Lab 6 - FIFO

Nate Herbst
A02307138
Nathan Walker
A02364124

Introduction

This lab builds on our previous work with the 24-bit accumulator by implementing a FIFO-based storage and read system. The project required designing and integrating two finite state machines (FSMs) on a DE10-Lite board to handle separate read and write operations across two clock domains using a clock-domain-crossing FIFO. Key tasks included resetting and accumulating values via push buttons, debouncing the "add" button, and reading the FIFO's input from the 10 toggle switches. Additionally, the 10 LEDs were used to reflect the state of the switches.

Procedure

Initial Setup:

- We began by setting up the design based on our previous accumulator implementation. Our goal was to implement two FSMs: one for writing values to the FIFO and another for reading and accumulating these values.
- We connected the accumulator's 24-bit output to the six 7-segment displays for hexadecimal output and used 10 toggle switches for input.

Challenges and Solutions:

- **FIFO Configuration Issue:** When testing the design, we observed that the FIFO did not behave as expected. Specifically, it consistently added a value of 1 to the accumulator after five button presses, regardless of switch states.
    - **Solution**: We traced this to a configuration error with the FIFO in the MegaWizard Plug-In Manager. The FIFO was not set as asynchronous, which led to clock-domain crossing issues. By setting the FIFO to asynchronous mode, we synchronized the two FSMs running on different clocks, ensuring consistent behavior.
- **FSM Modification for Timing:** We initially implemented a basic read FSM, but testing revealed timing issues where values were not being stored accurately.
    - **Solution**: To fix this, we added additional states to the read FSM to allow it to wait up to two clock cycles, ensuring the FIFO correctly handled the clock transitions and values were accurately iterated.

- **Syntax Errors in VHDL Modules:** After creating additional modules for FIFO control, we encountered several syntax errors, which required debugging and fixing.
    - **Solution**: We corrected these syntax errors by systematically reviewing each module's structure and ensuring consistency in signal and variable declarations across modules.

State Machine Transition Errors:

- Another issue was that the `output` signal of the debouncer did not hold the correct value for a sufficient duration after the button was pressed. The `RELEASED` state did not assert the `output` signal long enough to be detected reliably.
- **Solution:** We adjusted the state machine so that in the `RELEASED` state, the `output` was asserted for one clock cycle before returning to the `WAITING` state. This ensured that the button press was properly registered by the accumulator.

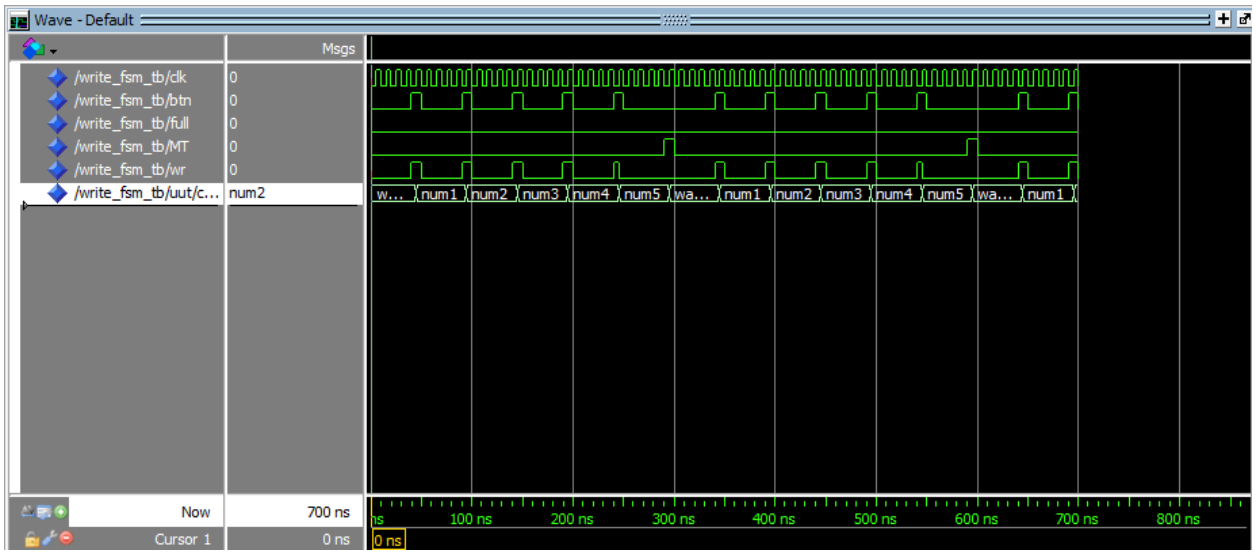
Results

Conclusion

This lab taught us about managing clock-domain crossing issues and the importance of careful FIFO configuration. Although we encountered challenges with FIFO synchronization and state management, adjusting the FIFO's settings and modifying the read FSM allowed us to meet the project requirements. These troubleshooting steps provided valuable insights into hardware timing and synchronization in multi-clock systems.

Appendix

`write_fsm` Simulation



Lab6_FIFO.vhd top level file

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

entity Lab6_FIFO is
   port (
       ADC_CLK_10 : in std_logic;
       MAX10_CLK1_50 : in std_logic;
       MAX10_CLK2_50 : in std_logic;

       KEY : in std_logic_vector(1 downto 0);
       SW : in std_logic_vector(9 downto 0);

       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);
```

```
        LEDR : out std_logic_vector(9 downto 0)
    );
end Lab6_FIFO;

architecture component_list of Lab6_FIFO is

    component 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);
            rdusedw     : OUT STD_LOGIC_VECTOR (2 DOWNTO 0);
            wrusedw     : OUT STD_LOGIC_VECTOR (2 DOWNTO 0)
            );
    end component fifo;

    component PLL is
        port (
            inclk0      : IN STD_LOGIC  := '0';
            c0      : OUT STD_LOGIC ;
            c1      : OUT STD_LOGIC
        );
    end component PLL;

    component write_fsm is
        port (
            clk : in std_logic;
            btn : in std_logic;
            full : in std_logic;
            MT : in std_logic;
            wr_en : out std_logic
        );
    end component write_fsm;

    component read_fsm is
```

```
    port (
        clk  : in std_logic;
        full : in std_logic;
        MT   : in std_logic;
        rd   : out std_logic;
        en   : out std_logic;
        clr  : out std_logic
        );
end component read_fsm;

component debouncer
    port (
        clk : in std_logic;
        btn : in std_logic;
        output : out std_logic
    );
end component debouncer;

component HEX_seven_seg_disp_6
    port (
        IN0 : in std_logic_vector(3 downto 0);
        IN1 : in std_logic_vector(3 downto 0);
        IN2 : in std_logic_vector(3 downto 0);
        IN3 : in std_logic_vector(3 downto 0);
        IN4 : in std_logic_vector(3 downto 0);
        IN5 : in std_logic_vector(3 downto 0);
        clk : in  std_logic;
        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)
    );
end component HEX_seven_seg_disp_6;

component accumulator
    generic (
        N : integer := 10;
        M : integer := 10
```

```
    );
        port (
        en : in std_logic;
        rst : in std_logic;
        clk : in std_logic;
        input : in std_logic_vector(N-1 downto 0);
        sum : out std_logic_vector(M-1 downto 0)
    );
  end component accumulator;

  signal rst : std_logic;
  signal sum : std_logic_vector(23 downto 0);

  signal key0_l : std_logic;
  signal key1_l : std_logic;

  signal pressed : std_logic;

  signal wr_clk : std_logic;
  signal rd_clk : std_logic;

  signal full : std_logic;
  signal MT : std_logic;

  signal q_sig : std_logic_vector(9 downto 0);

  signal en : std_logic;
  signal rd_en : std_logic;
  signal wr_en : std_logic;

  signal counter : integer;

  signal rdusedw_sig : std_logic_vector(2 downto 0);
  signal wrusedw_sig : std_logic_vector(2 downto 0);

  signal aclr_sig : std_logic;

begin

  fifo_inst : fifo
```

```
    port map (
        aclr     => aclr_sig,
        data     => SW,
        rdclk    => rd_clk,
        rdreq    => rd_en,
        wrclk    => wr_clk,
        wrreq    => wr_en,
        q    => q_sig,
        rdusedw  => rdusedw_sig,
        wrusedw  => wrusedw_sig
        );

PLL1 : PLL
    port map (
        inclk0 => MAX10_CLK2_50,
        c0 => rd_clk, --12.5 MHZ
        c1 => wr_clk -- 5 MHz
    );

fsm1 : write_fsm
    port map (
        clk => wr_clk,
        btn => pressed,
        full => full,
        MT => MT,
        wr_en => wr_en
    );

fsm2 : read_fsm
    port map (
        clk => rd_clk,
        full => full,
        MT => MT,
        rd => rd_en,
        en => en,
        clr => aclr_sig
    );

accumulator1 : accumulator
    generic map (
```

```
        N => 10,
        M => 24
    )
    port map (
        clk => rd_clk,
        en => en,
        rst => rst,
        input => q_sig,
        sum => sum
    );

display : HEX_seven_seg_disp_6
    port map (
        clk => rd_clk,
        IN0 => sum(3 downto 0),
        IN1 => sum(7 downto 4),
        IN2 => sum(11 downto 8),
        IN3 => sum(15 downto 12),
        IN4 => sum(19 downto 16),
        IN5 => sum(23 downto 20),
        HEX0 => HEX0,
        HEX1 => HEX1,
        HEX2 => HEX2,
        HEX3 => HEX3,
        HEX4 => HEX4,
        HEX5 => HEX5
    );

rst_btn : debouncer
    port map (
        clk => rd_clk,
        btn => key0_l,
        output => rst
    );

en_btn : debouncer
    port map (
        clk => wr_clk,
        btn => key1_l,
        output => pressed
```

```
        );

    LEDR <= SW;

    key0_l <= not KEY(0);
    key1_l <= not KEY(1);

    full <= '1' when (wrusedw_sig = "101") else '0';
    MT <= '1' when (wrusedw_sig = "000") else '0';

end component_list;
```

write_fsm.vhd

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

entity write_fsm is
    port (
        clk : in std_logic;
        btn : in std_logic;
        full : in std_logic;
        MT : in std_logic;
        wr_en : out std_logic
    );
end write_fsm;

architecture states of write_fsm is
    type state_type is (waiting, num1, num2, num3, num4, num5);
    signal current_state, next_state : state_type;
begin

    process(clk) begin
        if rising_edge(clk) then
            current_state <= next_state;
        end if;
    end process;
```

```
process (current_state, btn, full, MT)
begin
    case current_state is
        when waiting =>
            wr_en <= '0';
            if btn = '1' and full = '0' then
                wr_en <= '1';
                next_state <= num1;
            else
                wr_en <= '0';
                next_state <= waiting;
            end if;

        when num1 =>
            wr_en <= '0';
            if btn = '1' and full = '0' then
                wr_en <= '1';
                next_state <= num2;
            else
                wr_en <= '0';
                next_state <= num1;
            end if;

        when num2 =>
            wr_en <= '0';
            if btn = '1' and full = '0' then
                wr_en <= '1';
                next_state <= num3;
            else
                wr_en <= '0';
                next_state <= num2;
            end if;

        when num3 =>
            wr_en <= '0';
            if btn = '1' and full = '0' then
                wr_en <= '1';
                next_state <= num4;
            else
                wr_en <= '0';
```

```
                next_state <= num3;
            end if;

        when num4 =>
            wr_en <= '0';
            if btn = '1' and full = '0' then
                wr_en <= '1';
                next_state <= num5;
            else
                wr_en <= '0';
                next_state <= num4;
            end if;

        when num5 =>
            wr_en <= '0';
            if MT = '1' then
                next_state <= waiting;
            else
                wr_en <= '0';
                next_state <= num5;
            end if;
    end case;
  end process;
end states;
```

Read_fsm.vhd

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

entity read_fsm is
  port (
      clk  : in std_logic;
      full : in std_logic;
      MT   : in std_logic;
      rd   : out std_logic;
      en   : out std_logic;
      clr  : out std_logic
```

```
    );
end read_fsm;

architecture states of read_fsm is
    type state_type is (idle, reading_fifo, T1, T2);
    signal current_state, next_state : state_type;
begin

    -- State Transition Process
    process(clk)
    begin
        if rising_edge(clk) then
            current_state <= next_state;
        end if;
    end process;

    -- Next State Logic and Output Process
    process(current_state, full, MT)
    begin
        case current_state is
            when idle =>
                rd <= '0';
                en <= '0';
                clr <= '0';
                if full = '1' then
                    rd <= '1';
                    en <= '1';
                    next_state <= reading_fifo;
                else
                    next_state <= idle;
                end if;

            when T1 =>
                rd <= '0';
                en <= '0';
                clr <= '0';
                next_state <= T2;
            when T2 =>
                rd <= '0';
                en <= '0';
```

```
                clr <= '0';
                next_state <= reading_fifo;

            when reading_fifo =>
                rd <= '1';
                en <= '1';
                clr <= '0';
                if MT = '1' then
                    rd <= '0';
                    en <= '0';
                    clr <= '1';
                    next_state <= idle;
                else
                    next_state <= reading_fifo;
                end if;

            when others =>
                next_state <= idle;  -- default to idle
        end case;
    end process;
end states;
```