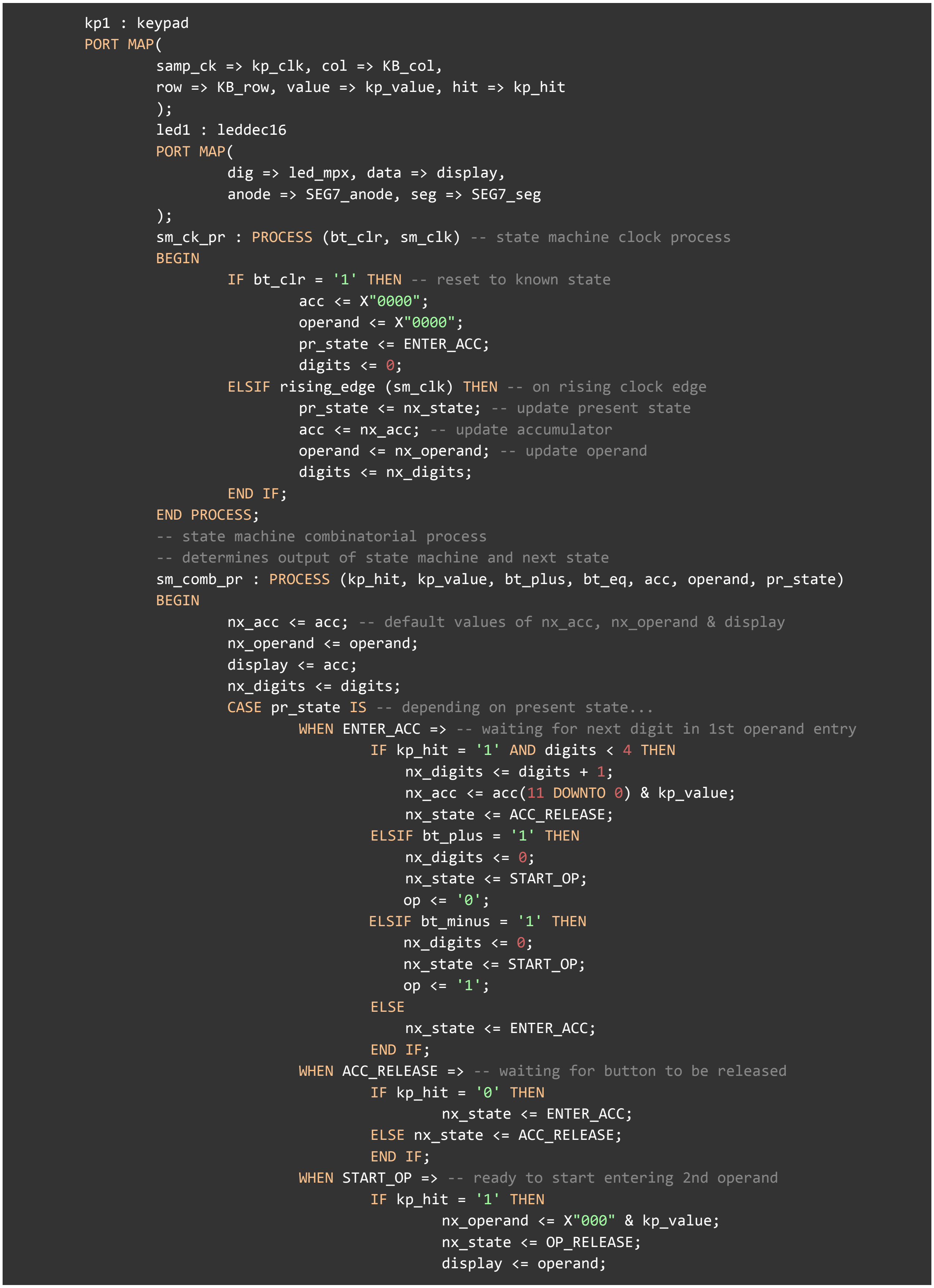


```
                                                nx_state <= ACC_RELEASE;
                                        ELSIF bt_plus = '1' THEN
                                                nx_state <= START_OP;
                                        ELSE
                                                nx_state <= ENTER_ACC;
                                        END IF;
                                WHEN ACC_RELEASE => -- waiting for button to be released
                                        IF kp_hit = '0' THEN
                                                nx_state <= ENTER_ACC;
                                        ELSE nx_state <= ACC_RELEASE;
                                        END IF;
                                WHEN START_OP => -- ready to start entering 2nd operand
                                        IF kp_hit = '1' THEN
                                                nx_operand <= X"000" & kp_value;
                                                nx_state <= OP_RELEASE;
                                                display <= operand;
                                        ELSE nx_state <= START_OP;
                                        END IF;
                                WHEN OP_RELEASE => -- waiting for button ot be released
                                        display <= operand;
                                        IF kp_hit = '0' THEN
                                                nx_state <= ENTER_OP;
                                        ELSE nx_state <= OP_RELEASE;
                                        END IF;
                                WHEN ENTER_OP => -- waiting for next digit in 2nd operand
                                        display <= operand;
                                        IF bt_eq = '1' THEN
                                                nx_acc <= acc + operand;
                                                nx_state <= SHOW_RESULT;
                                        ELSIF kp_hit = '1' THEN
                                                nx_operand <= operand(11 DOWNTO 0) & kp_value;
                                                nx_state <= OP_RELEASE;
                                        ELSE nx_state <= ENTER_OP;
                                        END IF;
                                WHEN SHOW_RESULT => -- display result of addition
                                        IF kp_hit = '1' THEN
                                                nx_acc <= X"000" & kp_value;
                                                nx_state <= ACC_RELEASE;
                                        ELSE nx_state <= SHOW_RESULT;
                                        END IF;
                        END CASE;
                END PROCESS;
END Behavioral;
```

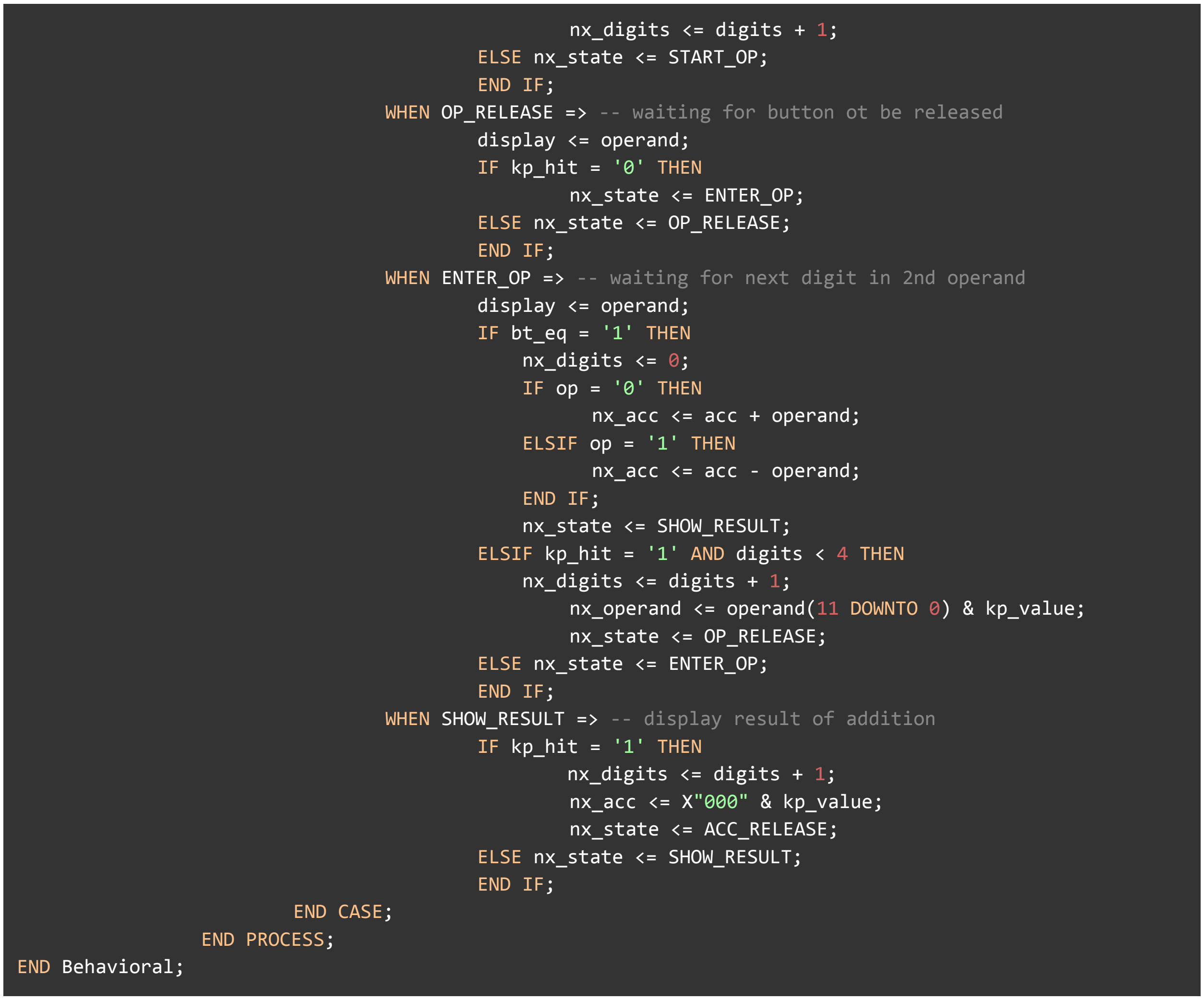
Modified:



```vhdl
LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.STD_LOGIC_UNSIGNED.ALL;

ENTITY hexcalc IS
        PORT (
                clk_50MHz : IN STD_LOGIC; -- system clock (50 MHz)
                SEG7_anode : OUT STD_LOGIC_VECTOR (7 DOWNTO 0); -- anodes of eight 7-seg displays
                SEG7_seg : OUT STD_LOGIC_VECTOR (6 DOWNTO 0); -- common segments of 7-seg displays
                bt_clr : IN STD_LOGIC; -- calculator "clear" button
                bt_plus : IN STD_LOGIC; -- calculator "+" button
                bt_minus : IN STD_LOGIC; -- calculator "-" button
                bt_eq : IN STD_LOGIC; -- calculator "=" button
                KB_col : OUT STD_LOGIC_VECTOR (4 DOWNTO 1); -- keypad column pins
        KB_row : IN STD_LOGIC_VECTOR (4 DOWNTO 1)); -- keypad row pins
END hexcalc;

ARCHITECTURE Behavioral OF hexcalc IS
        COMPONENT keypad IS
                PORT (
                        samp_ck : IN STD_LOGIC;
                        col : OUT STD_LOGIC_VECTOR (4 DOWNTO 1);
                        row : IN STD_LOGIC_VECTOR (4 DOWNTO 1);
                        value : OUT STD_LOGIC_VECTOR (3 DOWNTO 0);
                        hit : OUT STD_LOGIC
                );
        END COMPONENT;
        COMPONENT leddec16 IS
                PORT (
                        dig : IN STD_LOGIC_VECTOR (2 DOWNTO 0);
                        data : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
                        anode : OUT STD_LOGIC_VECTOR (7 DOWNTO 0);
                        seg : OUT STD_LOGIC_VECTOR (6 DOWNTO 0)
                );
        END COMPONENT;
        SIGNAL cnt : std_logic_vector(20 DOWNTO 0); -- counter to generate timing signals
        SIGNAL kp_clk, kp_hit, sm_clk : std_logic;
        SIGNAL kp_value : std_logic_vector (3 DOWNTO 0);
        SIGNAL nx_acc, acc : std_logic_vector (15 DOWNTO 0); -- accumulated sum
        SIGNAL nx_operand, operand : std_logic_vector (15 DOWNTO 0); -- operand
        SIGNAL nx_digits, digits : integer;
        SIGNAL display : std_logic_vector (15 DOWNTO 0); -- value to be displayed
        SIGNAL led_mpx : STD_LOGIC_VECTOR (2 DOWNTO 0); -- 7-seg multiplexing clock
        SIGNAL op : std_logic;
        TYPE state IS (ENTER_ACC, ACC_RELEASE, START_OP, OP_RELEASE,
        ENTER_OP, SHOW_RESULT); -- state machine states
        SIGNAL pr_state, nx_state : state; -- present and next states
BEGIN
        ck_proc : PROCESS (clk_50MHz)
        BEGIN
                IF rising_edge(clk_50MHz) THEN -- on rising edge of clock
                        cnt <= cnt + 1; -- increment counter
                END IF;
        END PROCESS;
        kp_clk <= cnt(15); -- keypad interrogation clock
        sm_clk <= cnt(20); -- state machine clock
        led_mpx <= cnt(19 DOWNTO 17); -- 7-seg multiplexing clock
```



```
kp1 : keypad
PORT MAP(
        samp_ck => kp_clk, col => KB_col,
        row => KB_row, value => kp_value, hit => kp_hit
        );
        led1 : leddec16
        PORT MAP(
                dig => led_mpx, data => display,
                anode => SEG7_anode, seg => SEG7_seg
        );
        sm_ck_pr : PROCESS (bt_clr, sm_clk) -- state machine clock process
        BEGIN
                IF bt_clr = '1' THEN -- reset to known state
                        acc <= X"0000";
                        operand <= X"0000";
                        pr_state <= ENTER_ACC;
                        digits <= 0;
                ELSIF rising_edge (sm_clk) THEN -- on rising clock edge
                        pr_state <= nx_state; -- update present state
                        acc <= nx_acc; -- update accumulator
                        operand <= nx_operand; -- update operand
                        digits <= nx_digits;
                END IF;
        END PROCESS;
        -- state machine combinatorial process
        -- determines output of state machine and next state
        sm_comb_pr : PROCESS (kp_hit, kp_value, bt_plus, bt_eq, acc, operand, pr_state)
        BEGIN
                nx_acc <= acc; -- default values of nx_acc, nx_operand & display
                nx_operand <= operand;
                display <= acc;
                nx_digits <= digits;
                CASE pr_state IS -- depending on present state...
                        WHEN ENTER_ACC => -- waiting for next digit in 1st operand entry
                                IF kp_hit = '1' AND digits < 4 THEN
                                    nx_digits <= digits + 1;
                                    nx_acc <= acc(11 DOWNTO 0) & kp_value;
                                    nx_state <= ACC_RELEASE;
                                ELSIF bt_plus = '1' THEN
                                    nx_digits <= 0;
                                    nx_state <= START_OP;
                                    op <= '0';
                                ELSIF bt_minus = '1' THEN
                                    nx_digits <= 0;
                                    nx_state <= START_OP;
                                    op <= '1';
                                ELSE
                                    nx_state <= ENTER_ACC;
                                END IF;
                        WHEN ACC_RELEASE => -- waiting for button to be released
                                IF kp_hit = '0' THEN
                                        nx_state <= ENTER_ACC;
                                ELSE nx_state <= ACC_RELEASE;
                                END IF;
                        WHEN START_OP => -- ready to start entering 2nd operand
                                IF kp_hit = '1' THEN
                                        nx_operand <= X"000" & kp_value;
                                        nx_state <= OP_RELEASE;
                                        display <= operand;
```



```
                                                nx_digits <= digits + 1;
                                        ELSE nx_state <= START_OP;
                                        END IF;
                                WHEN OP_RELEASE => -- waiting for button ot be released
                                        display <= operand;
                                        IF kp_hit = '0' THEN
                                                nx_state <= ENTER_OP;
                                        ELSE nx_state <= OP_RELEASE;
                                        END IF;
                                WHEN ENTER_OP => -- waiting for next digit in 2nd operand
                                        display <= operand;
                                        IF bt_eq = '1' THEN
                                            nx_digits <= 0;
                                            IF op = '0' THEN
                                                  nx_acc <= acc + operand;
                                            ELSIF op = '1' THEN
                                                  nx_acc <= acc - operand;
                                            END IF;
                                            nx_state <= SHOW_RESULT;
                                        ELSIF kp_hit = '1' AND digits < 4 THEN
                                            nx_digits <= digits + 1;
                                                nx_operand <= operand(11 DOWNTO 0) & kp_value;
                                                nx_state <= OP_RELEASE;
                                        ELSE nx_state <= ENTER_OP;
                                        END IF;
                                WHEN SHOW_RESULT => -- display result of addition
                                        IF kp_hit = '1' THEN
                                                nx_digits <= digits + 1;
                                                nx_acc <= X"000" & kp_value;
                                                nx_state <= ACC_RELEASE;
                                        ELSE nx_state <= SHOW_RESULT;
                                        END IF;
                        END CASE;
                END PROCESS;
END Behavioral;
```

In order to prevent more than 4 numbers from being entered, two new signals were created, digits and nx_digits. Whenever a number was entered, nx_digits was set to digits+1. When moving from one operand to another, nx_digits was set to 0. digits only gets updated to have the value of nx_digits on the rising edge of clock cycles, ensuring that it stays in sync with the rest of the FSM. An additional condition was placed in the FSM to only allow numbers to be entered if digits<4. This ensures that only 4 numbers can be entered per operand.

To implement subtraction a new signal named op was created. In the ENTER_ACC state, a new transition was created when the subtraction button was pressed. When doing addition op is set to 0 and when doing subtraction op is set to 1. When transitioning to the SHOW_RESULT state, the corresponding operation is done based on the value of op.