Term Project
EEE102, Fall Term

Arda Özkut
22101727
Section 2
Physics Department, Bilkent University
(Dated: December 25, 2022)

Video Link: `https://youtu.be/cGvVuNoRXDE`

The aim of the project is providing ways for hearing impaired individuals to communicate with voice activated technological devices and services. Since most virtual assistants have audial user interfaces inside mobile phones, these services may not available for use by the hearing impaired individuals. This project aims to design a visual user interface to convert certain sign language gestures into commands that can be recognised by computers.

I. METHODOLOGY

The design consists of 2 parts.

- Capturing with BASYS 3 and transmitting this data to the computer
- Training of the Convolutional Neural Network and running real time detection algorithms

For the first part OV7670 CMOS VGA camera module was used as the sensor. The VGA controller and the camera capture unit was built inside the BASYS 3. The initial design included UART to transmit the data acquired by the camera module, however since the master clock frequency of BASYS 3 is 100MHz, UART can transfer data at a rate of 9600 baud. Because this caused problems with the clarity of the video signal so I decided to switch to a data transfer method which is specifically designed for high speed video transfer, which is HDMI. To do this I used an ADC to convert the VGA signal into digital HDMI signal.

For the second part I used computer vision toolboxes and MediaPipe to build and train the perception pipeline. Since images from the OV7670 are noisy, color filters, Gaussian filters and high pass filters were used. Even with the processed images, for the CNN to give accurate results, the training took nearly 4000 images for each class of hand gestures. To build time efficiency when labeling the training data I automated the process of capturing and labeling these gestures. This ensures that the same process can be repeated for desired amount of hand gestures, for example, the entire ASL hand sign language.

FIG. 1: OV7670 CMOS VGA Camera Module

II. VHDL DESIGN SPECIFICATIONS

The inputs and outputs are as follows:

clk100 : in STD_LOGIC
scale1 : in STD_LOGIC
scale2 : in STD_LOGIC
reset : in STD_LOGIC
config_finished : out STD_LOGIC
vga_hsync : out STD_LOGIC
vga_vsync : out STD_LOGIC
vga_r : out STD_LOGIC_vector(3 downto 0)
vga_g : out STD_LOGIC_vector(3 downto 0)
vga_b : out STD_LOGIC_vector(3 downto 0)
ov7670_pclk : in STD_LOGIC
ov7670_xclk : out STD_LOGIC
ov7670_vsync : in STD_LOGIC
ov7670_href : in STD_LOGIC
ov7670_data : in STD_LOGIC_vector(7 downto 0)
ov7670_sioc : out STD_LOGIC
ov7670_siod : inout STD_LOGIC
ov7670_pwdn : out STD_LOGIC
ov7670_reset : out STD_LOGIC

*signals that are shown in bold are signals that are used to communicate with the OV7670 camera module

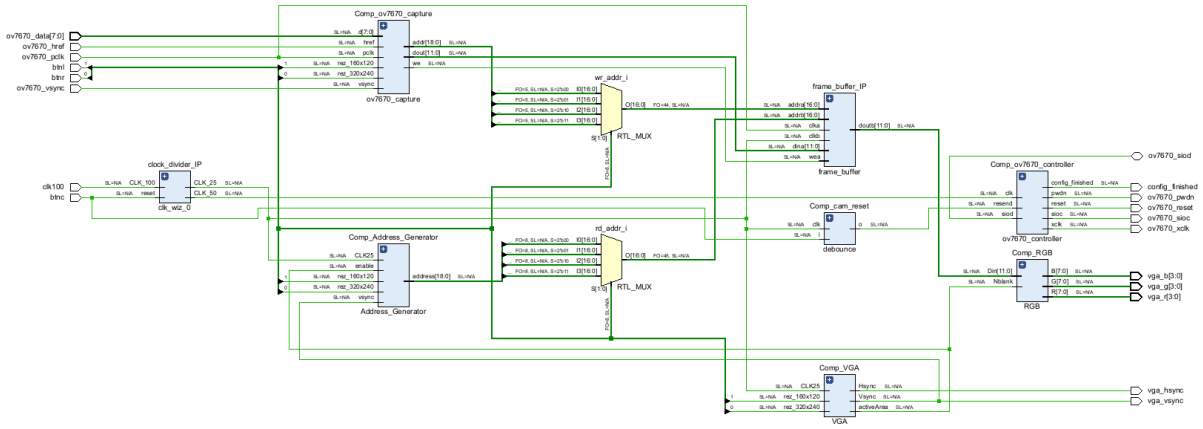

FIG. 2: RTL Schematic

The design includes a clock divider, a camera reset module, a frame buffer, an address generator, an RGB module, a VGA module, a Data Capture Module and a module that controls the OV7670 Camera using I2C communication protocol. The hierarchy can be seen in figure 3.

As can be seen in the figure 3, the clock divider and the frame buffer are constructed as IPs by the Vivado Software because the frame buffer is exceptionally hard to make without any bugs.

The Clock divider module divides the clock into two frequencies 50MHz and 25MHz.

FIG. 3: Design Sources

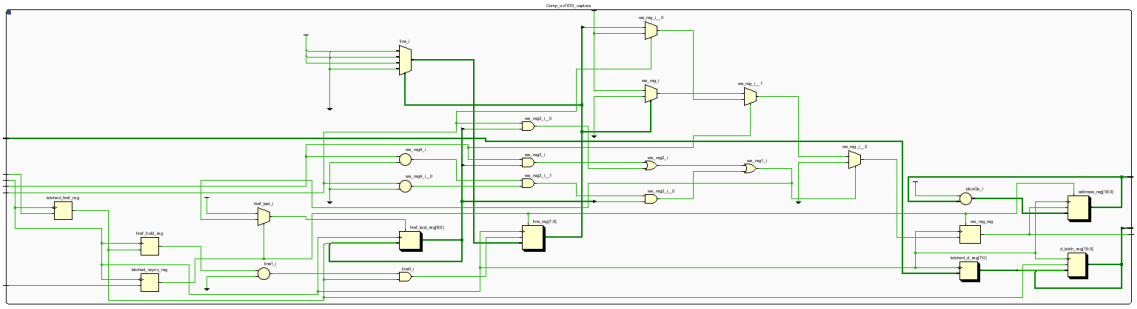

FIG. 4: RTL of the Capture Module

The capture module takes vertical synchronization, horizontal synchronization signals and data inputs, and sequentially stores these data inputs in d flip flops for each pixel of the OV7670 camera module. The module also takes inputs that specify the resolution, and outputs the latch data and the address to these latches where each pixels correspond. The RTL schematic of the capture module can be found in figure 4.

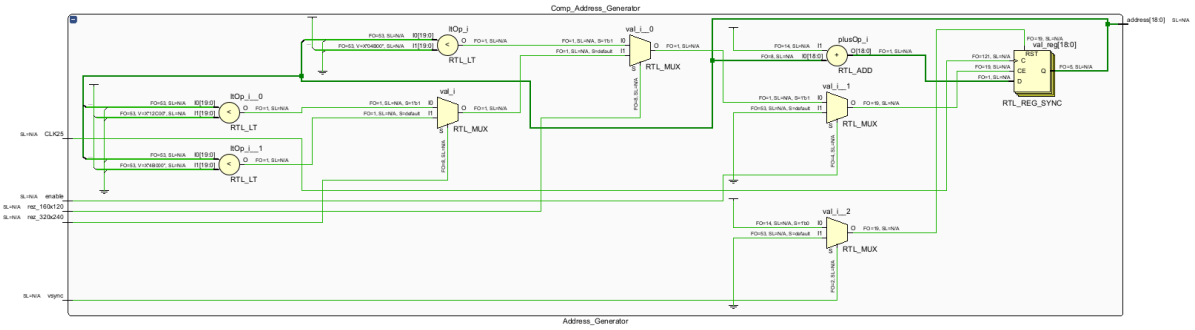

FIG. 5: RTL of the Adress Generator Module

The Adress Generator module takes resolution inputs, the vertical synchronization signal and the 25MHz clock signal to generate the address to the data in the d latches to be stored in the frame buffer line by line. The RTL schematic of the Address Generator Module can be found in figure 5.

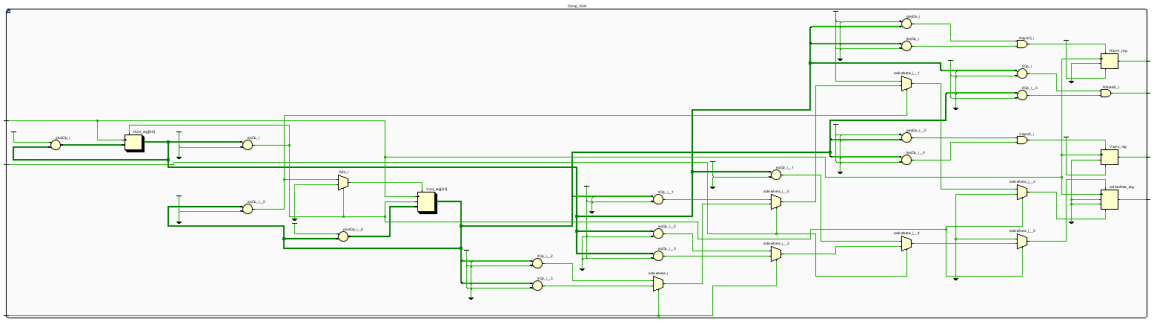

FIG. 6: RTL of the VGA Module

The VGA module takes in information about the horizontal and vertical synchronization signals, the resolution specifiers and the Nblank and activeArea signals that specify whether the vga pins are used or not used in the current moment. These signals are used to specify the black, unused space in the screen that is specified by both the front and back porch of the camera module defined by the horizontal and vertical synchronization signals as well as the resolution specifiers that has inputs via the switches on the BASYS 3. The RTL schematic can be found in figure 6.

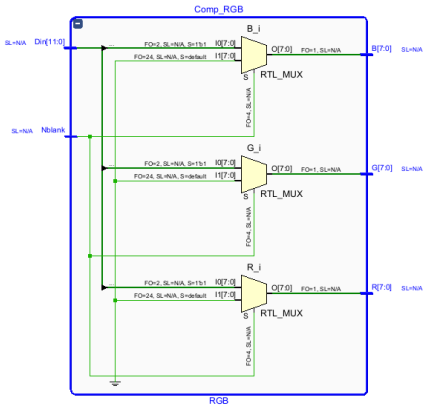

FIG. 7: RTL of the RGB Module

The RGB module takes inputs that are also outputs of the Frame Buffer module and the VGA module and separates the signal received from the frame buffer into red green and blue channels to be sent to the pins of the VGA cable. The Nblank inputs specify that when there is not a pixel at the specified location the output from the VGA pins are "00000000". The RTL schematic can be found in figure 7.

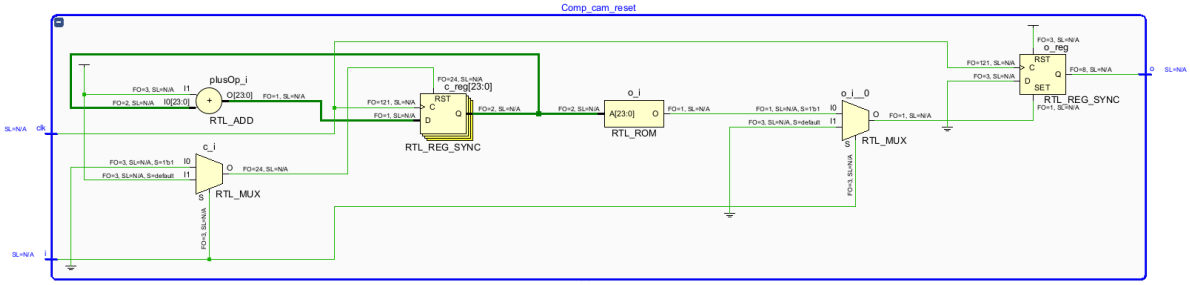

FIG. 8: RTL of the Reset Module

The Reset module takes input from the reset button and the clock divider module. The reset signal is generated in the case the reset button is pressed for a duration of one second. This module helps the camera to be reset in the case of power loss to the system. The RTL schematic can be seen in figure 8.

III. RESULTS

The first part of the project was completed when the video feed was continuous and the scale buttons were working. The video feed with no color conversion can be seen in figures 9, 10 and 11.

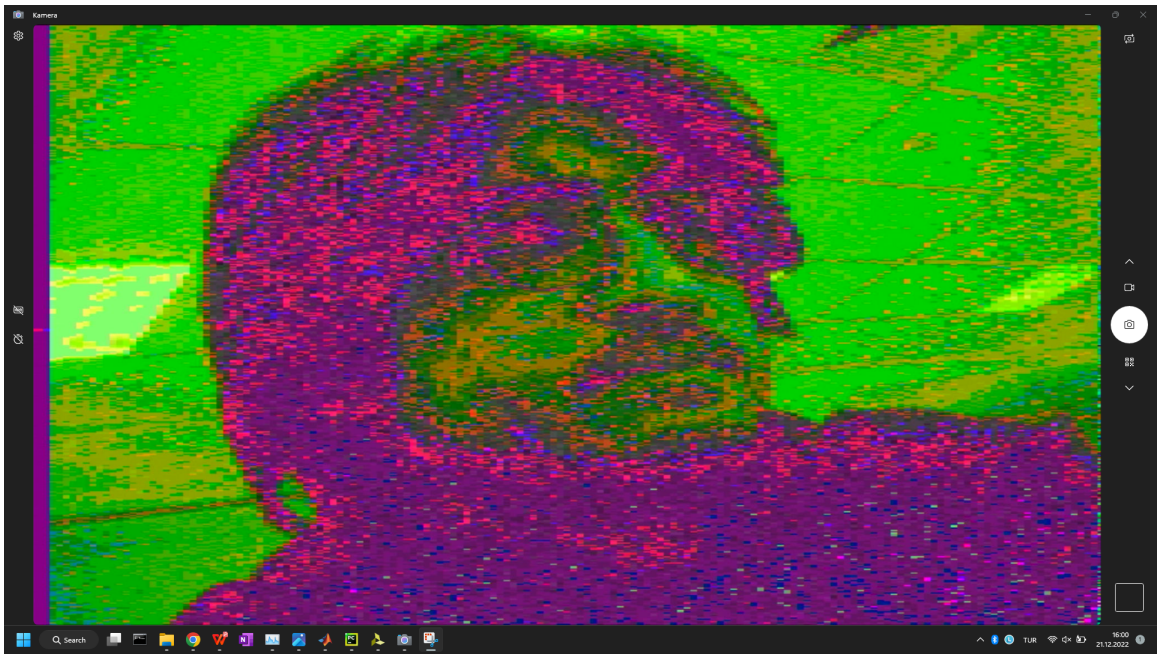

FIG. 9: Scale Down switches are off
No Color Conversion

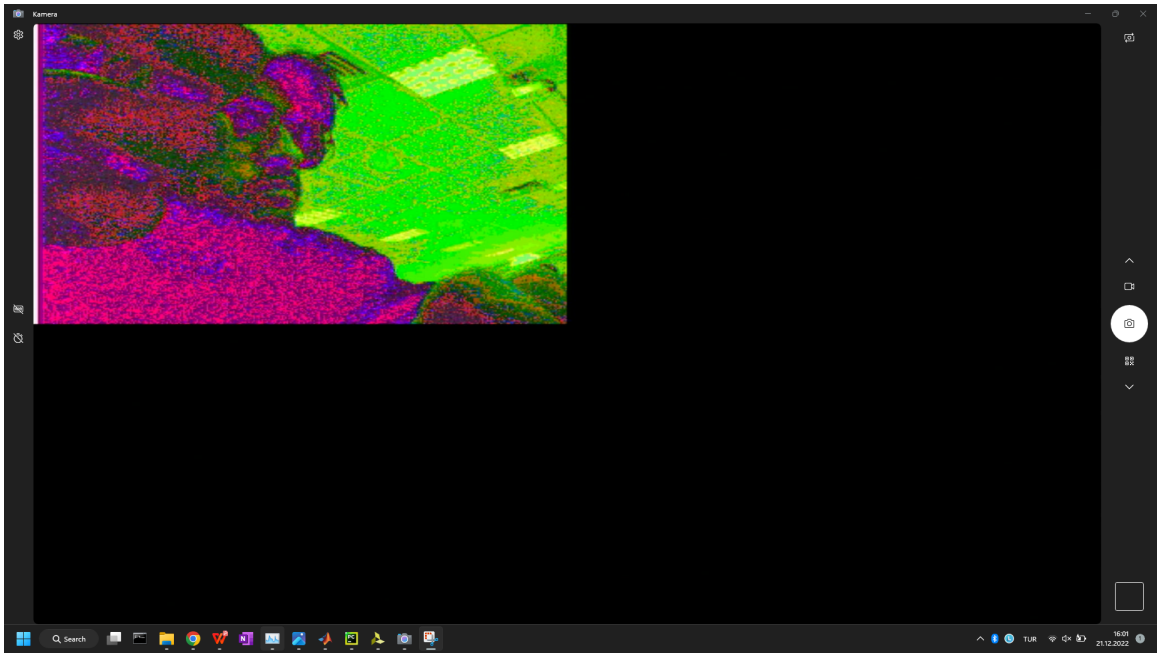

FIG. 10: Scale Down 1 switch is on
No Color Conversion

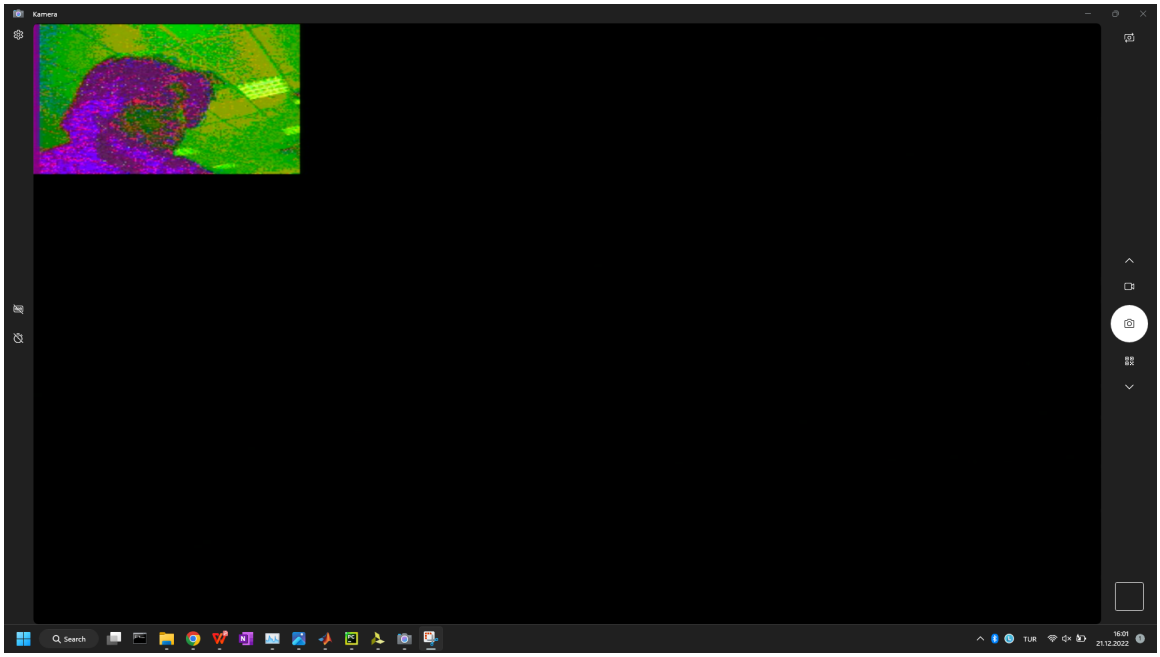

FIG. 11: Scale Down 2 switch is on
No Color Conversion

After the video feed was clear and the first part was completed an algorithm was written that crops the hand and saves the image to be labeled for the training of the network. The figure windows of the training code can be seen in figure 12.

As can be seen from figures 12, 13 and 14 the CNN can accurately detect the 3 dimensional configuration of the hand in space. The label "Left" in these figures is false due to the code running with the mirrored video footage.

The neural network was trained with more than 6000 images to get high confidence levels. The CNN was trained so that open hand is detected as class 0 [FIG. 13.] and closed hand is detected as class 1 [FIG. 14.], the labeling of these classes can be changed easily by the designer to adapt to specific uses and tailored according to the users needs.

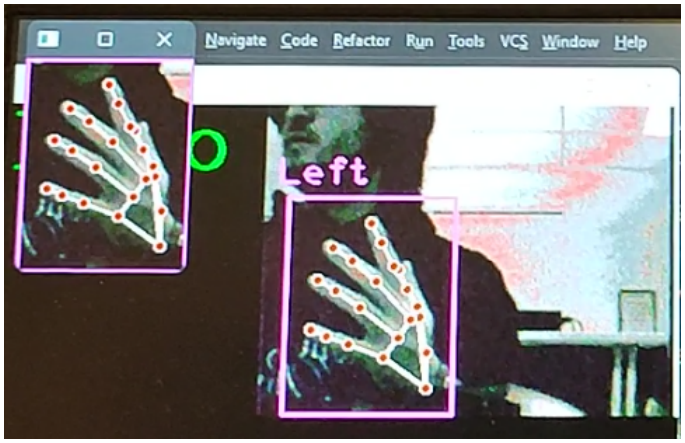

FIG. 12: Hand Capturing and Cropping
With Color Conversion and Gaussian Filtering

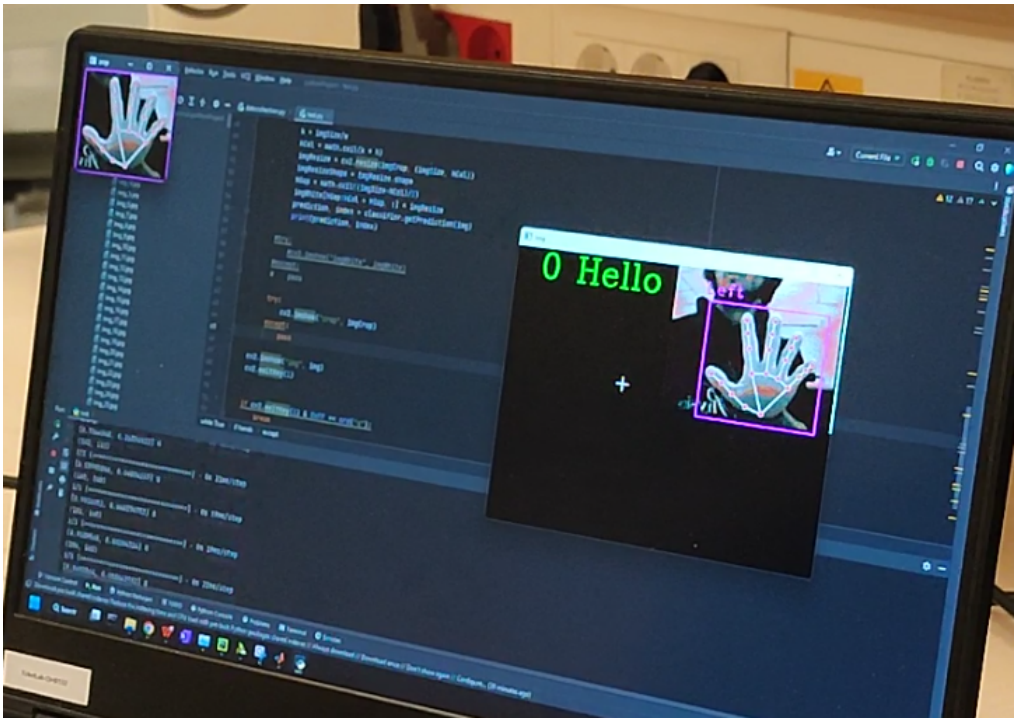

FIG. 13: Class 0 Detection (Hello)

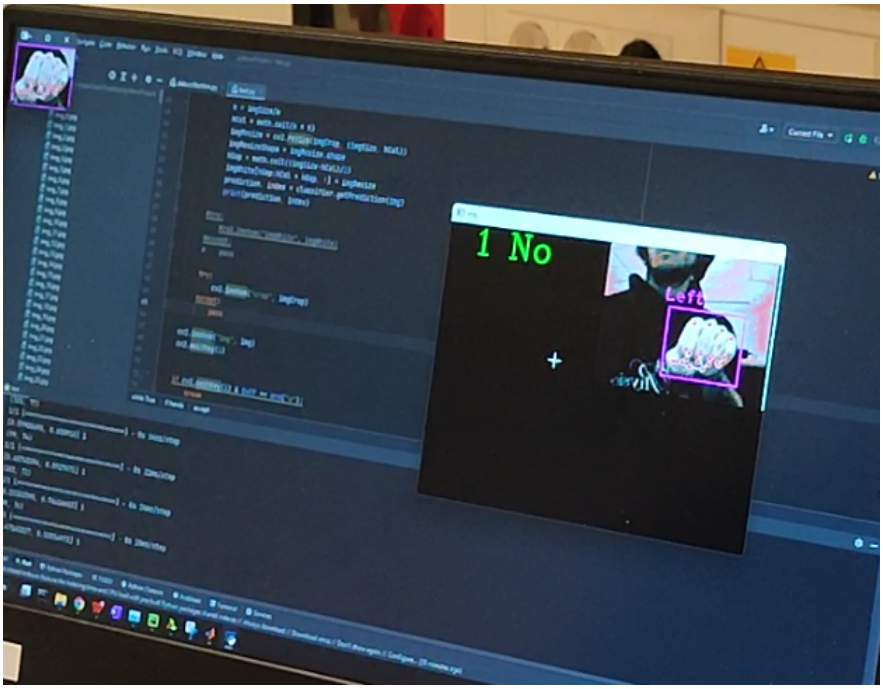

FIG. 14: Class 1 Detection (No)

IV. CONCLUSION

In this computer vision project a hand sign detector using the BASYS 3 FPGA board as a VGA controller card was constructed with the help of computer vision and CNN libraries. The final result is a CNN that can recognize 2 hand gestures with high confidence levels. Training of the neural network required more than 6000 images to gain

these high confidence levels since the noise in the OV-7670 VGA camera was high. To design the VGA controller with the BASYS 3, I first learned all the necessary details on how CRT-TVs, VGA signals and VGA cameras work. Than by looking at online sources and data sheets of the OV-767 camera module, calculating pixel dot rates, vsync-hscync signal rates and address generators, I designed the VGA controller.

Although the I have successfully completed the project, still, there is room for improvements. For example, the Gaussian filtering and the color space conversion can be implemented using VHDL to cut computing time and improve detection accuracy by a great amount.

V. REFERENCES

[1] "How VGA works," YouTube, 30-Oct-2020. [Online]. Available: https://www.youtube.com/watch?v=5exFKr-JJtgamp;ab_channel=TechTangents. [Accessed: 25-Dec-2022].
[2] "VGA video signal format and timing specifications," Javier Valcarce's Homepage, 15-Aug-2007. [Online]. Available: http://javiervalcarce.eu/html/vga-signal-format-timming-specs-en.html. [Accessed: 25-Dec-2022].
[3] Hamsternz, "Hamsternz/fpga_displayport: An implementation of displayport protocol for fpgas," GitHub. [Online]. Available: https://github.com/hamsternz/FPGA_DisplayPort. [Accessed: 25-Dec-2022].
[4] "Home," mediapipe. [Online]. Available: https://google.github.io/mediapipe/. [Accessed: 25-Dec-2022].

VI. APPENDIX

VHDL Code

1. Top Level

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
entity top_level is
    Port ( clk100          : in  STD_LOGIC;
         scale1            : in  STD_LOGIC;
         scale2            : in  STD_LOGIC;
         reset            : in  STD_LOGIC;
         config_finished : out STD_LOGIC;

         vga_hsync : out  STD_LOGIC;
         vga_vsync : out  STD_LOGIC;
         vga_r     : out  STD_LOGIC_vector(3 downto 0);
         vga_g     : out  STD_LOGIC_vector(3 downto 0);
         vga_b     : out  STD_LOGIC_vector(3 downto 0);

         ov7670_pclk  : in  STD_LOGIC;
         ov7670_xclk  : out STD_LOGIC;
         ov7670_vsync : in  STD_LOGIC;
         ov7670_href  : in  STD_LOGIC;
         ov7670_data  : in  STD_LOGIC_vector(7 downto 0);
         ov7670_sioc  : out STD_LOGIC;
         ov7670_siod  : inout STD_LOGIC;
         ov7670_pwdn  : out STD_LOGIC;
         ov7670_reset : out STD_LOGIC
        );
end top_level;
architecture Behavioral of top_level is

    COMPONENT VGA
        PORT(
            CLK25 : IN std_logic;
            rez_160x120 : IN std_logic;
            rez_320x240 : IN std_logic;
            Hsync : OUT std_logic;
            Vsync : OUT std_logic;
            Nblank : OUT std_logic;
            activeArea : OUT std_logic

        );
    END COMPONENT;

    COMPONENT ov7670_controller
        PORT(
            clk : IN std_logic;
            resend : IN std_logic;
            siod : INOUT std_logic;
            config_finished : OUT std_logic;
            sioc : OUT std_logic;
            reset : OUT std_logic;
            pwdn : OUT std_logic;
```

```
        xclk : OUT std_logic
    );
END COMPONENT;

COMPONENT debounce
    PORT(
        clk : IN std_logic;
        i : IN std_logic;
        o : OUT std_logic
    );
END COMPONENT;

COMPONENT frame_buffer
    PORT (
        clka : IN STD_LOGIC;
        wea : IN STD_LOGIC_VECTOR(0 DOWNTO 0);
        addra : IN STD_LOGIC_VECTOR(16 DOWNTO 0);
        dina : IN STD_LOGIC_VECTOR(11 DOWNTO 0);
        clkb : IN STD_LOGIC;
        addrb : IN STD_LOGIC_VECTOR(16 DOWNTO 0);
        doutb : OUT STD_LOGIC_VECTOR(11 DOWNTO 0)
    );
END COMPONENT;
COMPONENT ov7670_capture
    PORT(
        rez_160x120 : IN std_logic;
        rez_320x240 : IN std_logic;
        pclk : IN std_logic;
        vsync : IN std_logic;
        href : IN std_logic;
        d : IN std_logic_vector(7 downto 0);
        addr : OUT std_logic_vector(18 downto 0);
        dout : OUT std_logic_vector(11 downto 0);
        we : OUT std_logic
    );
END COMPONENT;
COMPONENT RGB
    PORT(
        Din : IN std_logic_vector(11 downto 0);
        Nblank : IN std_logic;
        R : OUT std_logic_vector(7 downto 0);
        G : OUT std_logic_vector(7 downto 0);
        B : OUT std_logic_vector(7 downto 0)
    );
END COMPONENT;
component clk_wiz_0
    port (
        CLK_100           : in     std_logic;
        -- Clock out ports
        reset : in std_logic;
        CLK_50          : out    std_logic;
        CLK_25          : out    std_logic);
end component;
COMPONENT Address_Generator
    PORT(
        CLK25       : IN  std_logic;
        rez_160x120 : IN std_logic;
        rez_320x240 : IN std_logic;
```

```
            enable      : IN  std_logic;
            vsync       : in  STD_LOGIC;
            address     : OUT std_logic_vector(18 downto 0)
        );
    END COMPONENT;
    signal clk_camera : std_logic;
    signal clk_vga    : std_logic;
    signal wren       : std_logic_vector(0 downto 0);
    signal resend     : std_logic;
    signal nBlank     : std_logic;
    signal vSync      : std_logic;

    signal wraddress  : std_logic_vector(18 downto 0);
    signal wrdata     : std_logic_vector(11 downto 0);

    signal rdaddress  : std_logic_vector(18 downto 0);
    signal rddata     : std_logic_vector(11 downto 0);
    signal red,green,blue : std_logic_vector(7 downto 0);
    signal activeArea : std_logic;

    signal rez_160x120 : std_logic;
    signal rez_320x240 : std_logic;
    signal size_select: std_logic_vector(1 downto 0);
    signal rd_addr,wr_addr  : std_logic_vector(16 downto 0);
begin
    vga_r <= red(7 downto 4);
    vga_g <= green(7 downto 4);
    vga_b <= blue(7 downto 4);

    rez_160x120 <= scale1;
    rez_320x240 <= scale2;


    clock_divider_IP : clk_wiz_0
        port map(CLK_100 => CLK100,
                 -- Clock out ports
                 reset => reset,
                 CLK_50 => CLK_camera,
                 CLK_25 => CLK_vga);

    vga_vsync <= vsync;

    Comp_VGA: VGA PORT MAP(
            CLK25      => clk_vga,
            rez_160x120 => rez_160x120,
            rez_320x240 => rez_320x240,
            Hsync      => vga_hsync,
            Vsync      => vsync,
            Nblank     => nBlank,
            activeArea => activeArea
        );
    Comp_cam_reset: debounce PORT MAP(
            clk => clk_vga,
            i   => reset,
            o   => resend
        );
    Comp_ov7670_controller: ov7670_controller PORT MAP(
            clk             => clk_camera,
```

```
            resend          => resend,
            config_finished => config_finished,
            sioc            => ov7670_sioc,
            siod            => ov7670_siod,
            reset           => ov7670_reset,
            pwdn            => ov7670_pwdn,
            xclk            => ov7670_xclk
        );
    size_select <= scale1&scale2;

    with size_select select
rd_addr <= rdaddress(18 downto 2) when "00",
        rdaddress(16 downto 0) when "01",
        rdaddress(16 downto 0) when "10",
        rdaddress(16 downto 0) when "11";

    with size_select select
wr_addr <= wraddress(18 downto 2) when "00",
        wraddress(16 downto 0) when "01",
        wraddress(16 downto 0) when "10",
        wraddress(16 downto 0) when "11";

    frame_buffer_IP: frame_buffer PORT MAP(
            addrb => rd_addr,
            clkb   => clk_vga,
            doutb        => rddata,

            clka   => ov7670_pclk,
            addra => wr_addr,
            dina      => wrdata,
            wea      => wren
        );
    Comp_ov7670_capture: ov7670_capture PORT MAP(
            pclk  => ov7670_pclk,
            rez_160x120 => rez_160x120,
            rez_320x240 => rez_320x240,
            vsync => ov7670_vsync,
            href  => ov7670_href,
            d     => ov7670_data,
            addr  => wraddress,
            dout  => wrdata,
            we    => wren(0)
        );
    Comp_RGB: RGB PORT MAP(
            Din => rddata,
            Nblank => activeArea,
            R => red,
            G => green,
            B => blue
        );
    Comp_Address_Generator: Address_Generator PORT MAP(
            CLK25 => clk_vga,
            rez_160x120 => rez_160x120,
            rez_320x240 => rez_320x240,
            enable => activeArea,
            vsync  => vsync,
            address => rdaddress
        );
```

```
end Behavioral;
```

2. VGA

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;



entity VGA is
    Port ( CLK25 : in  STD_LOGIC;
    -- 25 MHz input clock
         rez_160x120 : IN std_logic;
         rez_320x240 : IN std_logic;
         Hsync,Vsync : out  STD_LOGIC;
         --two synchronization signals for the VGA screen
         Nblank : out  STD_LOGIC;
         -- ADV7123 D/A converter control signal
         activeArea : out  STD_LOGIC);
end VGA;

architecture Behavioral of VGA is
    signal Hcnt:STD_LOGIC_VECTOR(9 downto 0):="0000000000";
    -- for column counting (horizontal)
    signal Vcnt:STD_LOGIC_VECTOR(9 downto 0):="1000001000";
    -- for row counting (vertical)

    signal video:STD_LOGIC;
    constant HM: integer :=799; --the maximum size considered 800 (horizontal)
    constant HD: integer :=640; --screen size (horizontal)
    constant HF: integer :=16;          --front porch
    constant HB: integer :=48;          --back porch
    constant HR: integer :=96;          --sync time
    constant VM: integer :=520; --the maximum size considered 525 (vertical)
    constant VD: integer :=480; --screen size (vertical)
    constant VF: integer :=10;          --front porch
    constant VB: integer :=29;          --back porch
    constant VR: integer :=2;           --retrace

begin

    -- initialization of a counter from 0 to 799 (800 pixels per line):
    -- at each clock edge increments the column counter
    process(CLK25)
    begin
        if (CLK25'event and CLK25='1') then
            if (Hcnt = HM) then
                Hcnt <= "0000000000";
                if (Vcnt= VM) then
                    Vcnt <= "0000000000";
                    activeArea <= '1';
                else
                    if rez_160x120 = '1' then
                        if vCnt < 120-1 then
```

```
                        activeArea <= '1';
                    end if;
                elsif rez_320x240 = '1' then
                    if vCnt < 240-1 then
                        activeArea <= '1';
                    end if;
                else
                    if vCnt < 480-1 then
                        activeArea <= '1';
                    end if;
                end if;
                Vcnt <= Vcnt+1;
            end if;
        else
            if rez_160x120 = '1' then
                if hcnt = 160-1 then
                    activeArea <= '0';
                end if;
            elsif rez_320x240 = '1' then
                if hcnt = 320-1 then
                    activeArea <= '0';
                end if;
            else
                if hcnt = 640-1 then
                    activeArea <= '0';
                end if;
            end if;
            Hcnt <= Hcnt + 1;
        end if;
    end if;
end process;


-- generation of the horizontal synchronization signal Hsync:

process(CLK25)
begin
    if (CLK25'event and CLK25='1') then
        if (Hcnt >= (HD+HF) and Hcnt <= (HD+HF+HR-1)) then
        -- Hcnt >= 656 and Hcnt <= 751
            Hsync <= '0';
        else
            Hsync <= '1';
        end if;
    end if;
end process;


--generation of the vertical synchronization signal Vsync:

process(CLK25)
begin
    if (CLK25'event and CLK25='1') then
        if (Vcnt >= (VD+VF) and Vcnt <= (VD+VF+VR-1)) then
        --Vcnt >= 490 and vcnt<= 491
            Vsync <= '0';
        else
            Vsync <= '1';
```

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


    -- Nblank and Nsync to order the ADV7123 builder:

    video <= '1' when (Hcnt < HD) and (Vcnt < VD)
    -- this is to use the full 640 x 480 resolution

             else '0';
    Nblank <= video;
end Behavioral;
```

3. Camera Reset

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

entity seq_res is
    Port ( clk : in  STD_LOGIC;
           in : in  STD_LOGIC;
           reset : out  STD_LOGIC);
end debounce;

architecture Behavioral of seq_res is
        signal count : unsigned(23 downto 0);
begin
        process(clk)
        begin
                if rising_edge(clk) then
                   if in = '1' then
                      if count = x"FFFFFF" then
                                        reset <= '1';
                                else
                                        reset <= '0';
                                end if;
                                count <= count + 1;
                        else
                                count <= (others => '0');
                                reset <= '0';
                        end if;
                end if;
        end process;

end Behavioral;
```

4. Camera Capture

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

```
entity ov7670_capture is
    Port ( pclk  : in   STD_LOGIC;
           rez_160x120 : IN std_logic;
           rez_320x240 : IN std_logic;
           vsync : in   STD_LOGIC;
           href  : in   STD_LOGIC;
           d     : in   STD_LOGIC_VECTOR (7 downto 0);
           addr  : out  STD_LOGIC_VECTOR (18 downto 0);
           dout  : out  STD_LOGIC_VECTOR (11 downto 0);
           we    : out  STD_LOGIC);
end ov7670_capture;

architecture Behavioral of ov7670_capture is
   signal d_latch      : std_logic_vector(15 downto 0) := (others => '0');
   signal address      : STD_LOGIC_VECTOR(18 downto 0) := (others => '0');
   signal line         : std_logic_vector(1 downto 0)  := (others => '0');
   signal href_last    : std_logic_vector(6 downto 0)  := (others => '0');
   signal we_reg       : std_logic := '0';
   signal href_hold    : std_logic := '0';
   signal latched_vsync : STD_LOGIC := '0';
   signal latched_href  : STD_LOGIC := '0';
   signal latched_d     : STD_LOGIC_VECTOR (7 downto 0) := (others => '0');
begin
   addr <= address;
   we <= we_reg;
   dout    <= d_latch(15 downto 12) & d_latch(10 downto 7) & d_latch(4 downto 1);

capture_process: process(pclk)
   begin
      if rising_edge(pclk) then
         if we_reg = '1' then
            address <= std_logic_vector(unsigned(address)+1);
         end if;

         if href_hold = '0' and latched_href = '1' then
            case line is
               when "00"   => line <= "01";
               when "01"   => line <= "10";
               when "10"   => line <= "11";
               when others => line <= "00";
            end case;
         end if;
         href_hold <= latched_href;

         -- capturing the data from the camera, 12-bit RGB
         if latched_href = '1' then
            d_latch <= d_latch( 7 downto 0) & latched_d;
         end if;
         we_reg  <= '0';

         -- Is a new screen about to start (i.e. we have to restart capturing
         if latched_vsync = '1' then
            address      <= (others => '0');
            href_last    <= (others => '0');
            line         <= (others => '0');
         else
```

```
            -- If not, set the write enable whenever we need to capture a pixel
            if (rez_160x120 = '1' and href_last(6) = '1') or
               (rez_320x240 = '1' and href_last(2) = '1') or
               (rez_160x120 = '0' and rez_320x240  = '0' and href_last(0) = '1') then

               if rez_160x120 = '1' then
                  if line = "10" then
                     we_reg <= '1';
                  end if;
               elsif rez_320x240 = '1' then
                  if line(1) = '1' then
                     we_reg <= '1';
                  end if;
               else
                   we_reg <= '1';
               end if;
               href_last <= (others => '0');
            else
               href_last <= href_last(href_last'high-1 downto 0) & latched_href;
            end if;
         end if;
      end if;
      if falling_edge(pclk) then
         latched_d     <= d;
         latched_href  <= href;
         latched_vsync <= vsync;
      end if;
   end process;
end Behavioral;
```

5. RGB

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;


entity RGB is
    Port ( Din  : in    STD_LOGIC_VECTOR (11 downto 0);
    -- 8-bit pixel grayscale
         Nblank : in    STD_LOGIC;
         -- signal indicates the display area, 0 outside the display area

         -- the three colors take 0
         R,G,B  : out   STD_LOGIC_VECTOR (7 downto 0));
         -- the three colors on 10 bits
end RGB;

architecture Behavioral of RGB is

begin
    R <= Din(11 downto 8) & Din(11 downto 8) when Nblank='1' else "00000000";
    G <= Din(7 downto 4)  & Din(7 downto 4)  when Nblank='1' else "00000000";
    B <= Din(3 downto 0)  & Din(3 downto 0)  when Nblank='1' else "00000000";
```

```
end Behavioral;
```

6. Address Generator

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;
use IEEE.STD_LOGIC_ARITH.ALL;
use IEEE.STD_LOGIC_UNSIGNED.ALL;

entity Address_Generator is
    Port (      CLK25,enable : in  STD_LOGIC;
            rez_160x120  : IN std_logic;
            rez_320x240  : IN std_logic;
            vsync        : in  STD_LOGIC;
                                 address : out STD_LOGIC_VECTOR (18 downto 0));
                                 -- generated address

end Address_Generator;

architecture Behavioral of Address_Generator is
   signal val: STD_LOGIC_VECTOR(address'range):= (others => '0');
   -- intermediate signal

begin
        address <= val;
        -- generated address


        process(CLK25)
                begin
         if rising_edge(CLK25) then
            if (enable='1') then

            -- if enable = 0, address generation is stopped
               if rez_160x120 = '1' then
                  if (val < 160*120) then
                  -- if the memory space is completely scanned

                     val <= val + 1 ;
                  end if;
               elsif rez_320x240 = '1' then
                  if (val < 320*240) then
                  --  if the memory space is completely scanned
                     val <= val + 1 ;
                  end if;
               else
                  if (val < 640*480) then
                  --  if the memory space is completely scanned
                     val <= val + 1 ;
                  end if;
               end if;
                                 end if;
            if vsync = '0' then
               val <= (others => '0');
            end if;
```

```
                    end if;
                end process;
end Behavioral;
```