Calculator

Lab 3

Matteo Golinelli, Nicolò Vantini, Marcus Gregory

3 December 2025

1 Introduction

The goal of this project is to design and implement a simple calculator with an accumulator on an FPGA board. The calculator performs basic arithmetic operations: addition, subtraction multiplication and division using an Arithmetic Logic Unit (ALU). The user provides the first operand using the switches on the FPGA, while the second operand is stored internally in an accumulator that holds the result of the previous computation. The output value is presented on the FPGA's 7-segment display.

2 Hardware Components and ports

- FPGA development board (Basys 3)
  - Switches, for entering operand (see Callout 5 of Fig.1)
  - Push-buttons; to select addition/subtraction/multiplication/division/reset (see Callout 7 of Fig.1)
  - LEDs; to display raw binary data from early design phase (see Callout 6 of Fig.1)
  - 7-segment display; for displaying results (see Callout 4 of Fig.1)
- Vivado Design Suite; for VHDL design, simulation, implementation, and bitstream generation

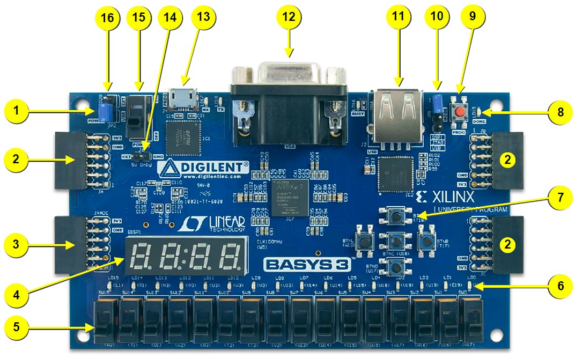

Figure 1. Basys 3 FPGA board with callouts.

Callout	Component Description	Callout	Component Description
1	Power good LED	9	FPGA configuration reset button
2	Pmod port(s)	10	Programming mode jumper
3	Analog signal Pmod port (XADC)	11	USB host connector
4	Four digit 7-segment display	12	VGA connector
5	Slide switches (16)	13	Shared UART/ JTAG USB port
6	LEDs (16)	14	External power connector
7	Pushbuttons (5)	15	Power Switch
8	FPGA programming done LED	16	Power Select Jumper

Figure 1: Basys 3 Layout

3 Components

The ALU (Arithmetic Logic Unit) and the 7-Segment Display Driver entities are the ones provided by professor Velha. Both the original files have been modified to include small changes and solve minor bugs. All the components have been successfully tested against the provided testbenches.

3.1 Debouncer

The debouncer was the central focus in the project. This component is necessary because analog buttons always introduce small oscillations in the signal they produce when pressed or released. A debouncer takes a noisy button input, *bouncy*, and converts it to a single pulse once the input is stable for a set amount of time. If between each clock cycle the input remains stable a counter is decremented, otherwise the counter is reset to all 1's. When the counter reaches zero, it is reset and on the next cycle the output is that of the input for only one clock cycle. Since the input is a button and we only need pulses to initialize the following operations, we did not need to debounce the null state.
The amount of bits for the counter have been changed after some testing. In our design, the counter restarts after the output has been signalled, so holding down the button results in multiple *pulse* outputs, which triggers repeated operations from the board. Future improvements may solve this issue by restarting the counter only after a change in the *bounce* input.

3.2 Accumulator

The accumulator is a synchronous 16 bit parallel register with low reset, high init and enable signals. We used it to hold the result of the previous operation of the calculator, in order to provide the second operand to the ALU. The accumulator is always initialized to zero when the calculator is started.

3.3 Calculator

Finally, each entity is connected with the others by a general entity, which binds the signals of different components. All the needed signals are initialized and then connected to the different components, following the diagram in Fig.2. The calculator contains a process which drives the accumulator signals and set the operation inputs for the ALU based on the buttons input.

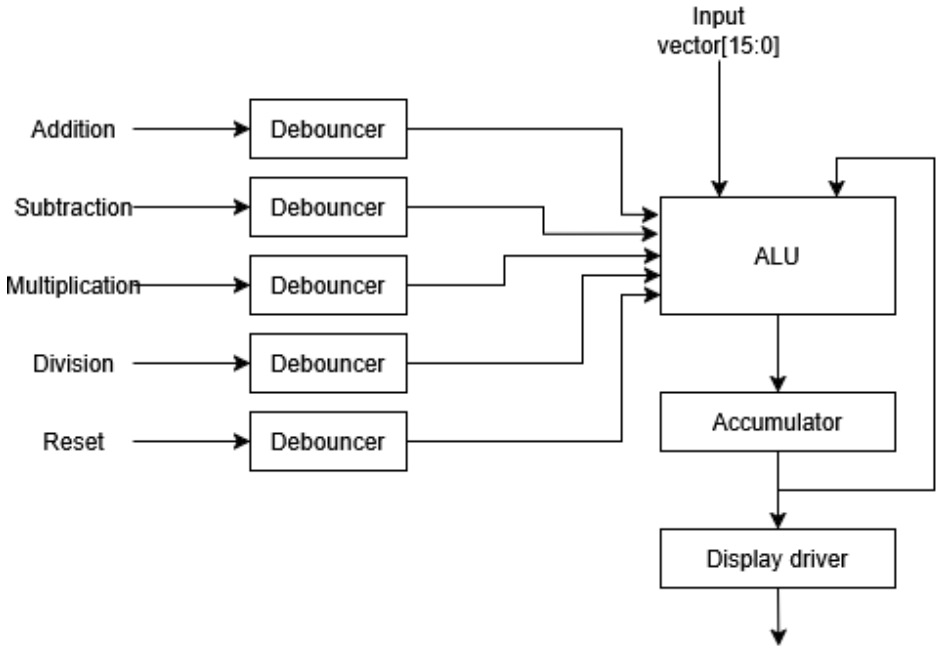

Figure 2: Calculator diagram

4 VHDL Code

4.1 Accumulator

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

entity Accumulator is
    port (
        clk         : in  std_logic;
        rst         : in  std_logic; -- Low active reset
        ac_init       : in  std_logic; -- High active init
        ac_enable     : in  std_logic; -- High active enable
        data_in     : in  signed(15 downto 0);
        result_out  : out signed(15 downto 0)
    );
end entity Accumulator;

architecture Behavioral of Accumulator is
begin
    process(clk, rst)
    begin
        if rst = '0' then
            result_out <= (others => '0');
        elsif rising_edge(clk) then
            -- Since not all the combinations are exploited, the VHDL simulator
            ↪  creates a memory element which stores the previous value
            if ac_init = '1' then
                result_out <= (others => '0');
            elsif ac_enable = '1' then
                result_out <= data_in;
            end if;
        end if;
    end process;
end architecture Behavioral;
```

4.2 Debouncer

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

entity Debouncer is
    generic(
        counter_size : integer := 23
    );
    port (
        clk         : in  std_logic;
        rst         : in  std_logic; -- Low active reset
        bouncy    : in  std_logic;
        pulse   : out std_logic
    );
end entity Debouncer;

architecture Behavioral of Debouncer is
    signal counter      : std_logic_vector(counter_size-1 downto 0) := (others =>
    ↪  '1');
    signal prev_state   : std_logic := '0';
    signal outputting   : std_logic := '0';
begin
    process(clk, rst)
    begin
        if rst = '0' then
            counter    <= (others => '1');
            prev_state <= '0';
            pulse  <= '0';
            outputting <= '0';
        elsif rising_edge(clk) then
            if outputting = '1' then
                -- Output has been updated (just for one clock), wait for next
                ↪  change
                outputting <= '0';
                counter <= (others => '1');
                pulse <= '0';
            elsif bouncy /= prev_state then
                -- Button state changed, reset counter
                counter <= (others => '1');
                prev_state <= bouncy;
            else
                if unsigned(counter) > 0 then
                    counter <= std_logic_vector(unsigned(counter) - 1);
                else
                    -- Counter has expired, update clean output
                    pulse <= prev_state;
                    outputting <= '1';
                end if;
            end if;
        end if;
    end process;
```

```
end architecture Behavioral;
```

4.3 Calculator

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.NUMERIC_STD.ALL;



entity Calculator is
    Port (
        clock : in std_logic;
        reset : in std_logic;
        SW : in std_logic_vector( 15 downto 0 );
        BTNC, BTNU, BTNL, BTNR, BTND : in std_logic;
        LED : out std_logic_vector( 15 downto 0 );
        CA, CB, CC, CD, CE, CF, CG, DP : out std_logic;
        AN : out std_logic_vector( 3 downto 0 )
    );
end Calculator;

architecture Behavioral of Calculator is

-- Internal signals for debouncers
    signal center_edge, up_edge, left_edge, right_edge, down_edge : std_logic;
    -- Input/output signals for accumulator
    signal acc_in, acc_out : signed( 15 downto 0 );
    -- Init and load signals for accumulator
    signal acc_init, acc_enable : std_logic;
    -- Control signals for ALU
    signal do_add, do_sub, do_mult, do_div : std_logic;
    -- The accumulator output should be converted to std_logic_vector
    signal display_value : std_logic_vector( 15 downto 0 );
    -- Signals for input switches
    signal sw_input : std_logic_vector( 15 downto 0 );

    -- Components declaration:
    component Debouncer is
        generic(
            counter_size : integer := 23
        );
        port (
            clk : in std_logic;
            rst : in std_logic;
            bouncy : in std_logic;
            pulse : out std_logic
        );
    end component Debouncer;

    component Accumulator is
        port (
            clk : in std_logic;
            rst : in std_logic;
            ac_init : in std_logic;
```

```
            ac_enable : in std_logic;
            data_in : in signed(15 downto 0);
            result_out : out signed(15 downto 0)
        );
    end component Accumulator;

    component ALU is
        Port (
            a : in signed( 15 downto 0 );
            b : in signed( 15 downto 0 );
            add : in std_logic;
            subtract : in std_logic;
            multiply : in std_logic;
            divide : in std_logic;
            r : out signed( 15 downto 0 )
        );
    end component ALU;

    component Seven_segment_driver is
        generic (
            size : integer := 20
        );
        Port (
            clk : in std_logic;
            rst : in std_logic;
            binary_input : in std_logic_vector( 15 downto 0 );
            CA, CB, CC, CD, CE, CF, CG, DP : out std_logic;
            AN : out std_logic_vector( 3 downto 0 )
        );

    end component Seven_segment_driver;

begin

    -- Buttons Declaration:
    center_detect : Debouncer
    port map (
        clk => clock,
        rst => reset,
        bouncy => BTNC,
        pulse => center_edge
    );

    up_detect : Debouncer
    port map (
        clk => clock,
        rst => reset,
        bouncy => BTNU,
        pulse => up_edge
    );

    down_detect : Debouncer
    port map (
```

```
    clk => clock,
    rst => reset,
    bouncy => BTND,
    pulse => down_edge
);

left_detect : Debouncer
port map (
    clk => clock,
    rst => reset,
    bouncy => BTNL,
    pulse => left_edge
);

right_detect : Debouncer
port map (
    clk => clock,
    rst => reset,
    bouncy => BTNR,
    pulse => right_edge
);

-- Instantiate the seven segment display driver
display_driver_comp : Seven_segment_driver
generic map (
    size => 21
) port map (
    clk => clock,
    rst => reset,
    binary_input => display_value,
    CA => CA,
    CB => CB,
    CC => CC,
    CD => CD,
    CE => CE,
    CF => CF,
    CG => CG,
    DP => DP,
    AN => AN
);

-- Instantiate the accumulator
accum_comp : Accumulator
port map (
    clk => clock,
    rst => reset,
    ac_init => acc_init,
    ac_enable => acc_enable,
    data_in => acc_in,
    result_out => acc_out
);

-- Instantiate the ALU
```

```
alu_comp : ALU
port map (
    a => acc_out,
    b => signed( sw_input ),
    add => do_add,
    subtract => do_sub,
    multiply => do_mult,
    divide => do_div,
    r => acc_in
);

-- Control logic
display_value <= std_logic_vector( acc_out );
LED <= std_logic_vector( acc_out );

-- Operations input
process(clock, reset)
begin
    if reset = '0' then
        acc_init <= '0';
        acc_enable <= '0';
        do_add <= '0';
        do_sub <= '0';
        do_mult <= '0';
        do_div <= '0';
    elsif rising_edge(clock) then
        -- Default values
        acc_init <= '0';
        acc_enable <= '0';
        do_add <= '0';
        do_sub <= '0';
        do_mult <= '0';
        do_div <= '0';

        -- Load switches input
        sw_input <= SW;

        -- Check which button was pressed
        if center_edge = '1' then
            acc_init <= '1';
        elsif up_edge = '1' then
            acc_enable <= '1';
            do_add <= '1';
        elsif down_edge = '1' then
            acc_enable <= '1';
            do_sub <= '1';
        elsif right_edge = '1' then
            acc_enable <= '1';
            do_mult <= '1';
        elsif left_edge = '1' then
            acc_enable <= '1';
            do_div <= '1';
        end if;
```

```
        end if;
    end process;

end Behavioral;
```