

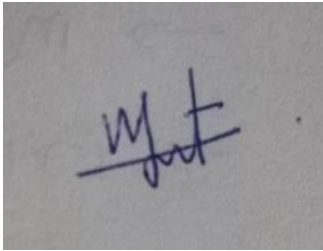
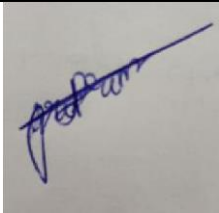


# CASE STUDY EMBEDDED CONTROL SOLUTIONS

## FPGA, GROUP A

I hereby declare that I have written this paper independently. I have not submitted it for any other examination purposes. I have not use other references or material than mentioned in the bibliography and I have marked all literal and analogous citations.

| <b>Student:</b>   | <b>Signature:</b>   | <b>Contribution:</b>  |
|---|---|---|
| <p>Hosamth Abhishek</p> <p>Matriculation Number: - 2213179</p>            |    | <p>Abstract, Introduction, Objectives, Implementation-</p> <ul style="list-style-type: none"> <li>• Board Setup</li> <li>• Graphical User Interface</li> </ul>  |
| <p>Medina Jorge</p> <p>Matriculation Number: - 22301049</p>               |   | <p>Design-</p> <ul style="list-style-type: none"> <li>• Software Design</li> </ul> <p>Implementation- •</p> <ul style="list-style-type: none"> <li>• Camera</li> <li>• Hardware Accelerated Algorithm</li> </ul>  |
| <p>Samant Vinayak</p> <p>Matriculation Number: - 22303846</p>             |  | <p>Design-</p> <ul style="list-style-type: none"> <li>• Process Flowchart</li> <li>• Hardware Design</li> </ul> <p>Implementation-</p> <ul style="list-style-type: none"> <li>• Background Removal</li> <li>• Non-Accelerated Algorithm</li> <li>• SVG Generation</li> <li>• PNG Generation</li> <li>• Pen Plotter</li> </ul> <p>Summary-</p> <ul style="list-style-type: none"> <li>• Limitations</li> </ul> |
| <p>Prakash Ram Gopalakrishnan</p> <p>Matriculation Number: - 12304894</p> |  | <p>Summary- •</p> <p>Results</p> <ul style="list-style-type: none"> <li>• Conclusion</li> </ul>   |

Hosamth Abhishek

Matriculation Number:- 2213179  
Technische Hochschule Deggendorf  
Cham, Germany  
[abhishek.hosamath@stud.th-deg.de](mailto:abhishek.hosamath@stud.th-deg.de)

Medina Jorge

Matriculation Number:- 22301049  
Technische Hochschule Deggendorf  
Cham, Germany  
[jorge.medina@stud.th-deg.de](mailto:jorge.medina@stud.th-deg.de)

Samant Vinayak

Matriculation Number:- 22303846  
Technische Hochschule Deggendorf  
Cham, Germany  
[vinayak.samant@stud.th-deg.de](mailto:vinayak.samant@stud.th-deg.de)

Prakash Ram Gopalakrishnan

Matriculation Number:- 12304894  
Technische Hochschule Deggendorf  
Cham, Germany  
[prakashram.gopalkrishnan@stud.th-deg.de](mailto:prakashram.gopalkrishnan@stud.th-deg.de)

**Abstract**—This paper focuses on the design, development, testing and implementation of an embedded system whose primary objective is to convert captured images by a camera into line drawings and later plot onto a paper using pen plotter, where all these actions can be controlled using a touchscreen display with a simple and friendly graphical user interface. The system features a camera module for clicking pictures, processing the images into line drawings, and producing Scalable Vector Graphics (SVG) format making it suitable for the pen plotter machine. The whole system is coordinated using the Field-Programmable Gate Array (FPGA) module available on the PYNQ-Z1 board to process the images, feeding inputs to the pen plotter and receiving the commands from user interface touch/button inputs.

**Keywords**—Dots Algorithm, FPGA, PYNQ Z1, Hardware Acceleration, AxiDraw Penplotter

## I. INTRODUCTION

The convergence of art and technology has always given rise to an abundance of creative applications, redefining the possibilities of creative expressions like Augmented reality (AR), Virtual Reality (VR), 3D printing in sculptures, Digital Animation etc. In this case study an attempt was made to create an embedded platform that showcases such a creative application, especially focusing on converting camera images like .jpg or .png format files into dot drawings and then producing them with a pen plotter which emphasizes real-time image processing and pen plotter rendering.

Reconfigurable circuits, such as Field Programmable Gate Arrays (FPGAs), are appealing from a technological standpoint. In our study, FPGA plays a crucial role in accelerating the image processing speed. In general, FPGA is known for its reconfigurable integrated circuits, offering flexibility and computational power ensuring a responsive and efficient system when integrated with FPGA giving rise to its use in various range of applications.

FPGAs are widely used in the communication domain because of their capability to handle baseband processing in wireless communication and protocol implementation for networking [1] like in the field of avionics, radar processing, and electronic warfare

applications by aerospace and defense sectors. Not only in the field of networking but FPGAs are also used wisely in the field of medical industries for medical imaging activities like image processing [2] in diagnostic devices like CT scanners and MRI machines. Some other applications of FPGAs are seen in the field of automotive systems for engine management, infotainment, and Advanced Driver Assistance Systems (ADAS) [3]. Additionally, FPGAs are used in consumer electronics, embedded systems, security, cryptography [4], and other fields that highlight their versatility and effectiveness in solving a variety of computing problems.

The motivation for this is driven by the goal of understanding the workings of the system that combines components both from software and hardware integration with the PYNQ-Z1 board. This embedded platform is designed to accommodate a camera module for image capture, employ image processing through a dot algorithm to produce a .svg file format for pen plotting, and using Field Programmable Gate Array (FPGA) for acceleration, all this with a user interface. Thereby clarifying our goals for this study.

In this paper, the sole purpose of plotting the captured images into paper as dot drawings is to work with FPGA on a PYNQ board. While various other microcontrollers or integrated circuits exist for the purpose of achieving this objective, we have made an intentional selection of FPGA while also understanding its limitations and probable challenges of using FPGAs. The use of this FPGA for pen plotting is for research and education purposes only. Moreover, this study also aims to show and explore the hardware acceleration process carried out for this task.

## II. REQUIREMENTS

The primary goal of this project is to create an algorithm which can take inputs from the users to perform actions like capturing images, converting images to dot drawings, and plotting the drawing on paper using a pen plotter machine while meeting our defined goals which are classified as functional and non-functional requirements.

#### Functional Requirements:

- The user should be able to capture pictures with the camera module.
- The user should be able to preview the captured picture on the Graphical User Interface (GUI)
- The user should be able to see the preview of the dotted vector image.
- The user should be able to print the dotted image on the pen plotter.
- The user should be able to enable a hardware acceleration feature, that will process the image faster.

#### Non-Functional Requirements:

- The system should be developed in the PYNQ-Z1 board.
- All modules should be running on the CPU of the board, except for the hardware acceleration that needs to run on the FPGA.
- There must be some contrast between the background and the object into picture.

#### Acceptance Criteria:

The outcome of the project is to print the dotted drawing of the captured image as shown in the below example.



Fig. 1. Captured image to dotted image.

#### Hardware Requirements:

**PYNQ-Z1 Board:** This board is based on Xilinx Zynq-7000 All Programmable Soc, this board is a stable platform for developing embedded systems that include both programmable logics (FPGA) and dual-core ARM Cortex-A9 processors. Because onboard flash memory for storing data and DDR3 memory will support the computational demands of both ARM cores and the FPGA and moreover with so many I/O interfaces such as HDMI, USB ports, buttons and switches makes this board suitable for our project objectives to accomplish. Also, the PYNQ framework, allows access to Jupyter Notebook enabling the use of Python coding language which is also the coding language used in this project.

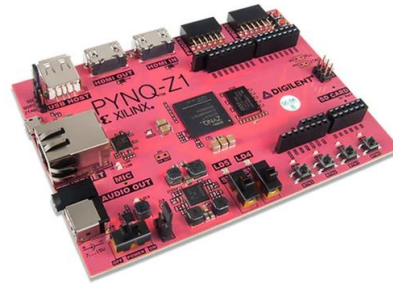


Fig. 2. PYNQ-Z1 board.

**Touchscreen Display:** The touchscreen functionality is used to add an interactive dimension to this project by creating engaging and user-friendly interfaces. For this project, the touchscreen display used is 10.1" from Joy-it, model RB-LCD-2 which is an IPS display with a maximum resolution of 1280 x 800 Pixel which runs on the power supply of 5V/3A. It comes with a brightness of 350cd/m2 with tilt angle adjustment options. The main purpose of this component is to display the GUI through the HDMI port and read the touch coordinates made on the GUI through the USB port.



Fig. 3. Touchscreen display.

**Camera Module:** The camera module's purpose is to capture the image and store it in JPG or PNG format files. The model used in this project is USBFHD08S-H110 from ELP brand and has specifications of 2 megapixels with a captured image resolution of 640x360 pixels. It is based on a 1/3-inch OV4689 sensor and runs on 5V/150mA and working temperature between -20 to 70°C. The data transfer from the camera is through a USB port.



Fig. 4. Camera module.

**Pen Plotter:** The purpose of this device is to achieve the goal of plotting the dotted images converted from

captured images. In this project, the pen plotter used is from the brand AxiDraw, made by machined and/or folded aluminum, an easel (board and clips) for paper holding and a gripper to hold the pen or any writing instrument. This plotter is powered through an external power supply the input format supported is a .svg format file and is transferred through a USB port connected to the plotter.

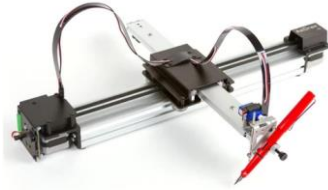


Fig. 5. Pen Plotter.

#### Other Requirements

- USB Hub: Because of the single port available on the PYNQ-Z1 board, a USB hub of at least 3 ports to connect the display, camera and pen plotter is used.
- Router and LAN Cable: connect to the IP of the PYNQ-Z1 board and access the Jupyter Notebook.
- Power cables and adapters: to power up the display, the PYNQ-Z1 board and pen plotter necessary power cables with suitable adapters are used.

#### Software and Library requirements

- Vivado: This is software used as an integrated development environment (IDE) that is necessary for the design and programming of Xilinx FPGAs and SoCs. This software facilitates the integration of pre-designed IP blocks and improves design efficiency. The sole purpose of using this software was in the part to create custom overlay for hardware acceleration.
- Python: The coding language mostly used in this project is Python because of its extensive standard libraries and third-party libraries and frameworks, which provide a wide range of modules and packages for various purposes, which in our case is to control camera, display and pen plotter which had exclusive libraries designed in python, moreover libraries are available to read and send buffer data and communicate with the programmable logics. The version running exclusively on ARM processor is Python 3.6
- Jupyter Notebooks: It is an interactive computing environment enabling users to author the live code, texts, plots etc. This is where all the codes of this project are written. This Jupyter notebook is integrated with the PYNQ board and can be connected through the IP address.

Some of the third-party libraries used in this project are listed below:

- Pillow: This library often also called as Python Imaging library (PIL) is used for image manipulation tasks such as opening, modifying, scaling, and storing as different file formats. In this project this library was mainly used to open the images stored in a path, reading pixel data, and changing the resolutions of the images to make it suitable for HDMI output available on the PYNQ board.
- PyAxiDraw: This library is used to interface with the AxiDraw pen plotter. This library facilitates tasks to plot by sending commands to the AxiDraw. PyAxiDraw also enables users to define paths, set pen positions, and control the plotting process with ease.

### III. DESIGN

#### A. Process Flowchart

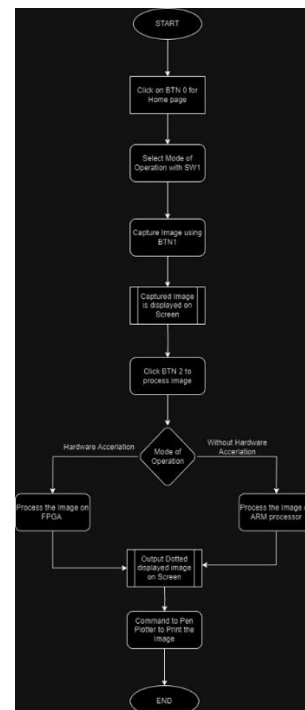


Fig. 6. Process flowchart of the project.Hardware Design

## B. Hardware Design

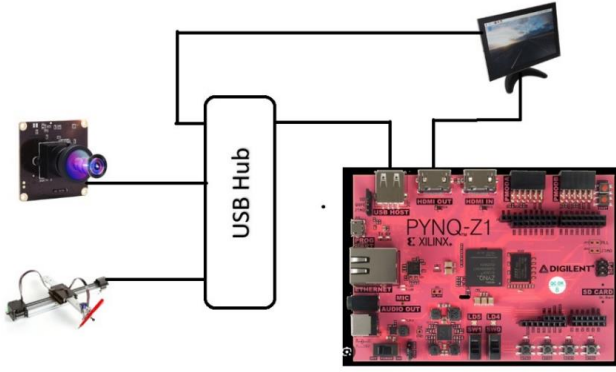


Fig. 7. Hardware connection diagram.

- PYNQ Z1 Board is the Central Processing Unit.
- Touch screen display is connected to HDMI Out of the Board.
- ELP 2 MP USB camera, AxiDraw penplotter and Touch input of Screen is connected to 'USB HOST' port of the board via USB hub.

## C. Software Design

For the implementation of the project software, a modular architecture was proposed, where the code is divided into multiple modules that will then be interconnected by a main class as shown in the "Fig 8". For the user to notice the difference in image processing, two modules were created for image processing: with and without hardware acceleration.

The principles of Single Responsibility were followed, where each module is assigned to do an specific task of the process, hence allowing the code to be cleaner and easier to work by multiple contributors.

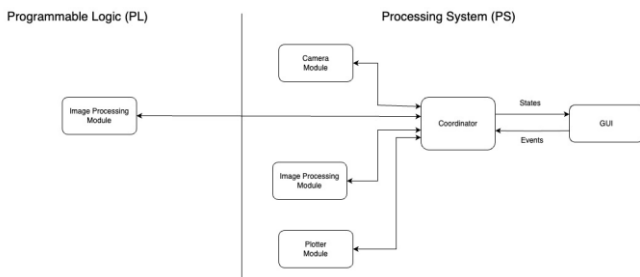


Fig. 8. Software modules interconnection.

Since the PYNQ-Z1 board contains the PS and PL modules, it was specified to which part each module corresponds.

### Modules in Processing System (PS)

- Camera Module: Contains all the code related to taking images from the USB camera.
- Image Processing Module: Contains the functions that process the image once captured by the camera.

- Pen Plotter Module: Interface to send the final product to the Pen Plotter.
- Graphical User Interface: Contains the code used to build the User Interface.
- Coordinator: It must be the class in charge of managing the communication between the GUI and the rest of the modules.

### Modules in Programmable Logic (PL)

- Image Processing Module: It must contain the interface that allows access to the FPGA to execute the image processing code with Hardware Acceleration.

## IV. IMPLEMENTATION

### A. Board Setup

The PYNQ-Z1 board was set up as per detailed instructions mentioned in the documentation from the official website. However below are the following steps explained in brief which were performed to setup the board:

1. Preparing the Micro SD card: The Pynq-Z1 SD card image v3.0.1 was downloaded from <http://www.pynq.io/board.html>, and this file was extracted and written to SD card using Win32 Disk Imager program.

Downloadable PYNQ images

If you have a Zynq board, you need a PYNQ SD card image to get started. You can download a pre-compiled PYNQ image from the table below. If an image is not available for your board, you can build your own SD card image (see below for details).

| Board           | SD card image | Previous versions | Documentation       | Board webpage          |
|-----------------|---------------|-------------------|---------------------|------------------------|
| PYNQ-Z1         | v3.0.1        | v2.7 v2.6         | PYNQ setup guide    | TUL Pynq-Z1            |
| PYNQ-Z1         | v3.0.1        | v2.7 v2.6         | PYNQ setup guide    | Digital Pynq-Z1        |
| PYNQ-Z1         | v3.0.1        | v2.7 v2.6         | GitHub project page | TUL PYNQ-Z1            |
| Kria KV260      | Ubuntu 22.04  |                   | Kria PYNQ setup     | Xilinx Kria KV260      |
| Kria KV260      | Ubuntu 22.04  |                   | Kria PYNQ setup     | Xilinx Kria KV260      |
| ZCU104          | v3.0.1        | v2.7 v2.6         | PYNQ setup guide    | Xilinx ZCU104          |
| RFSoc-2x2       | v3.0.1        | v2.7 v2.6         | RFSoc-PYNQ          | XUP RFSoc-2x2          |
| RFSoc-4x2       | v3.0.1        | v2.7              | RFSoc-PYNQ          | XUP RFSoc-4x2          |
| ZCU111          | v3.0.1        | v2.7 v2.6         | RFSoc-PYNQ          | Xilinx ZCU111          |
| ZCU108          | v3.0.1        | v2.7 v2.6         | RFSoc-PYNQ          | Xilinx ZCU108          |
| UrbaniV2        | v3.0.1        | v2.7 v2.6         | Armet PYNQ webpage  | Armet UrbaniV2         |
| UrbaniV2 (exp)  | v3.0.1        | v2.7 v2.6         | See UrbaniV2        | See UrbaniV2           |
| ZUBoard 1C9     | v3.0.1        |                   | GitHub project page | Armet ZUBoard 1C9      |
| TySOM-3-ZU7EV   | v3.0.1        | v2.7              | GitHub project page | Aisler TySOM-3-ZU7EV   |
| TySOM-3A-ZU19EG | v3.0.1        | v2.7              | GitHub project page | Aisler TySOM-3A-ZU19EG |

For the Kria KV260 and KV260, follow the links above for getting started with the Ubuntu image, and then follow the Kria PYNQ setup instructions to install PYNQ.

Fig. 9. Booting the board image.

2. Setting up the board jumpers: Since this PYNQ-Z1 was powered by external power regulator, the power jumper labelled JP5 was set to REG, instead of USB. Boot jumper was set to SD card mode because the board setup was through SD card. SD card was later inserted into the SD card slot available on the back side of the board.



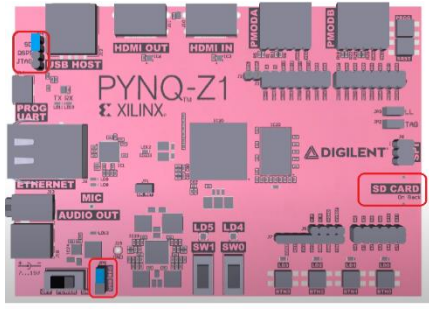


Fig. 10. Jumper connections and labels.

3. Connecting to the board: Connecting the board to the ethernet router and the computer is connected to the same network.

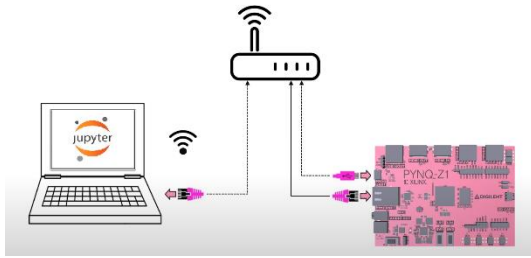


Fig. 11. Ethernet connection.

4. Switched the Power on and the LD13 light turned on to confirm the power supply. LD12 after some time was confirming the bitstream loading was completed. LD5, LD4 and LD0 to LD3 were turned on and blinking indicating that the boot was completed successfully completed.

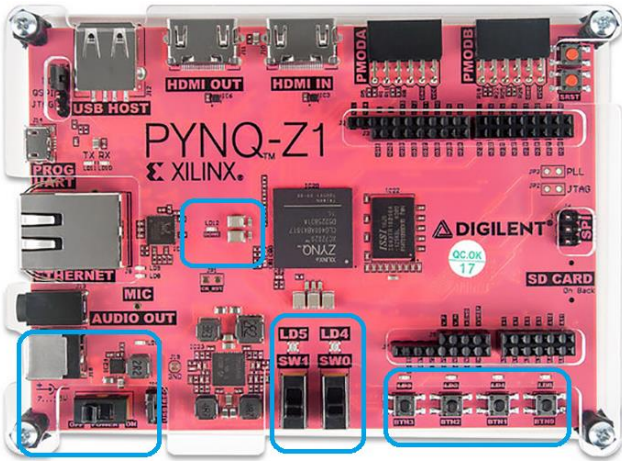


Fig. 12. PYNQ-Z1 Leds, switches and buttons.

5. Accessing the Jupyter notebooks logging in through IP address and password. Board was restarted once it was booted and configured.

#### B. Camera Module

For capturing the image, the *fswebcam* library from linux was used. Since the camera is connected to the USB host, then a reading is made from the proper serial port and the image is then saved to a file location inside the PS folder system. Then this functionality is integrated into a class function that returns the RGB data of the image created by the Pillow python library or it can be accessed by reading the image on the path that is saved.

#### C. Background Removal

The captured image will then taken by this script. This script is used to modify brightness and contrast of the image. To access the image and perform modifications, 'PILLOW' python library was used. Specifically, 'Image' function of PILLOW library is responsible for accessing the image inside the code and 'ImageEnhance' function is responsible for changing the brightness and contrast of the image. ImageEnhance adjusts the brightness and contrast of the image by multiplying the pixel values by the brightness\_factor and contrast\_factor respectively.

Another important operation which is performed on the image in this script is background removal of the image. This is done upon the user input. To perform this 'rembg' python library is used. 'rembg' uses a deep learning model that has pre-trained for semantic segmentation. Semantic segmentation involves classifying each pixel in an image into different classes, such as foreground or background. In this case, the model is trained to recognize and separate foreground (the main subject) from the background. The model is likely trained on a large dataset with annotated images, where each pixel is labeled with its corresponding class (foreground or background). During training, the model learns to generalize this knowledge and can then be applied to new images. The program uses 'remove' function from rembg to remove the background.

The input to this library is RGB image and it gives the output in the form of RGBA where R stands for (Red): Intensity of the red color channel, G stands for (Green): Intensity of the green color channel, B stands for (Blue): Intensity of the blue color channel and A stands for (Alpha): Opacity or transparency of the pixel. A value of 0 means fully transparent, and a value of 255 (or 1.0 in normalized form) means fully opaque.

To enable background removal, the user must toggle switch SW0 to a high state. If the SW0 is in low state, then the image will be processed without removing the background.

This part of the code is not included in the final process as there was a conflict while installing 'rembg' with PYNQ libraries.

#### D. Non-Accelerated Algorithm

##### 1) Generating Luminance data from Image

This part of the code is used to get width, height and brightness list of the Image. To open the image, get image width and height and to generate RGB pixel data 'Pillow' python library is used. Using '.size' function of Pillow library, width and height of the image was obtained. To generate the pixel data, 'image.getdata()' function was used. The code uses a nested list comprehension to iterate over each pixel in the image and calculate its brightness. The result is a 2D list where each row represents a row of pixels in the image, and each element within a row represents the brightness value of a specific pixel. For the RBG image, the transformation is done with following formula-

$$B = 0.299 * R + 0.587 * G + 0.114 * B \quad (1)$$

For the Grayscale image, pixel data is saved as the it is in the brightness list.

The background removed image has 4 channels and its transformation is done using the formula-

$$B = (0.299 * R + 0.587 * G + 0.114 * B) * (A / 255.0) \quad (2)$$

This brightness list, image width and height are then sent to the algorithm for further processing.

For this case study we have used the 'Dots' algorithm to convert the image into a dotted image format. The user has two different modes to convert the image into dot format i.e. with hardware acceleration and non-hardware acceleration. The difference between these two modes is that hardware acceleration mode processes the image on the PL (Programming Logic) of the board and non-hardware acceleration mode processes the image on (PS) Programming System.

## 2) Algorithm

After the preprocessing of the image, the luminance data will be stored in a text file. This luminance data, width and height of the image are the input to the algorithm. In the algorithm, there are various configurations like 'Resolution', 'Line Direction', 'Random Direction' and 'Seed' which can be adjusted to change the output.

1. Resolution (config['Resolution']): - The resolution determines the spacing between the dots. Increasing the resolution will result in more dots being generated in the same image, effectively reducing the spacing between dots. Decreasing the resolution will have the opposite effect, resulting in fewer dots with increased spacing.
2. Line Direction (config['Line Direction']): The line direction determines the angle at which the dots are drawn. A Line Direction of 0 means dots are drawn horizontally, and 90 means dots are drawn vertically.
3. Random Direction (config['Random Direction']): If Random Direction is set to True, each dot will have a random direction within the specified angle. If set to

False, all dots will have the same fixed direction based on the angle specified by Line Direction.

4. Seed (config['Seed']): The seed is used to initialize the random number generator. Changing the seed will result in a different set of random numbers, affecting the positions and directions of the dots.

Along with these above configurations there are two more configurations i.e. image width and height which will be taken from preprocessing stage.

'post\_lines()' function in algorithm converts the pixel data into dot image. This function takes 'config' (configuration dictionary) and brightness data list as input. 'Spacing' is set to 'Resolution' from the config. Two random number generators dot\_rand and direction\_rand are initialized using the 'random.Random' class. Both numbers are generated with the same value derived from the string representation of the 'Seed' parameter in the config dictionary. Generating the random number generators with the same value ensures that they produce the same sequence of random numbers.

In the next steps the function calculates the 'x\_offset' and 'y\_offset' based on the specified Line Direction in the config dictionary. If 'Line Direction' (from configuration dictionary) is less than 90 degrees, then 'x\_offset' is calculated on the ration of Line direction to 90 and scaled by the spacing. Whereas 'y\_offset' is set to the spacing. On the other hand, if Line Direction is 90 degrees or more, the 'x\_offset' is set to the spacing and 'y\_offset' is set to the difference between the line direction and 90, scaled by the spacing. These offsets are used later when determining the positions of the dots.

Next steps involves the function to iterates over the image pixels in a grid pattern with a given spacing. For each pixel, 'pixel\_val' is extracted from the pixel data. If the pixel is outside the image boundaries, a default value of 0 is used. A probability 'p' calculated by below formulae-

$$p = (pixel\_val / 255.0) * 0.5 * spacing \quad (3)$$

In the next part program checks whether a dot should be added to the points list based on a random probability p and the configuration settings. If the calculated probability (p) is less than a random number generated by dot\_rand.random() then dot will be drawn. Config['Random Direction'] Checks whether the configuration allows for randomizing the direction of the dots. If Random Direction is set to True, the direction of the line will be randomized. If set to False, a fixed direction based on x\_offset and y\_offset will be used.

This process continues until all grid positions are considered, generating a set of dots based on the specified conditions and randomness. At the end a list called 'points' will be generated which contains X and Y coordinates of the start and end points of the line.

### E. SVG Generation

Creating the output SVG image is important as AxiDraw penplotter requires SVG image to plot. Once the image is processed, a function is called to generate svg image. Inputs to this part are 'Points' list which was generated using algorithm, width and height of the image. The code starts with creating svg file path for saving the svg image. The 'svg\_content' variable is initialized with the XML declaration and the opening tag of the SVG element. For each pair of points in the points list, a <line> SVG element is added to svg\_content with attributes x1, y1 representing the starting point, x2, y2 representing the ending point, and stroke set to "black" for the color. Finally, the closing tag for the SVG element is added to svg\_content. Once the SVG image is generated, it is then saved to the specific folder using python file operations. In summary, the code generates an SVG file by concatenating XML-formatted strings. It creates line elements based on the coordinates of the line endpoints (points) and saves the entire SVG content to a file specified by svg\_file\_path.

### F. PNG Generation

Generating output image in png format is crucial, as 'video.hdm1\_out' function of overlay can only send frames converted into Numpy array.

'Points' list generated from Algorithm, width and height of the image are required to perform this operation. To perform this, we have used 'Image' function from 'Pillow' python library. Initially, 'Image.new' creates a new white image (image with 255,255,255) pixel values) with specified width and height. Then start and end coordinates of the line are extracted from the 'points' list.

'ImageDraw.Draw' function from 'Image' draws a line on the image with given start and end points, width of the line and fill colour. We have used black colour as the fill colour (0,0,0).

In summary, this code creates a white image with specified dimensions and then draws black lines on it based on the coordinates provided in the **points** list.

### G. Pen Plotter

To plot the dotted image, AxiDraw penplotter was used. In order to perform this, 'pyaxidraw' python library was used. Once the captured image is converted into dotted svg image, 'Main\_controller' calls this code. Input to this code is the location of the output svg image. This code runs in the following steps:

1. Importing the class
2. Create the class Instances.
3. Loading the svg file
4. Plot

TABLE I. RUNTIME ERRORS FROM PYAXIDRAW LIBRARY

| Variable            | Raise Error                  |
|---------------------|------------------------------|
| errors.connect()    | On failure to connect        |
| errors.button()     | When pause button is pressed |
| errors.keyboard()   | On keyboard interrupt        |
| errors.disconnect() | On loss of USB connection    |

These errors can also be enabled by setting them 'TRUE' in the code. But this will pause the execution of the current function. If the code terminates normally, it is possible to get the error condition by checking the errors.code value.

TABLE II. ERROR CODES FROM PYAXIDRAW LIBRARY

| errors.code value | Meaning                                    |
|-------------------|--|
| 0                 | No error; operation nominal                |
| 101               | Failed to connect                          |
| 102               | Stopped by pause button press              |
| 103               | Stopped by keyboard interrupt (if enabled) |
| 104               | Lost USB connectivity                      |

### H. Graphical User Interface

#### 1) Initial Design

As mentioned in the section on hardware requirements, the display used has two functions, one to display the GUI and the other to receive the touch coordinates of the user made on the GUI. The approach to design the user interface was to create multiple image templates to display as GUI pages as in the figures shown below.

The buttons like shaped, in the images are created, to act as a button. For example, "Click here to start", "i", "Capture Image", and "Back", to Start page" are pseudo buttons to perform actions like start, info, capture and back respectively. But these are actually just rectangle box shapes in the images which occupy pixels in the images. Using the co-ordinates of the pixel of the rectangle shape, when the user makes any touch between these co-ordinates the necessary actions are performed, and the suitable GUI page is displayed based on the whether touch co-ordinates and the co-ordinates of the pseudo buttons match or not. To design this GUI, a library called OpenCV was used to read the touch co-ordinates, close any previously open windows if any and show the next suitable GUI page based on the input given by the user.





Fig. 13. Start/Home Page.

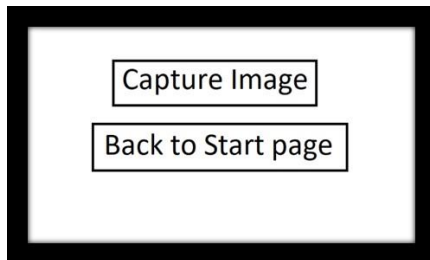


Fig. 14. Capture Image Page.

## 2) Summary of the code

A touch event handler function is created that responds to touch made on the particular region interface. The variable called “current\_window” initially set to “start\_page” keeps track of the current window and based on the touch coordinates when it matches the coordinates of the rectangle shaped like buttons, the execution of specific actions or page navigation takes place.

## 3) Challenges faced in this design

This method of creating GUI was successful when this was tested with display connected to the computer, both touch coordinates were recorded successfully, display and navigation of GUI pages in the display was also successful. But, when this GUI was tested with the PYNQ board, touch inputs were not received.

## 4) Current Design

Due to this challenge mentioned in the initial design, a different approach was to be designed such that the user’s input on the GUI no longer was by touch inputs but by using buttons available on the PYNQ board. In this GUI design approach, all the inputs given by the user is made using the 4 buttons available on the PYNQ board. The four buttons on the board are responsible for a particular action to be performed. The actions assigned for each button are as follows:

- BTN 0 – Home
- BTN 1 – Click picture and preview the clicked image.
- BTN 2 – Run algorithm and preview the dotted image.
- BTN 3 – To plot the dotted image.
- SW 1 – to enable/disable hardware acceleration.

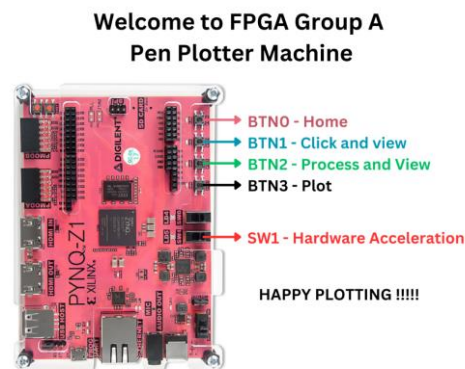


Fig. 15. Start/Home Page.

## 5) Summary of the code

First all the necessary modules and classes are imported, so that that these classes can be used to perform action when the user presses on the any button. Next, HDMI out port is accessed and configured available in the base overlay. A function is created called as “display” whose primary function is to resize the images to the resolution supported by HDMI out of the PYNQ which in this case is 640 x 480 pixels and convert it to numpy array. The function creates a new frame for the HDMI output, assigns the NumPy array representing the image to this frame, and writes the frame to the HDMI output.

Once all the classes and functions are defined and the blocks are run. The block responsible for running up the GUI is a while loop, which keeps running until BTN 2 and BTN 3 are pressed at once to stop this while loop. Inside this while loop is an if else condition statement representing a BTN condition to check if it is pressed or not and in turn responsible to perform necessary tasks like to take picture, run the algorithm, or plot the dotted image. Once when both BTN 2 and BTN 3 are pressed at once, the while loop is stopped, and the HDMI is stopped ending the GUI.

## 6) Limitations of current UI design

The main limitation of this UI design is that the instructions is not known to the user about using the UI regarding the sequence of the steps or the buttons involved to stop the UI etc. This UI design can be tricky for a user without reading or knowing the instructions.

1. Button’s Information: The only information of the functionality of the button is available in the home page, however when the user is clicking the picture, and the image is displayed for preview. The user has no access to the button information without going back to the home page.
2. Misreading switch’s state: the switch SW1 is used to enable or disable hardware acceleration. However, it will be at times confusing to know if the switch’s condition is in ‘ON’ or ‘OFF’ state.
3. Miss-hits: due to the small clearance space available to press the button, there is a potential chance that user might instead of just pressing the

button 3 which is to plot the dotted image but end up clicking the button 2 and button 3 simultaneously causing the UI to close and HDMI to stop.

**Resolution incompatibility:** The HDMI out IP on the PYNQ-Z1 board has several supported resolutions. But the resolution of the display used in this case study does not have a matching supported resolution from PYNQ. Sometimes, the images displayed on the screen have a shifted frame, the root cause reason is still unknown.

#### 7) Future developments for UI design

1. Touchscreen UI: The reason for current design of UI is solely because when the touch display is connected to the PYNQ board. The touch coordinates made on the display is not tracked.
2. Single Page UI: Having multiple navigation pages to preview the images will be more difficult to compare the results. A single page UI to view both clicked picture and the converted dotted image along with all the buttons on the same screen.
3. Information panel: A panel to the information regarding the state of the switch for hardware acceleration completed steps and plotting progress, processing completion progress.
4. LED Blinks: LEDs available on the PYNQ board can also be used along with information panel. For example, to represent the state of switch for hardware acceleration, the LED above SW1 turns blue for enable and red when disabled. Similarly LEDs above BTNn can be used to mark the completed state and sequencing the blinks for processing or plotting progress.

### I. Hardware Acceleration

#### 1) About the FPGA board

The PYNQ - Z1 board is designed in such a way that it contains two separate on-chip modules. The Programming System (PS) is composed of an ARM A9 CPU which has a Linux image (Ubuntu) installed on which a Jupyter Notebook server is running, and which acts as the interface between the user and the board. The Programmable Logic (PL) module is the part that corresponds to the FPGA where the programmable logic gates that are used to realize the hardware design to be used are located.

According to the specification in the hardware connection diagram: the USB camera, Pen Plotter and touch screen information are connected to the USB Host on the board and the touch screen should be connected through the HDMI Out port. However, in the PYNQ-Z1 configuration, not all the board ports are connected to both components, therefore, as part of the Overlay design requirement, the connections that must be made from the PL to the PS for the

User Interface code to have access to display frames through the HDMI out port must be included.

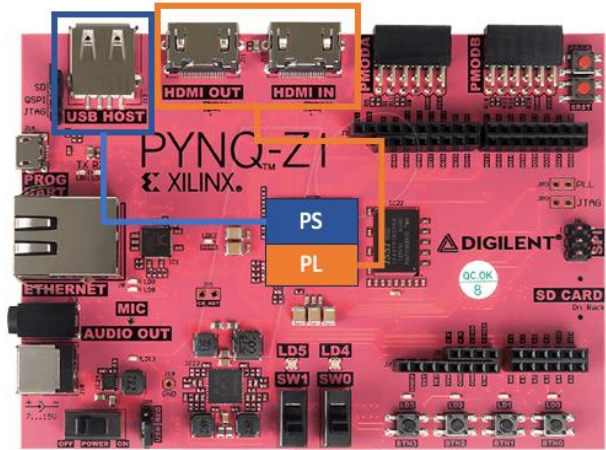


Fig. 16. Diagram with pseudo-diagram connections from PYNQ-Z1.

To carry out the use of the FPGA for Hardware Acceleration in this project, it is necessary to understand the components required for it:

- **Intellectual Property (IP):** is a functional and reusable unit of logic or integrated circuit used to create an application specific circuit or to be used in a Field Programmable Gate Array (FPGA). An IP is normally used as a module to create a larger and complex Integrated Circuit.
- **Overlay:** is a virtual reconfigurable architecture that overlays on top of the physical FPGA configurable fabric. They contain the specification for the different IP, connections and design that are to be implemented and which describe how the hardware design should be created in the reusable logic gates.

The objective behind the Hardware Acceleration feature is to run the same algorithm that is used to process the images and convert it to a point vector, using the hardware components of the FPGA. Following is a description of the design components.

#### 2) Processing Image Algorithm

A block designed to execute the dots algorithm calculations was necessary to complete the task. To achieve this, the VITIS HLS software, designed by Vivado for this purpose, was used. HLS stands for High Level Synthesis and is used to create the hardware description using programming languages without having to resort to using VHDL or Verilog.

In it, the code corresponding to the algorithm was written as a function in the C++ language and by using pragma interfaces, the relationship between the function inputs and the ports that should then be connected to the rest of the blocks was created. For this approach, a port named **I** was declared through which it was expected to sequentially

read the data from the images, and a port named **O** through which the result of the calculation is returned.

### 3) Communication

Once the block in charge of executing the algorithm logic has been generated, it is necessary to create the design that allows communicating the information between the PS and the customized block.

The AXI protocol available for the PYNQ-Z1 board components was used to carry out this communication. AXI stands for Advanced Extensible Interface and is an on-chip communication protocol that is part of the Advanced Microcontroller Bus Architecture.

The AXI4-Stream interface is a point-to-point link where the transmitter is known as a master, and the receiver a slave.

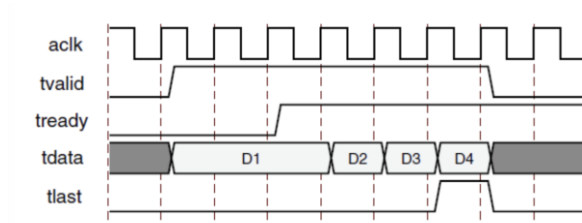


Fig. 17. Data Transfer timing diagram in AXI4-Stream Channel.

Two signals (tvalid and tready) are used to control the data transfer. The tvalid signal is used on the source (master) and indicates that the values in the payload fields are valid. The tready is used in the destination (slave) and indicates that the slave is ready to accept the data. The transfer only happens when tvalid and tready are detected on the same clock cycle as can be seen on “Fig 17” from AMD documentation[5].

Vivado software was used to create an overlay design to meet this need. For this design 3 blocks were primarily needed:

1. Custom IP Bloc
2. IP of the Processing System
3. AXI DMA IP

Blocks 2 and 3 are found by default in the Vivado library. The PS block is a hardware representation which is used to create the connections between the ports of the different hardware elements with the PS as well as for clock connections, control signals, etc. but doesn't create the Processing System in the board.

To transfer data to and from the PS, the DRAM memory included in the chip architecture will be used. It contains 4 HP ports containing internal connections to the DRAM of which two are enabled: S\_AXI\_HP0 and S\_AXI\_HP2.

The AXI DMA or AXI Direct Memory Access block is in charge of performing the data transfer between the PS

DRAM and the Custom IP Block that executes the algorithm code. The two available communication channels were enabled for the block: Read and Write. This enables two pairs of ports for Slave-to-Master and Master-to-Slave transfer between the two blocks. In the end the connection made was as follows, with the nomenclature "Block(Port)":

- AXI\_DMA(M\_AXI\_MM2S) -> PS (S\_AXI\_HP0)
- AXI\_DMA(M\_AXI\_S2MM) -> PS (S\_AXI\_HP2)
- AXI\_DMA(S\_AXIS\_S2MM) -> Custom IP(I)
- AXI\_DMA(M\_AXIS\_MM2S) -> Custom IP (O)

Some additional connections for clock signals, or AXI Lite control signals corresponding to the AXI protocol were made with the help of Vivado's automated processes that help to assign those ports to their corresponding connection thus avoiding human errors in the design phase.

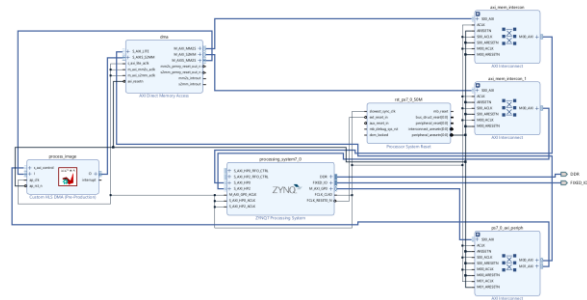


Fig. 18. Final overlay design for the hardware acceleration.

Once this design is created, we proceed to run the process to generate the hardware implementation in Vivado and then generate the bitstream and hardware\_handoff files that will be used in the PS to connect to the design.

### 4) Execution from PS

To perform the integration between the PS code blocks with the design made for the PL, the use of two modules of the PYNQ python library is required:

#### a. Overlay Module

The Overlay module is used to manage the content and status of the overlays created for implementation on the FPGA. With this package the bitstream and hardware\_handoff files created in the previous step are loaded.

Once the overlay is instantiated, the functions/information of the IP's contained in the design can be accessed. The figure shows how by accessing the register control of the custom IP process\_image, you have access to manage the states in of the block.

```

from pynq import Overlay

ol = Overlay("process_rgb.bit")

dma = ol.dma
dma_send = dma.sendchannel
dma_rcv = dma.recvchannel

hls_ip = ol.process_image

CONTROL_REGISTER = 0x0
hls_ip.write(CONTROL_REGISTER, 0x81) # 0x81 will set bit 0

hls_ip.register_map

RegisterMap {
  CTRL = Register(AP_START=0, AP_DONE=0, AP_IDLE=1, AP_READY=0, RESERVED_1=0, AUTO_RESTART=0),
  GIER = Register(Enable=0, RESERVED=0),
  IP_IER = Register(CHAN0_INT_EN=0, CHAN1_INT_EN=0, RESERVED_0=0),
  IP_ISR = Register(CHAN0_INT_ST=0, CHAN1_INT_ST=0, RESERVED_0=0)
}

```

Fig. 19. Accessing Overlay and IP through Pynq Python library module.

### b. Buffer Module

The buffer module contains the necessary methods to transfer information between Python code to DRAM. It offers good integration with numpy, so when allocating contiguous memory spaces, you can specify the size and contents using numpy arrays.

When calling the allocate method, an array is created with the numpy.shape specified in the method and additionally the function allocates the memory addresses belonging to that buffer, differentiating it from a normal array.

Two different buffers are created for the input and output data and then, through the instances created in the Overlay module, instructions are given for data transfer between the DRAM and the AXI DMA.

```

from pynq import allocate
import numpy as np

input_buffer = allocate(shape=(360,630,3), dtype=np.uint32)
output_buffer = allocate(shape=(360,630), dtype=np.uint32)

dma_send.transfer(input_buffer)
dma_rcv.transfer(output_buffer)

```

Fig. 20. Writing and reading back data from DRAM memory with Python.

### 5) Challenges and Limitations

The outcome of the implementation of the Hardware Acceleration module was affected by the following factors:

- The use of Vivado tools is contemplated for users with high knowledge of Hardware design, so the learning curve of the software to perform the implementation was slow.
- Some design errors such as bad specifications of buffer sizes, clock synchronization, etc. Did not allow the data transfer to be carried out successfully.

- The lack of tools to debug the overlay once installed added difficulty to the problem identification and resolution process.
- The Overlay designed for the project was to include access to the HDMI Out port but AMD's documentation was very technical about which modules could be used to implement this feature making it harder to be designed.

## III. SUMMARY

### A. Limitations

- 1) To get the touch input of touch screen on FPGA we tried using OpenCV and Pynput python libraries. Still, we were unable to get the touch input. Instead of this we have used buttons and switches on FPGA board to get input from the user.
- 2) We were unable to perform Hardware acceleration since we were not able to initiate data transfer between PL and PS and vice versa.
- 3) For the background removal, we have used 'rembg'. We were unable to install rembg on FPGA due to conflicts with PYNQ libraries. Also we tried blurring the image after specific distance using and 'Pillow' library's 'ImageFilter' function. The drawback of this approach was if there is slight change in the position of the object to be captured with respect to camera then output dot image was not up to the mark.

### B. Results

- All codes were executed successfully with minimal errors and warnings. The image is captured by the camera module connected and the captured image is seen on the display interface. The touch screen display response is used to capture and save the initial three-dimensional image in the specified location.
- After capturing the image, the second set of functions were executed to enable the processing of image. It is evident that the challenges mentioned limit the performance of the output produced and hence only an approximate output is generated. Moreover, the hardware acceleration capabilities of the FPGA were not been able to utilized to its full potential, thereby inhibiting the processing power further.
- The processed image is taken and is saved as a SVG file in the designated location as mentioned in the source code. The SVG file is verified for the file location and properties before making way for the next steps. It is to make sure that the file is ready for AxiDraw to line draw the image instead of requiring any additional processing.
- The AxiDraw functions are later executed and the pen plotter is interfaced with the device to initialize the plotting procedure. Upon successful execution the code starts to transform the SVG file and starts plotting using the pen on a sheet of paper. It has to be noted that the

plotter will be initialized only upon successful execution of code and the image parameters match the compatibility of the device. Additionally, due to the performance metrics seen earlier it the transfer rate of SVG file to the pen plotter and then on to the paper takes an ample lot of time. When the image captured is a detailed object and/ or has a lot of objects then this time increases further based on the complexity of the image.

- Further analysis can be made with respect to the actual image and the pen plotter output to study the image processing capabilities and to study the efficiency of processing using an embedded system built using a FPGA.

### C. Conclusion

An FPGA based embedded system is created and executed in a test environment successfully. The FPGA works along with other components interfaced to it without any significant deviations from the functional definitions provided with relevance to the process that is required. It is also to be noted that the FPGA has been programmed to process the image and print it on a sheet of paper by converting it into suitable format. Also, the libraries used for the functions have provided great significance in optimizing the liabilities of software constraints. The embedded system

design for image processing is considered a success as the design enables the processing of image by extracting relevant data and converting them into another format. AxiDraw assists by transferring the converted file output on something to see and analyze the results in comparison to the actual results.

### IV. REFERENCES

- [1] J. Lorandel, J. -C. Prévotet and M. H  lard, "Fast Power and Performance Evaluation of FPGA-Based Wireless Communication Systems," in *IEEE Access*, vol. 4, pp. 2005-2018, 2016, doi: 10.1109/ACCESS.2016.2559781.
- [2] I. Chiuchisan, "A new FPGA-based real-time configurable system for medical image processing," 2013 E-Health and Bioengineering Conference (EHB), Iasi, Romania, 2013, pp. 1-4, doi: 10.1109/EHB.2013.6707301..
- [3] A. . Adil Yazdeen, S. R. M. . Zeebaree, M. Mohammed Sadeeq, S. F. . Kak, O. M. . Ahmed, and R. R. Zebari, "FPGA Implementations for Data Encryption and Decryption via Concurrent and Parallel Computation: A Review", *QAJ*, vol. 1, no. 2, pp. 8–16, Mar.
- [4] J. Peng et al., "Multi-task ADAS system on FPGA," 2019 IEEE International Conference on Artificial Intelligence Circuits and Systems (AICAS), Hsinchu, Taiwan, 2019, pp. 171-174, doi: 10.1109/AICAS.2019.8771615.
- [5] AMD Adaptive Computing Documentation Portal. Docs.xilinx.com, docs.xilinx.com/r/en-US/pg256-sdfec-integrated-block/AXI4-Stream-Interface. Accessed 23 Jan. 2024.