10/29/2024

Lab 06 Report

Ryan Beck
Jared Bronson

Objectives

Implement a revised version of the Lab 5 Accumulator on the DE10-Lite development board. The design will use the two on-board push buttons. One push button will reset the board, and clear the accumulated value. The other push button will store the value on the 10-toggle switches in a FIFO. When five items are in the FIFO, it will be drained and the values from the FIFO will be added to the accumulated value. The 24-bit accumulated value will be displayed on the six 7-segment displays. The value of the toggle switches will be reflected on the 10 LEDs. Each Finite State Machine(FSM) will run on different clocks using a Phase-Locked Loop(PLL). The first FSM will run on 5 MHz clock and will manage the write side of the FIFO and Accumulator. The second FSM will run on a 12.5 MHz clock and will manage the read side of the FIFO and Accumulator. This must be implemented entirely in VHDL.

Procedures

Map out the first FSM with states for Clear, Waiting, Debounce, Pressed, Check, and Write. Map out the second FSM with states for Clear, Empty, Waiting, Accumulate, and Display. Next, create a Quartus Project File with the system builder to include the on-board clock, seven-segment displays, LEDs, push buttons, and toggle switches. Create a VHDL file with port names to match what is in the generated Verilog file, then remove the Verilog file from the project. In the generic, create an integer to use as a delay in Debounce to make sure the button does not add after false triggering.

Next, create a 10x128 FIFO to accept the value for the switches when the store button is pushed. Make variables that will be taken in by the FIFO to enable the writing process for the first FSM. Variables to enable the reading process for the second FSM will also be needed.

Create a PLL with needed variables to output the clock frequencies needed for each of the state machines. Initialize any other needed variables for the design of the FIFO Accumulator in the architecture. For the seven-segment display, make a lookup table to display each hex value. Last, create a type to be able to move through the states of each state machine.

Start by instantiating the FIFO and the PLL. Then, create a process sensitive to the reset key, to clear the FIFO. Set one process sensitive to the 5 MHz clock and one sensitive to the 12.5 MHz clock. In these two processes set the corresponding FSM to go to the clear state when reset is pushed, and set the current state to the next state.

In the next process, the sensitivity list will include KEY, fifo_full, and the 5 MHz clock. This process will control the first state machine. In the clear state, all values will be cleared and the state machine goes to Waiting. The FSM goes to the Clear state if reset is pushed, and Debounce if store is pushed. A timer is incremented in the Debounce state. When the timer equals the delay value, if the button is still pushed, the FSM will go to the Pressed state. If the button is not pushed when the timer equals delay, the FSM will go back to the Waiting state. While the FSM is in the Pressed state, it waits for the store button to be released and will go to the Check state. The Check state looks at the amount of input in the FIFO. If the FIFO is full, the FSM goes to the back to the Waiting state, otherwise it will go to the writing state. Once in the Writing state, the value given by the 10 toggle switches is stored in the FIFO, and the FSM goes back to Waiting.

The next process will control the second FSM and will be sensitive to the push buttons, fifo_empty, fifo_full, rd_en, and the 12.5 MHz clock. The Clear state will set everything to zero including the seven-segment display, and go into the Empty state. The FIFO is cleared in the Empty state by reading what is in the FIFO until it is empty. Once the FIFO is empty, the FSM goes to Waiting. If reset is pushed, the FSM goes back to the Clear state. This Waiting state will send the FSM to Accumulate only when the FIFO has 5 inputs. The Accumulate state will take an input out of the FIFO and add it to the accumulated value. Once the FIFO is empty, the FSM will navigate to the Display state. The Display state takes the 24-bit accumulated value and shows it on the six 7-segment displays. After the value is displayed, the FSM goes back to Waiting.

Define one more process sensitive to the 50 MHz clock to continuously update the LEDs to show the state of the 10 toggle switches. Implement the design on the development board to ensure it works as intented.

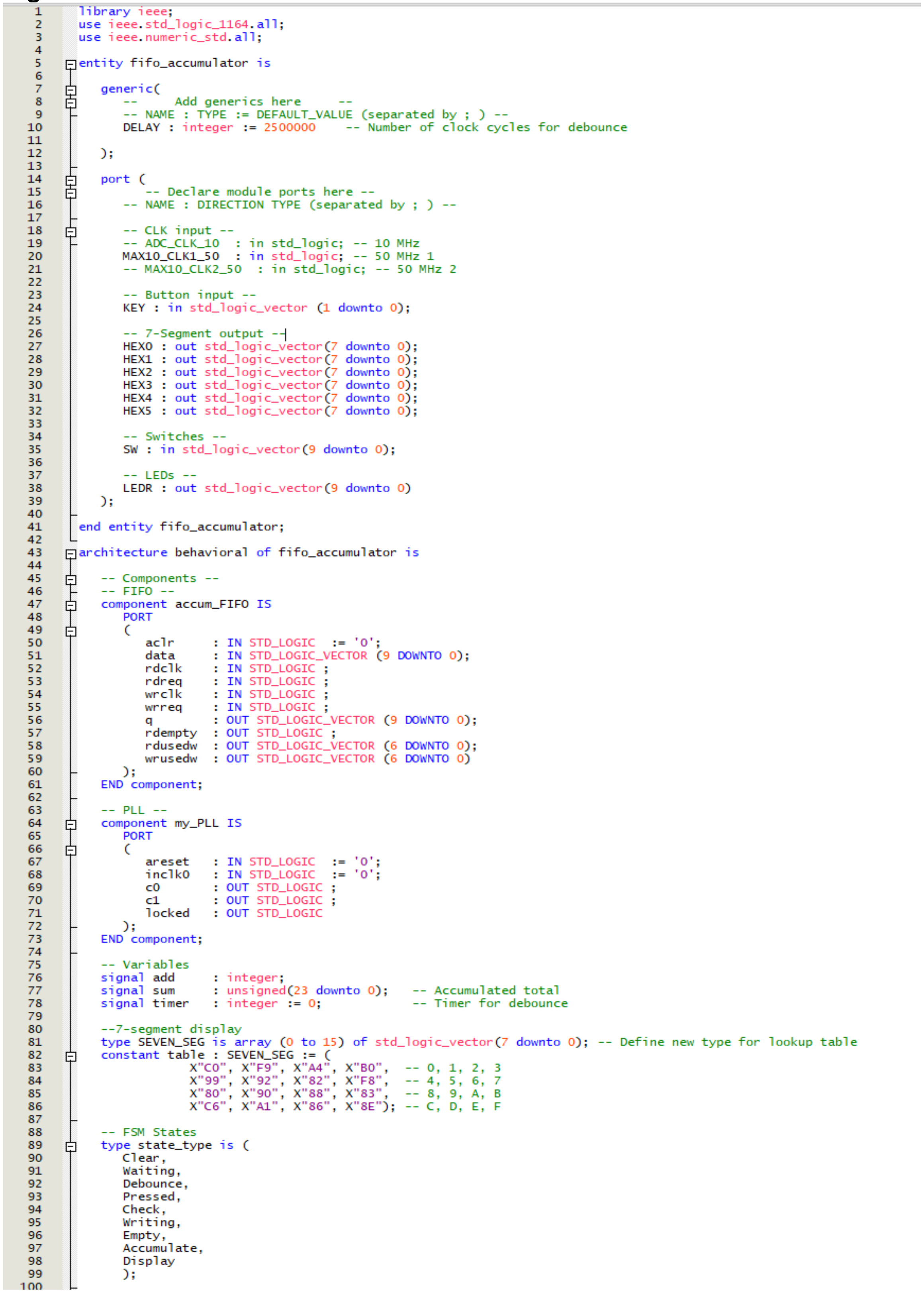
Results

The FIFO_Accumulator file shown in figures successfully implements the design. There were a lot of issues implementing the two state machines, and processing data in the FIFO properly. There were problems with reading from the FIFO at the proper time, inferred latches, and the FSMs not going between states properly. To debug the design, a test bench was used to run simulations, as well as the signal tap logic analyzer in Quartus. Because of the issues, the design instantiates the PLL, but the FSMs are running off of the same 50 MHz clock.

Conclusion

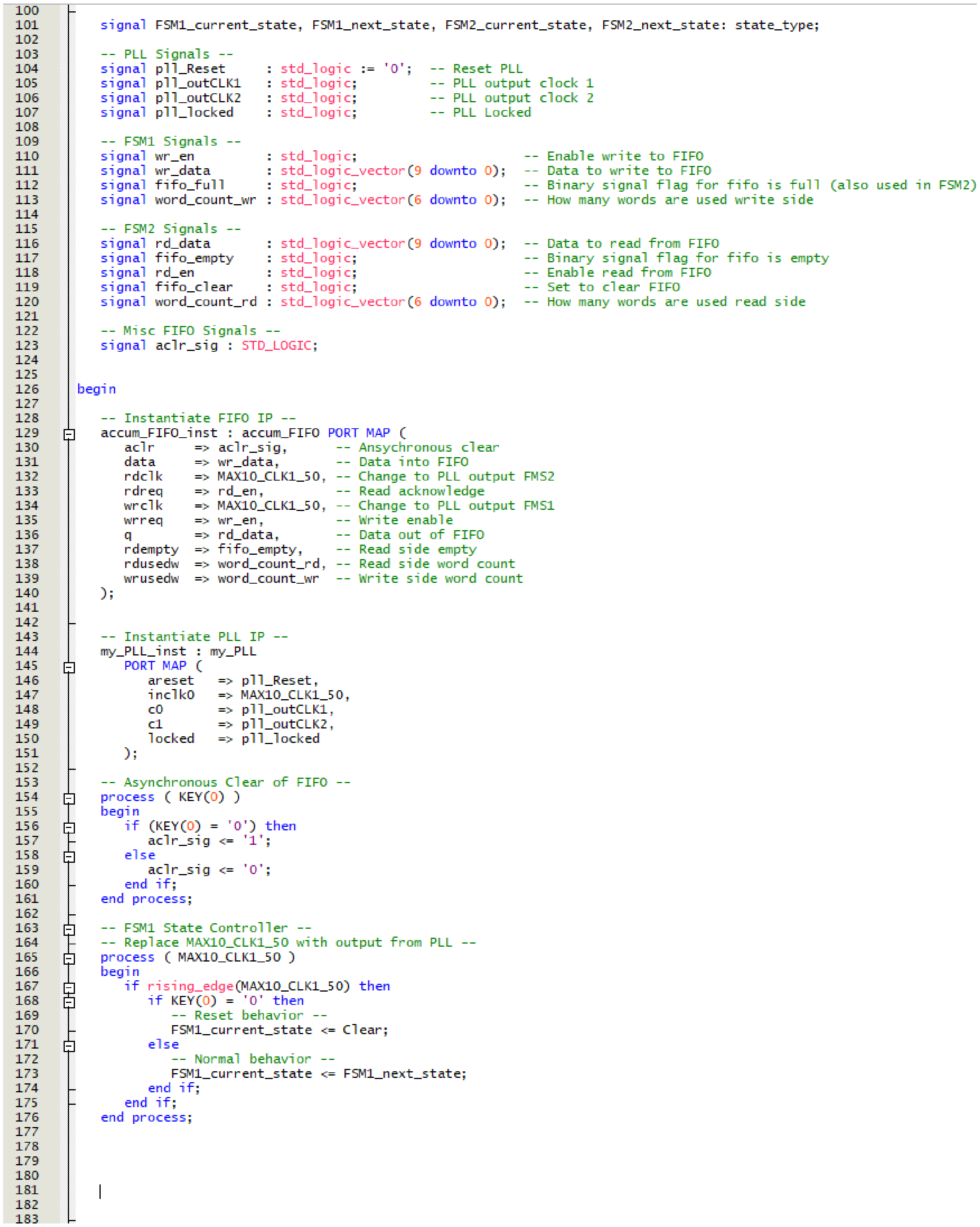
In conclusion, we were able to implement a design using two state machines. The design took input from the 10 toggle switches and stored it in a FIFO when the store button was pushed. When the FIFO was full, the values were read and added to the accumulated value. The clear button resets all the values, including the 7-segment displays. The value of the toggle switches displays on the LEDs. While we instantiated the PLL were not able to implement it into the design of the two FSMs.

Figures



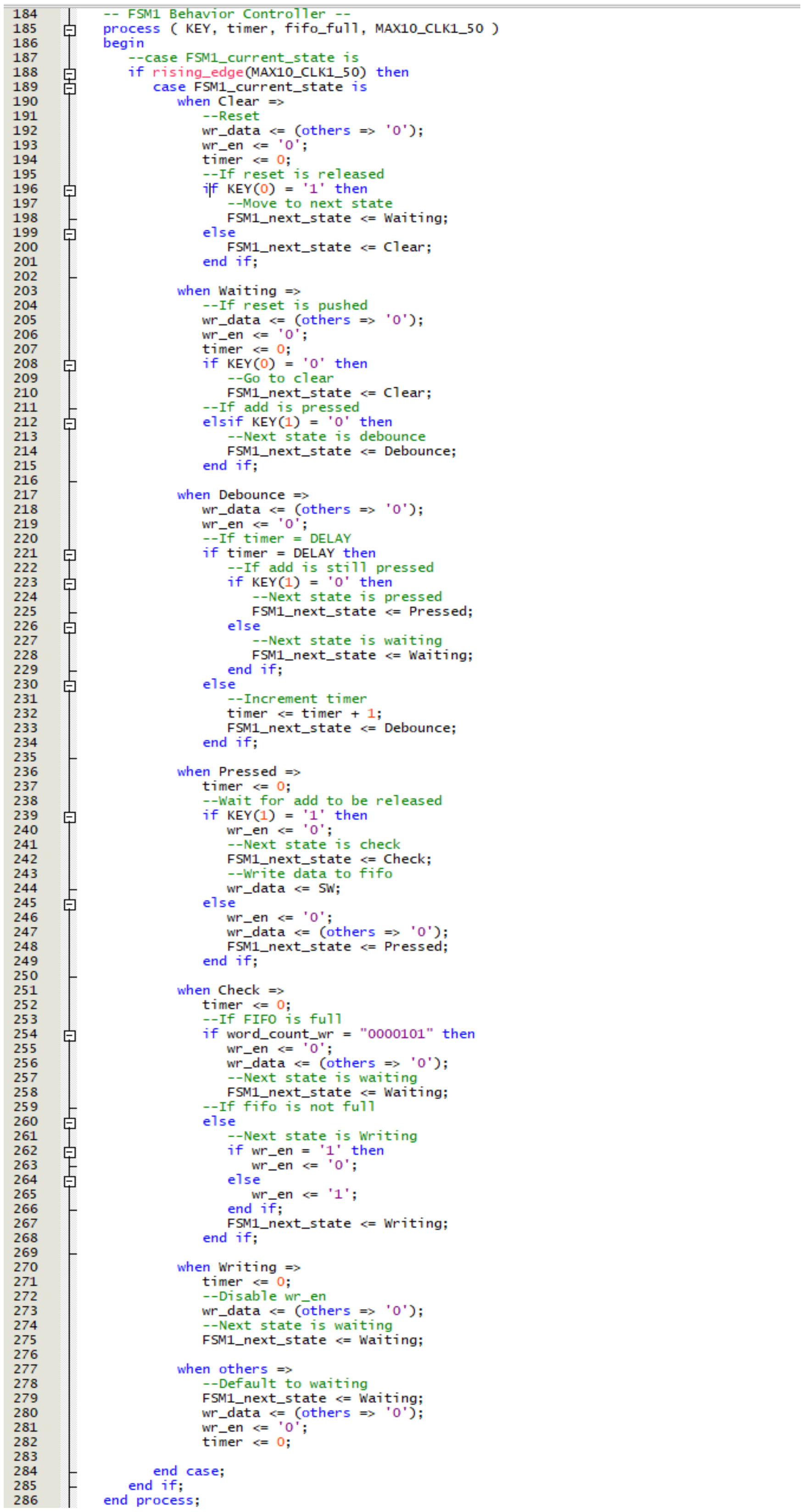
```vhdl
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity fifo_accumulator is

    generic(
        --      Add generics here     --
        -- NAME : TYPE := DEFAULT_VALUE (separated by ; ) --
        DELAY : integer  := 2500000    -- Number of clock cycles for debounce

    );

    port (
            -- Declare module ports here --
        -- NAME : DIRECTION TYPE (separated by ; ) --

        -- CLK input --
        -- ADC_CLK_10   : in std_logic; -- 10 MHz
        MAX10_CLK1_50   : in std_logic; -- 50 MHz 1
        -- MAX10_CLK2_50   : in std_logic; -- 50 MHz 2

        -- Button input --
        KEY : in std_logic_vector (1 downto 0);

        -- 7-Segment output --
        HEX0 : out std_logic_vector(7 downto 0);
        HEX1 : out std_logic_vector(7 downto 0);
        HEX2 : out std_logic_vector(7 downto 0);
        HEX3 : out std_logic_vector(7 downto 0);
        HEX4 : out std_logic_vector(7 downto 0);
        HEX5 : out std_logic_vector(7 downto 0);

        -- Switches --
        SW : in std_logic_vector(9 downto 0);

        -- LEDs --
        LEDR : out std_logic_vector(9 downto 0)
    );

end entity fifo_accumulator;

architecture behavioral of fifo_accumulator is

    -- Components --
    -- FIFO --
    component accum_FIFO IS
        PORT
        (
            aclr        : IN STD_LOGIC   := '0';
            data        : IN STD_LOGIC_VECTOR (9 DOWNTO 0);
            rdclk       : IN STD_LOGIC ;
            rdreq       : IN STD_LOGIC ;
            wrclk       : IN STD_LOGIC ;
            wrreq       : IN STD_LOGIC ;
            q           : OUT STD_LOGIC_VECTOR (9 DOWNTO 0);
            rdempty     : OUT STD_LOGIC ;
            rdusedw     : OUT STD_LOGIC_VECTOR (6 DOWNTO 0);
            wrusedw     : OUT STD_LOGIC_VECTOR (6 DOWNTO 0)
        );
    END component;

    -- PLL --
    component my_PLL IS
        PORT
        (
            areset      : IN STD_LOGIC   := '0';
            inclk0      : IN STD_LOGIC   := '0';
            c0          : OUT STD_LOGIC ;
            c1          : OUT STD_LOGIC ;
            locked      : OUT STD_LOGIC
        );
    END component;

    -- Variables
    signal add      : integer;
    signal sum      : unsigned(23 downto 0);    -- Accumulated total
    signal timer    : integer := 0;             -- Timer for debounce

    --7-segment display
    type SEVEN_SEG is array (0 to 15) of std_logic_vector(7 downto 0); -- Define new type for lookup table
    constant table : SEVEN_SEG := (
                X"C0", X"F9", X"A4", X"B0",  -- 0, 1, 2, 3
                X"99", X"92", X"82", X"F8",  -- 4, 5, 6, 7
                X"80", X"90", X"88", X"83",  -- 8, 9, A, B
                X"C6", X"A1", X"86", X"8E"); -- C, D, E, F

    -- FSM States
    type state_type is (
        Clear,
        Waiting,
        Debounce,
        Pressed,
        Check,
        Writing,
        Empty,
        Accumulate,
        Display
        );
```

Figure 1: FIFO_Accumulator.vhd Pt 1



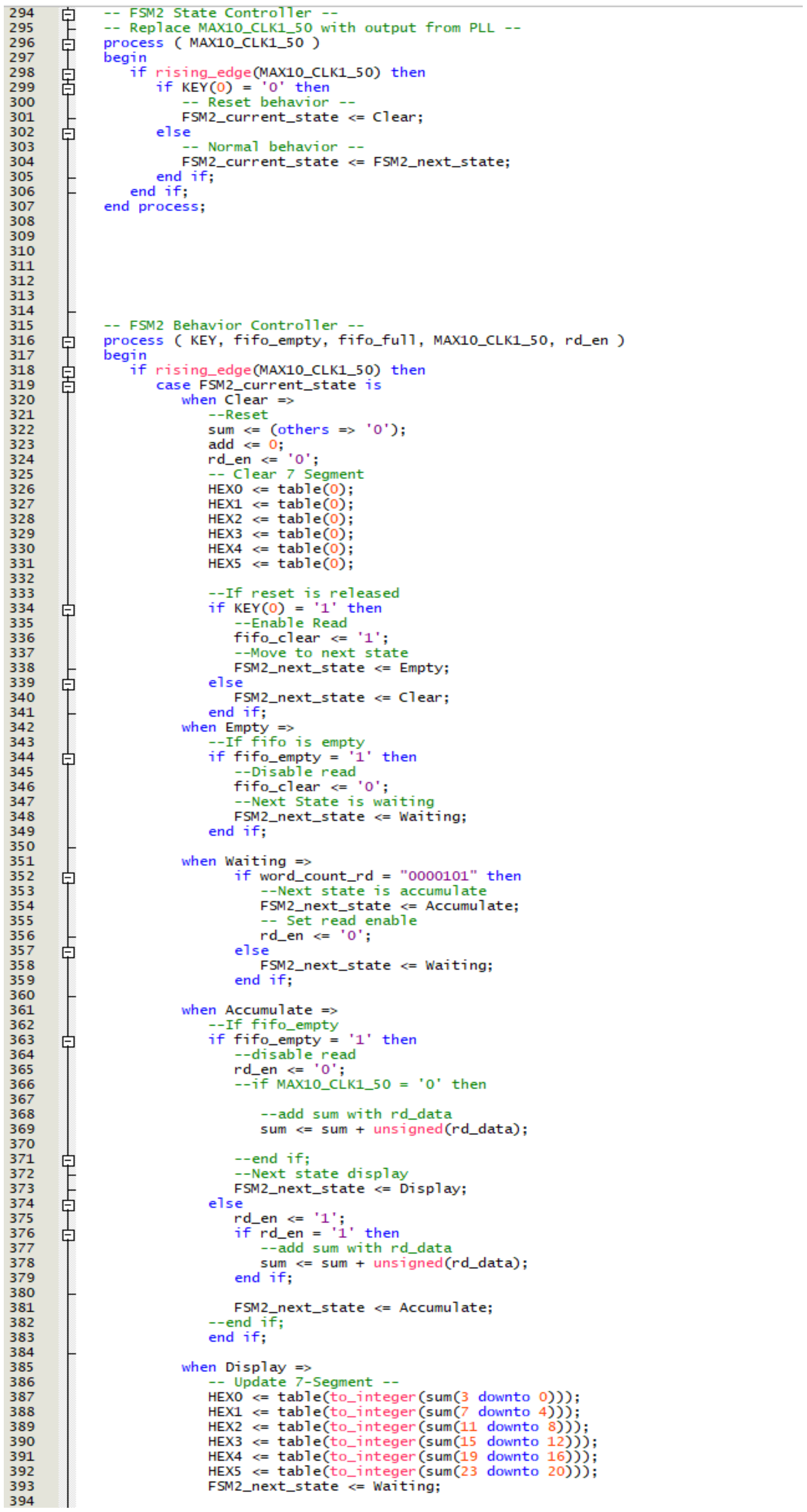
```vhdl

   signal FSM1_current_state, FSM1_next_state, FSM2_current_state, FSM2_next_state: state_type;

   -- PLL Signals --
   signal pll_Reset     : std_logic := '0';  -- Reset PLL
   signal pll_outCLK1   : std_logic;         -- PLL output clock 1
   signal pll_outCLK2   : std_logic;         -- PLL output clock 2
   signal pll_locked    : std_logic;         -- PLL Locked

   -- FSM1 Signals --
   signal wr_en         : std_logic;                     -- Enable write to FIFO
   signal wr_data       : std_logic_vector(9 downto 0);  -- Data to write to FIFO
   signal fifo_full     : std_logic;                     -- Binary signal flag for fifo is full (also used in FSM2)
   signal word_count_wr : std_logic_vector(6 downto 0);  -- How many words are used write side

   -- FSM2 Signals --
   signal rd_data       : std_logic_vector(9 downto 0);  -- Data to read from FIFO
   signal fifo_empty    : std_logic;                     -- Binary signal flag for fifo is empty
   signal rd_en         : std_logic;                     -- Enable read from FIFO
   signal fifo_clear    : std_logic;                     -- Set to clear FIFO
   signal word_count_rd : std_logic_vector(6 downto 0);  -- How many words are used read side

   -- Misc FIFO Signals --
   signal aclr_sig : STD_LOGIC;


begin

   -- Instantiate FIFO IP --
   accum_FIFO_inst : accum_FIFO PORT MAP (
      aclr     => aclr_sig,       -- Ansychronous clear
      data     => wr_data,        -- Data into FIFO
      rdclk    => MAX10_CLK1_50,  -- Change to PLL output FMS2
      rdreq    => rd_en,          -- Read acknowledge
      wrclk    => MAX10_CLK1_50,  -- Change to PLL output FMS1
      wrreq    => wr_en,          -- Write enable
      q        => rd_data,        -- Data out of FIFO
      rdempty  => fifo_empty,     -- Read side empty
      rdusedw  => word_count_rd,  -- Read side word count
      wrusedw  => word_count_wr   -- Write side word count
   );


   -- Instantiate PLL IP --
   my_PLL_inst : my_PLL
      PORT MAP (
         areset   => pll_Reset,
         inclk0   => MAX10_CLK1_50,
         c0       => pll_outCLK1,
         c1       => pll_outCLK2,
         locked   => pll_locked
      );

   -- Asynchronous Clear of FIFO --
   process ( KEY(0) )
   begin
      if (KEY(0) = '0') then
         aclr_sig <= '1';
      else
         aclr_sig <= '0';
      end if;
   end process;

   -- FSM1 State Controller --
   -- Replace MAX10_CLK1_50 with output from PLL --
   process ( MAX10_CLK1_50 )
   begin
      if rising_edge(MAX10_CLK1_50) then
         if KEY(0) = '0' then
            -- Reset behavior --
            FSM1_current_state <= Clear;
         else
            -- Normal behavior --
            FSM1_current_state <= FSM1_next_state;
         end if;
      end if;
   end process;




   
 
```

Figure 2: FIFO_Accumulator.vhd Pt 2



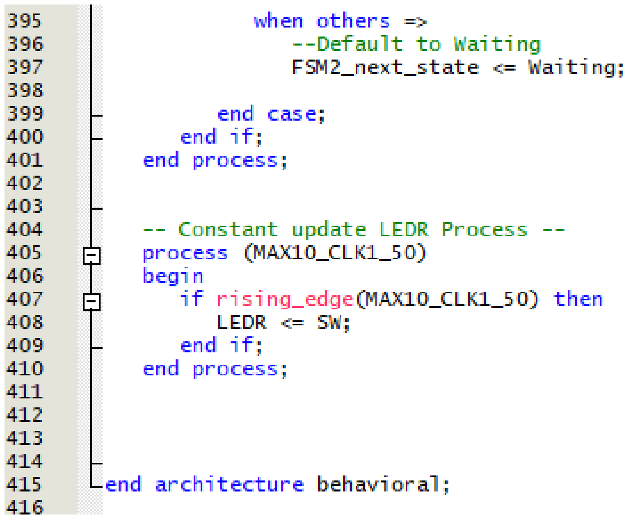
```vhdl
-- FSM1 Behavior Controller --
process ( KEY, timer, fifo_full, MAX10_CLK1_50 )
begin
   --case FSM1_current_state is
   if rising_edge(MAX10_CLK1_50) then
      case FSM1_current_state is
         when Clear =>
            --Reset
            wr_data <= (others => '0');
            wr_en <= '0';
            timer <= 0;
            --If reset is released
            if KEY(0) = '1' then
               --Move to next state
               FSM1_next_state <= Waiting;
            else
               FSM1_next_state <= Clear;
            end if;

         when Waiting =>
            --If reset is pushed
            wr_data <= (others => '0');
            wr_en <= '0';
            timer <= 0;
            if KEY(0) = '0' then
               --Go to clear
               FSM1_next_state <= Clear;
            --If add is pressed
            elsif KEY(1) = '0' then
               --Next state is debounce
               FSM1_next_state <= Debounce;
            end if;

         when Debounce =>
            wr_data <= (others => '0');
            wr_en <= '0';
            --If timer = DELAY
            if timer = DELAY then
               --If add is still pressed
               if KEY(1) = '0' then
                  --Next state is pressed
                  FSM1_next_state <= Pressed;
               else
                  --Next state is waiting
                  FSM1_next_state <= Waiting;
               end if;
            else
               --Increment timer
               timer <= timer + 1;
               FSM1_next_state <= Debounce;
            end if;

         when Pressed =>
            timer <= 0;
            --Wait for add to be released
            if KEY(1) = '1' then
               wr_en <= '0';
               --Next state is check
               FSM1_next_state <= Check;
               --Write data to fifo
               wr_data <= SW;
            else
               wr_en <= '0';
               wr_data <= (others => '0');
               FSM1_next_state <= Pressed;
            end if;

         when Check =>
            timer <= 0;
            --If FIFO is full
            if word_count_wr = "0000101" then
               wr_en <= '0';
               wr_data <= (others => '0');
               --Next state is waiting
               FSM1_next_state <= Waiting;
            --If fifo is not full
            else
               --Next state is Writing
               if wr_en = '1' then
                  wr_en <= '0';
               else
                  wr_en <= '1';
               end if;
               FSM1_next_state <= Writing;
            end if;

         when Writing =>
            timer <= 0;
            --Disable wr_en
            wr_data <= (others => '0');
            --Next state is waiting
            FSM1_next_state <= Waiting;

         when others =>
            --Default to waiting
            FSM1_next_state <= Waiting;
            wr_data <= (others => '0');
            wr_en <= '0';
            timer <= 0;

      end case;
   end if;
end process;
```

Figure 3: FIFO_Accumulator.vhd Pt 3



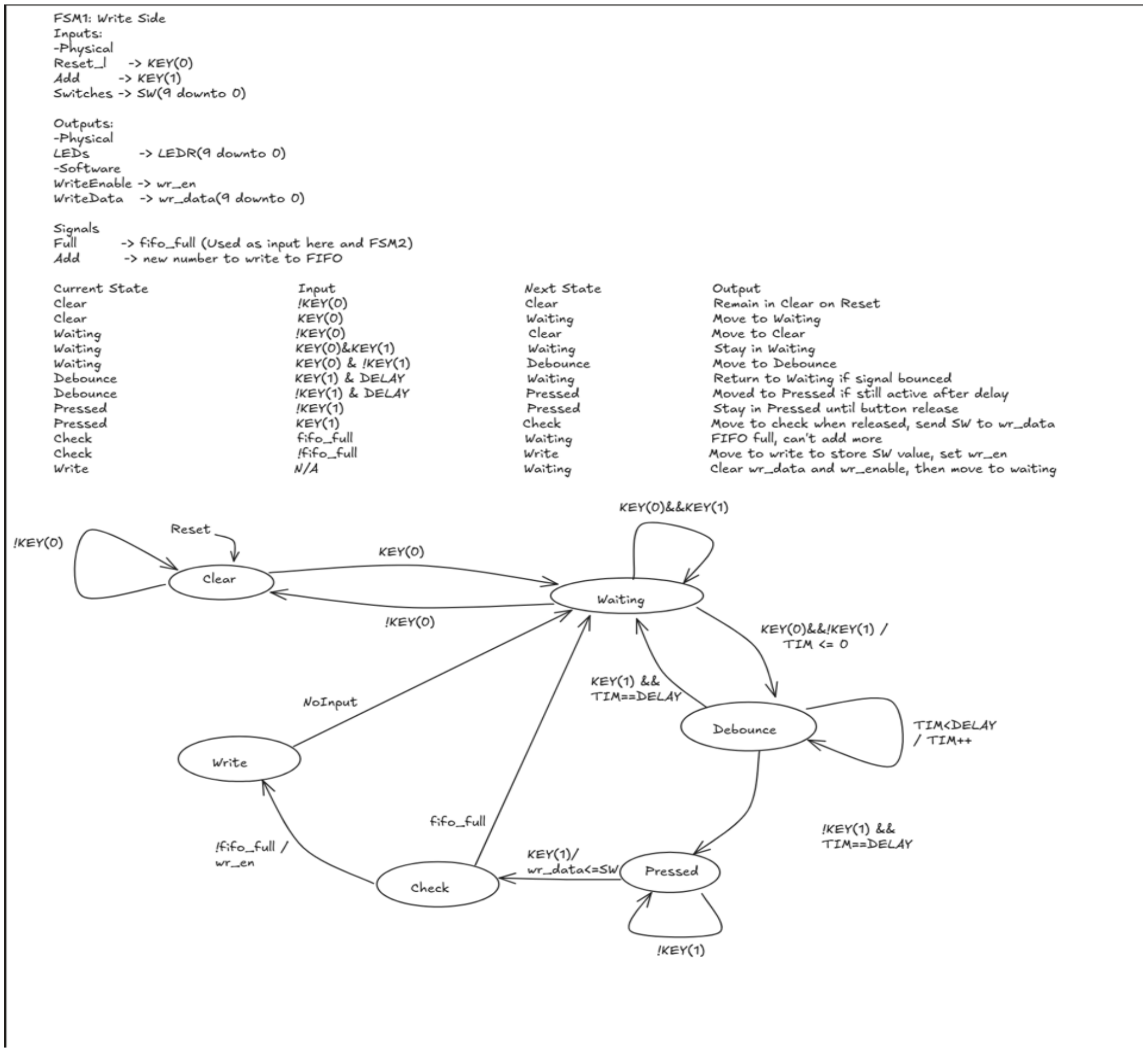
```vhdl
-- FSM2 State Controller --
-- Replace MAX10_CLK1_50 with output from PLL --
process ( MAX10_CLK1_50 )
begin
   if rising_edge(MAX10_CLK1_50) then
      if KEY(0) = '0' then
         -- Reset behavior --
         FSM2_current_state <= Clear;
      else
         -- Normal behavior --
         FSM2_current_state <= FSM2_next_state;
      end if;
   end if;
end process;







-- FSM2 Behavior Controller --
process ( KEY, fifo_empty, fifo_full, MAX10_CLK1_50, rd_en )
begin
   if rising_edge(MAX10_CLK1_50) then
      case FSM2_current_state is
         when Clear =>
            --Reset
            sum <= (others => '0');
            add <= 0;
            rd_en <= '0';
            -- Clear 7 Segment
            HEX0 <= table(0);
            HEX1 <= table(0);
            HEX2 <= table(0);
            HEX3 <= table(0);
            HEX4 <= table(0);
            HEX5 <= table(0);

            --If reset is released
            if KEY(0) = '1' then
               --Enable Read
               fifo_clear <= '1';
               --Move to next state
               FSM2_next_state <= Empty;
            else
               FSM2_next_state <= Clear;
            end if;
         when Empty =>
            --If fifo is empty
            if fifo_empty = '1' then
               --Disable read
               fifo_clear <= '0';
               --Next State is waiting
               FSM2_next_state <= Waiting;
            end if;

         when Waiting =>
               if word_count_rd = "0000101" then
                  --Next state is accumulate
                  FSM2_next_state <= Accumulate;
                  -- Set read enable
                  rd_en <= '0';
               else
                  FSM2_next_state <= Waiting;
               end if;

         when Accumulate =>
            --If fifo_empty
            if fifo_empty = '1' then
               --disable read
               rd_en <= '0';
               --if MAX10_CLK1_50 = '0' then

                  --add sum with rd_data
                  sum <= sum + unsigned(rd_data);

               --end if;
               --Next state display
               FSM2_next_state <= Display;
            else
               rd_en <= '1';
               if rd_en = '1' then
                  --add sum with rd_data
                  sum <= sum + unsigned(rd_data);
               end if;

               FSM2_next_state <= Accumulate;
            --end if;
            end if;

         when Display =>
            -- Update 7-Segment --
            HEX0 <= table(to_integer(sum(3 downto 0)));
            HEX1 <= table(to_integer(sum(7 downto 4)));
            HEX2 <= table(to_integer(sum(11 downto 8)));
            HEX3 <= table(to_integer(sum(15 downto 12)));
            HEX4 <= table(to_integer(sum(19 downto 16)));
            HEX5 <= table(to_integer(sum(23 downto 20)));
            FSM2_next_state <= Waiting;
```

Figure 4: FIFO_Accumulator.vhd Pt 4



```vhdl
                when others =>
                    --Default to Waiting
                    FSM2_next_state <= Waiting;

            end case;
        end if;
    end process;


    -- Constant update LEDR Process --
    process (MAX10_CLK1_50)
    begin
        if rising_edge(MAX10_CLK1_50) then
            LEDR <= SW;
        end if;
    end process;




end architecture behavioral;

```

Figure 5: FIFO_Accumulator.vhd Pt 5



FSM1: Write Side
Inputs:
-Physical
Reset_l -> KEY(0)
Add -> KEY(1)
Switches -> SW(9 downto 0)

Outputs:
-Physical
LEDs -> LEDR(9 downto 0)
-Software
WriteEnable -> wr_en
WriteData -> wr_data(9 downto 0)

Signals
Full -> fifo_full (Used as input here and FSM2)
Add -> new number to write to FIFO

Current State	Input	Next State	Output
Clear	!KEY(0)	Clear	Remain in Clear on Reset
Clear	KEY(0)	Waiting	Move to Waiting
Waiting	!KEY(0)	Clear	Move to Clear
Waiting	KEY(0)&KEY(1)	Waiting	Stay in Waiting
Waiting	KEY(0) & !KEY(1)	Debounce	Move to Debounce
Debounce	KEY(1) & DELAY	Waiting	Return to Waiting if signal bounced
Debounce	!KEY(1) & DELAY	Pressed	Moved to Pressed if still active after delay
Pressed	!KEY(1)	Pressed	Stay in Pressed until button release
Pressed	KEY(1)	Check	Move to check when released, send SW to wr_data
Check	fifo_full	Waiting	FIFO full, can't add more
Check	!fifo_full	Write	Move to write to store SW value, set wr_en
Write	N/A	Waiting	Clear wr_data and wr_enable, then move to waiting

Figure 6: FIFO_Accumulator FSM1

FSM2: Read Side

Inputs

ReadData	-> rd_data(23 downto 0)
Empty	-> fifo_empty

Outputs

-Physical

7-Segment	-> HEX(5 downto 0)

-Software

ReadEnable	-> rd_en

Signals

Full	-> fifo_full (Used as input here and FSM1)
Total	-> sum (Running total of accumulated value)

Current State	Input	Next State	Output
Clear	!KEY(0)	Clear	Remain in Clear, clear sum, clear HEX output
Clear	KEY(0)	Empty	Move to Empty, set rd_en
Empty	!fifo_empty	Empty	Stay in Empty
Empty	fifo_empty	Waiting	Move to waiting when FIFO empty, reset rd_en
Waiting	!KEY(0)	Clear	Move to clear on Reset
Waiting	KEY(0)&fifo_full	Accumulate	Move to Accumulate when FIFO full, set rd_en
Accumulate	!fifo_empty	Accumulate	Remain in Accumulate until empty, adding data from rd_data to sum
Accumulate	fifo_empty	Display	Move to display, reset rd_en
Display	N/A	Waiting	Move to waiting, store sum in HEX output

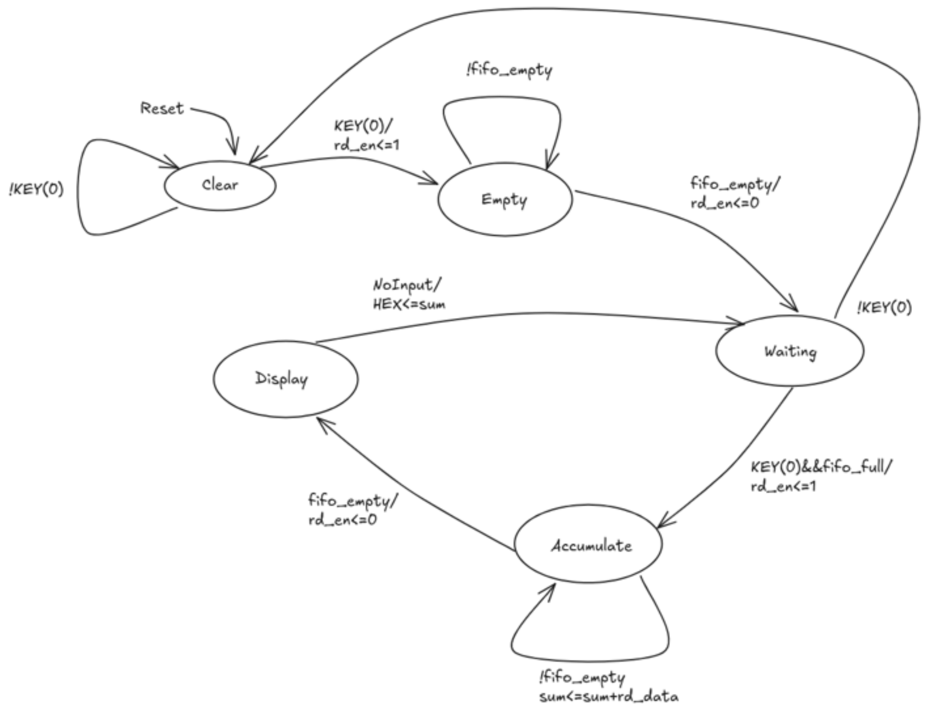


Figure 7: FIFO_Accumulator FSM2