ECE 441/543 Lab 3 Report

Digital Image Acquisition , Display and Processing System

1st Khephren Gould
V01012827
Dept. of Electrical and Computer Engineering
University of Victoria
khephrengould@uvic.ca

2nd Zarina Guzman
V01075558
Dept. of Electrical and Computer Engineering
University of Victoria
zguzman@uvic.ca

***Abstract*—This report describes the design, implementation, and evaluation of a real-time digital image acquisition, display, and processing system implemented on a Xilinx Artix-7 NEXYS A7 FPGA development board. The system connects an OV7670 digital camera, operating at up to 30 frames per second, to a 640×480 VGA display using on-chip block RAM (BRAM) as a frame buffer. All modules are written in VHDL and synthesized using the Xilinx Vivado® design environment.**

A series of experiments demonstrate key components of the system, including VGA signal generation and timing, push-button-controlled movement of graphical objects, live camera capture and display, RGB-to-greyscale conversion using an NTSC-style luminance approximation, and real-time Sobel edge detection. The programming assignment extends the system by implementing a negative-image filter for both colour and greyscale modes, selectable using slide switches.

The results show correct operation of each subsystem and demonstrate that FPGA-based designs are well suited for high-throughput, low-latency image processing pipelines.

***Index Terms*—FPGA, VHDL, VGA, digital image processing, OV7670, block RAM, edge detection, Sobel operator, RGB-to-greyscale, real-time video, Artix-7, Vivado, image acquisition, negative image, synchronisation.**

I. Introduction

This laboratory explores the design and evaluation of a complete image acquisition and processing pipeline implemented in VHDL on the Xilinx Artix-7 NEXYS A7 FPGA board. The system is developed incrementally over five steps.

Steps 1 and 2 focus on the fundamentals of VGA signal generation. Horizontal and vertical synchronisation pulse timing is measured with an oscilloscope, and a static coloured square is displayed on a 640×480 VGA monitor. Push-button inputs are then used to move and resize the square during runtime.

In Step 3, an OV7670 CMOS camera module is integrated to enable live video capture. Frames from the camera are stored in an on-chip BRAM frame buffer, and multiple image sources and processing modes can be selected using the board's slide switches.

Step 4 implements an RGB-to-greyscale conversion using the NTSC-style luminance approximation

$$Y \approx 0.25R + 0.50G + 0.125B, \tag{1}$$

where the coefficients are chosen as powers of two so that the computation can be implemented efficiently using shift operations rather than floating-point arithmetic.

Step 5 investigates the Sobel edge detector, a common spatial filtering technique used to estimate image gradients. Two 3×3 convolution kernels compute horizontal and vertical gradients, and the combined magnitude $|x| + |y|$ is displayed as a greyscale edge-strength image in real time.

In addition to the guided steps, the programming assignment requires the implementation of a negative-image filter by replacing the provided stub module. The filter performs a bitwise complement on each colour channel to invert the image.

Overall, the experiment provides practical experience with FPGA-based video systems, including memory interfacing, synchronous video signal generation, and the design of streaming image-processing datapaths in VHDL.

The remainder of this report is organised as follows. Section 2 describes the experimental method and VHDL implementation for each step. Section 3 presents the results, including oscilloscope measurements, resource utilisation, and example image outputs. Section 4 discusses the results and answers the questions from the lab specification. Section 5 concludes the report.

II. In Lab Steps

A. Step 1 VGA Display Example 1

As shown in Figure 1, the clock signal required for the VGA interface is a 25 MHz square wave. This is generated in the vga_display_P615_p256.vhd source file. This file contains an entity (clock_generator.vhd) that implements a 4:1 clock division scheme (4 100MHz clock pulses produce 1 25MHz Clock Pulse). This is accomplished by using nested conditional

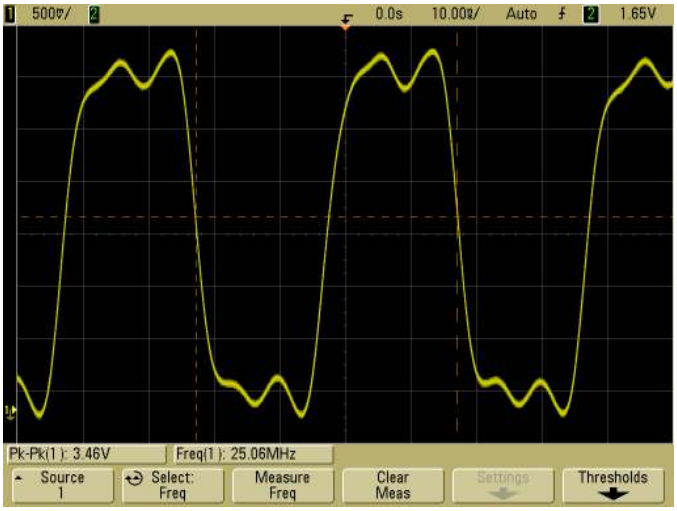

Fig. 1: VGA Clock Signal

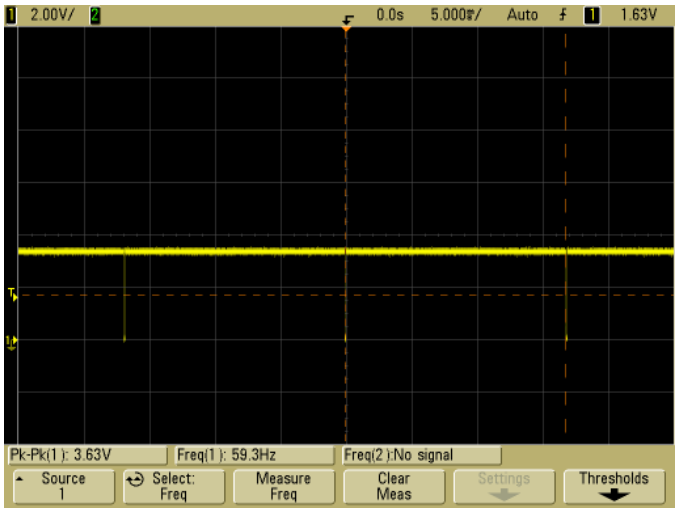

Fig. 2: VGA V-Sync Signal

(if) statements. The outer condition checks for the rising edge of the clock. Inverting the output on every rising edge of the input divides the input by two since the output is being inverted half as frequently as the input. The inner condition checks a counter. The counter alternates between 0 and 1 which, when nested within the rising edge condition implements an additional division by 2. Thus, the resulting output is 1/4 of the input frequency.

Figures 4, and 2 show the H-sync and V-sync signals generated by the FPGA for VGA communication with the monitor. These signals are generated in vga_display_P614_p260_v2.vhd. This file contains an entity (vga_signal_gen) that generates the signal timings. The h-sync signal is driven based on the value of a 0-800 up counter. As the counter is incremented at each clock pulse it effectively divides the clock by 800 resulting in the 31.1KHz frequency shown in Figure 4. Hsync signal is divided into 4 sections: front porch, back porch, signal pulse, and active video. The entity uses conditional statements within a process block to set the horizontal signals according to each region.

The v-sync signal is driven based on another up counter (0-525). Rather than being sensitive to the 25MHz clock (as with h-sync) the v-sync signal is sensitive to the h-sync signal itself. This results in a further division of the 31.1KHz signal by a factor of 525. The resulting signal has a frequency of 60Hz which reflects the frequency shown in Figure 2. As with the h-sync signal, the v-sync signal is divided into 4 sections. The entity generates the v-sync signal in a similar way to the h-sync signal.

The waveforms show the durations of the sync pulses of each signal. Examining the horizontal grid divisions we can estimate a sync pulse duration of 3.81330 us for h-sync and 0.0636 us for v-sync. This is reflected in the hdl source. V-sync is set to low for only 2 out of 525 count values whereas H-sync is set to low for 96 of 800 count values.

When Ri = Gi = Bi, the colour specified for each pixel has equal red, green, and blue components. When i = 0000,

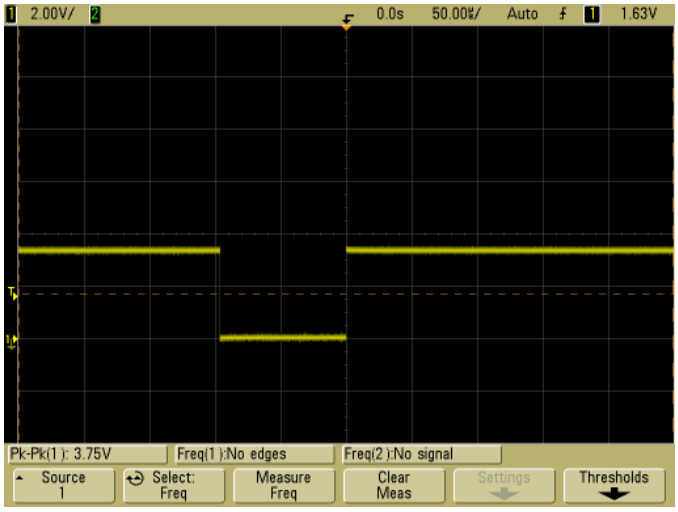

Fig. 3: VGA V-Sync Signal Zoomed

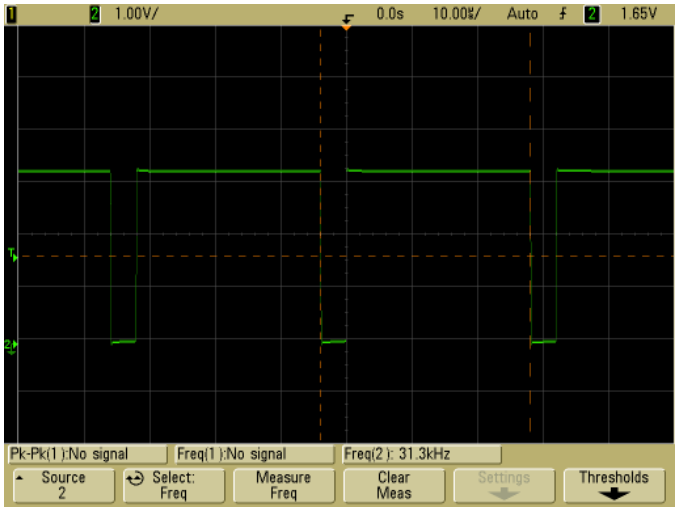

Fig. 4: VGA H-Sync Signal

TABLE I: Colour Observed For Varying RGB Signals

R	G	B	Colour Observed
0000	0000	0000	Black
1111	1111	1111	White
1111	1111	0000	Dirty Yellow
1111	0000	1111	Light Purple
0000	1111	0000	Teal

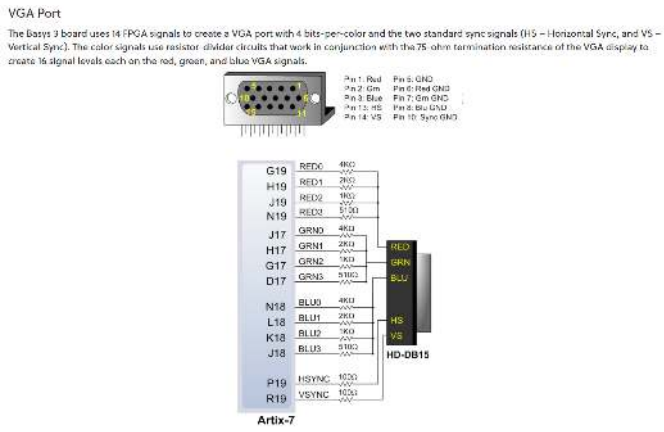

Fig. 5: Basys3 Resistor Network

it results in the colour black being displayed on the screen. As i is increased towards 1111, the colour on the screen shifts towards the colour white. At 1111, the screen is completely white.

For generating the Analog VGA signals, The Nexys A7 has a resistor network that works to create voltage levels needed for the VGA connector. Figure 5 shows an excerpt from the BASYS 3 Development Board documentation which shows the network (the circuit is similar for the Nexys). By varying the resistance between each pin of the VGA connector, the bits of each R,G, and B output from the FPGA can be assigned weights. This results in an analog signal that typically ranges from 0V to 0.7V for each colour.

Each colour component (R,G,B) can be assigned 16 distinct values. As the final colour is a combination of the three colour components, we can multiply the number of each individual component values together to obtain the overall number of combinations. This results in:

$$Y = (16)(16)(16) = 4,096 \tag{2}$$

B. Step 2 - VGA Display Example 2

In step 2, the user is given the option to move the red square displayed on the screen in step 1. This is implemented by the vga_display_P640_p269.vhd source file. This source contains an entity (vga_obj_motion) that exposes input ports (btn_left,btn_right ....) to be connected to the physical buttons on the Nexys 3 Development board. Within the architecture of this entity, an "object_move" process checks to see which of the 4 inputs is high (which indicates the button has been pressed). This is synchronized to the display refresh cycle by making the process sensitive to the rising edge of v-sync. This ensures phenomena such as tearing do not occur. Once the active button has been determined, the value of an internal register storing the current position of the square is updated (a constant is either added or subtracted to the x / y position) based on the button that has been pressed.

C. Step 3 - Image Capture and Display With Digital Camera

Switch SW[0] controls whether writing to the BRAM frame buffer is enabled or disabled. When SW[0] is low, pixel data from the camera is continuously written into BRAM, updating the frame buffer in real time. When SW[0] is set high, writes to BRAM are halted, freezing the most recently captured frame. The VGA display continues to read from BRAM regardless of SW[0], so the frozen frame remains visible on the monitor. This is useful in combination with SW[7] to capture and hold a single still frame for examination.

D. Step 4 - Conversion of an RGB Colour Image to Greyscale

describe/explain the VHDL implemented in Steps 4

The RGB-to-greyscale conversion is implemented in rgb_to_gray.vhd using a purely combinational design, meaning no clock is required. The 12-bit input RGBin is first separated into its three 4-bit colour components. Each component is then right-shifted to approximate the NTSC weighting factors from Equation 3: the red channel is shifted by 2 bits (approximately ×0.25), green by 1 bit (×0.5), and blue by 3 bits (×0.125).

The resulting values are added together to form a single 4-bit luminance value. This value is then copied into the red, green, and blue output fields to generate the final 12-bit greyscale pixel. Using bit shifts instead of multipliers simplifies the hardware and keeps the design efficient for real-time pixel streaming.

E. Step 5 - Demonstration of a video rate sobel edge detector

describe/explain the VHDL implemented in Steps 5

In step 5, the Sobel edge detection algorithm was demonstrated. This algorithm applies a two dimensional convolution operation to the video signal received from the camera. The video signal maps a position (x,y) to an intensity. The Sobel filter convolves this signal with two filter kernels (each of which are also two dimensional). The horizontal filter kernel is shown in Figure . The vertical filter kernel is shown in Figure . The output of each operation is combined using the equation to produce

III. Questions

A. Draw a top-level block diagram of the project in Step 3 and briefly describe the operation of each module

A block diagram of the project is shown in Figure 6,

As shown in the figure the project integrates an OV7670 image sensor, DDR2 RAM storing still images, user input from slide switches and buttons as well as two output displays (VGA and SPI).

- **camera (top-level):** Integrates all sub-modules and routes signals, including a slide-switch multiplexer that selects the active video stream for VGA output.

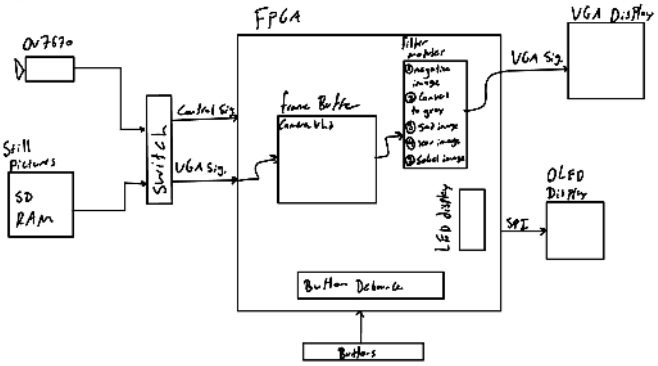

Fig. 6: Block Diagram of Video Rate Filtering Project

- **camera_video (video.vhd):** Interfaces with the OV7670 camera, reconstructs 12-bit RGB pixels, writes frames to BRAM, and supplies pixel data to the display pipeline.
- **histogram_red / histogram_green / histogram_blue / histogram_gray (histogram.vhd):** Generate per-channel intensity histograms from the pixel stream for OLED display.
- **camera_status (oled_display.vhd):** Drives the onboard OLED to display system status or histogram data (selected by SW[12]).
- **convert_to_gray (rgb_to_gray.vhd):** Converts 12-bit RGB pixels to 8-bit greyscale using an NTSC-based approximation.
- **video_negative_image (negative.vhd):** Produces a negative image via bitwise colour inversion.
- **video_sad_image (sad.vhd):** Implements a Sum of Absolute Differences edge detector ("Dinosaur vision" effect).
- **video_xor_image (xor.vhd):** Applies a frame-difference XOR filter.
- **video_sobel_image (sobel.vhd):** Performs Sobel edge detection with selectable direction (SW[5:6]).

B. How much FPGA on-chip memory is required to store one frame of a VGA image from the OV7670 camera?

BRAM is being used to buffer the OV7670 VGA Camera Data. A frame output from the OV7670 Camera is composed of 640 x 480 pixels. Each pixel's colour is represented by a 12 bit RGB value. The total number of bits occupied by a Camera frame is 460kB. With 36kb BRAM cells (as found in the XC7A100T Artix 7 FPGA) we would expect 58 BRAM cells or 42% memory usage.

$$640 \times 480 \times 12 = 3,686,400 bits = 460 kB \tag{3}$$

C. What is meant by an edge detector for images

An edge detector is an image processing operation that identifies locations within an image where the pixel intensity changes abruptly. These changes in intensity typically correspond to physical boundaries between objects, transitions in surface material or orientation, or variations in lighting. Mathematically, edges correspond to regions of high spatial gradient in the image intensity function.

The output of an edge detector is typically a greyscale image in which bright pixels indicate strong edges and dark pixels indicate smooth, uniform regions.

D. Briefly describe an example of another edge detector that operates in a 3x3 window similar to the Sobel as seen in this lab

The Prewitt operator is a 3×3 edge detector that operates similarly to the Sobel edge detector. It uses two convolution kernels to estimate the horizontal and vertical image gradients:

$$P_x = \begin{bmatrix} -1 & 0 & 1 \\ -1 & 0 & 1 \\ -1 & 0 & 1 \end{bmatrix}, \quad P_y = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ -1 & -1 & -1 \end{bmatrix} \tag{4}$$

The combined edge magnitude is then computed as $|P_x| + |P_y|$. The difference from the Sobel edge detector is that the Prewitt kernel weights all rows equally, whereas the Sobel applies a weight of 2 to the centre row or column. This means the Sobel operator incorporates a degree of Gaussian smoothing along the perpendicular direction, making it slightly less sensitive to noise than the Prewitt operator. In an FPGA implementation, both operators have similar complexity since neither requires a true multiplier, all weights are ± 1.

E. Explain the operation and implementation of the functions that are controlled by the slide switches in this project

SW[0] – Frame buffer write enable: When low, pixel data from the OV7670 camera is continuously written into the BRAM frame buffer, enabling live video. When high, writes are stopped and the most recently captured frame is frozen in memory. The VGA read path continues unaffected, so the frozen frame remains displayed.

SW[1] – Live camera / still image select: When low, the live camera pixel stream is used as the input to all processing modules. When high, one of five pre-stored still images held in SDRAM is selected instead (SMPTE colour bars, checkerboard, mountain scene, river scene, or geometric shapes). The active still image is cycled using BTNL and BTNR.

SW[2:4] – Processing pipeline select: These three bits form a 3-bit select signal for the output multiplexer:

- `000` – Normal colour camera image
- `001` – Negative image (bitwise complement of each channel)
- `010` – SAD moving edge detector ("Dinosaur vision")
- `011` – Sobel edge detection (direction configured by SW[5:6])
- `100` – Greyscale conversion
- `101` – XOR filter
- `110`, `111` – Normal colour image (reserved for future use)

Resource	Utilization	Available	Utilization %
LUT	10788	63400	17.02
LUTRAM	2578	19000	13.57
FF	7859	126800	6.20
BRAM	9	135	6.67
IO	116	210	55.24
BUFG	12	32	37.50
MMCM	2	6	33.33
PLL	1	6	16.67

Fig. 7: Video Rate Edge Detection FPGA Utilization

SW[5:6] – Sobel filter direction: Configures which gradient output of the Sobel module is displayed: `01` selects the vertical gradient only, `10` selects the horizontal gradient only, and `11` selects the combined magnitude $|G_x| + |G_y|$.

SW[7] – Video capture enable: Enables storage of multiple consecutive frames into memory. At 30 Hz this captures approximately 4 seconds of video; at 15 Hz approximately 8 seconds.

SW[12] – OLED display mode: Selects between a text-based status display and a graphical colour histogram on the onboard OLED screen.

SW[13] – Camera frame rate: Selects between 15 Hz and 30 Hz operation of the OV7670 camera.

SW[14:15] – OV7670 drive current: Sets the output drive current for the OV7670 camera interface lines. This setting is latched only at power-up and has no effect once the system is running.

F. Capture a project a summary and show the resources used in the Artix-7 FPGA

As shown in Figure 7, significant FPGA resources were utilized in implementing the video rate filtering algorithms in steps 3 through 5. In addition to the usage of lookup tables, block RAM (BRAM) as well as clock source resources (PLL and MMCM) were used. The BRAM was required to act both as a temporary buffer for the VGA signal data from the image sensor as well as permanent (but still volatile) storage for the static images.

IV. Conclusion

This laboratory successfully demonstrated the design and evaluation of a complete real-time image acquisition, display, and processing system on the Xilinx Artix-7 NEXYS A7 FPGA. Each subsystem was verified. VGA sync signal generation was confirmed against oscilloscope measurements, live camera capture and display was demonstrated using the OV7670 module, and image processing operations including greyscale conversion, Sobel edge detection, and negative image generation were all implemented and observed in real time.

The use of BRAM as a frame buffer proved effective, consuming approximately 6.67% of the available on-chip memory for a single $640 \times 480 \times 12$-bit frame. The slide-switch multiplexer architecture allowed flexible selection of processing pipelines without requiring re-synthesis.

Overall, the experiment demonstrated the strengths of FPGAs for real-time video processing applications. Their ability to perform many operations in parallel and provide predictable timing makes them well suited for high-throughput, low-latency image processing tasks compared to traditional software-based approaches.

V. Appendix A: Code Listings - Negative Image Assignment

Extend the image processing system to include the display of a negative image (for both color and grey-scale) when selected by slide switches. The current ”negative.vhd” component in the top level module of the project is coded as a ’stub’ which currently displays the normal color image. You will replace this with your own code that generates the negative. Test your code on the NEXYS A7 FPGA board using the OV7670 digital camera as input and the VGA monitor for output. Include your code listing in your Lab Report.

Listing 1: Negative Image Design

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

entity negative_image is
   Port (
      pixel_in : in STD_LOGIC_VECTOR (11 downto 0);
      pixel_out : out STD_LOGIC_VECTOR (11 downto 0)
   );
end negative_image;

architecture Behavioral of negative_image is

signal red, green, blue : unsigned( 3 downto 0 );

begin
--image negative: subtract the current value from 255
   red <= NOT(unsigned(pixel_in( 11 downto 8 )));
   green <= NOT(unsigned( pixel_in( 7 downto 4 )));
   blue <= NOT(unsigned( pixel_in( 3 downto 0 )));


   pixel_out <= std_logic_vector( red & green & blue );

end Behavioral;
```

Fig. 8: Negative Image Results 1

References

[1] Doulos Ltd., ”EDA Playground,” EDA Playground. [Online]. Available: https://www.edaplayground.com. [Accessed: Feb. 19, 2026].