University of Strathclyde Engineering

Advancement of Audio Processing on PYNQ-Z2 Through the Implementation of Real-Time Speech-to-Text

EM401 Final Report

Student: Eva Peter

Registration: 201912153

Signature: __Eva Peter_____ Date: 30 March 2023

Supervisor: Louise H. Crockett

Acknowledgments

"I hereby declare that this work has not been submitted for any other degree/course at this University or any other institution and that, except where reference is made to the work of other authors, the material presented is original and entirely the result of my own work at the University of Strathclyde under the supervision of Dr. Louise H. Crocket."

Abstract

Project was a guide through embedded system design utilising PYNQ framework within the processing system of PYNQ-Z2 and implementing hardware designs upon ZYNQ-7020 programmable logic fabric. Before project was undertaken, there was no prior knowledge of PYNQ frameworks or ZYNQ architecture, so the main motivation was for education. Along with student's educational benefit, it was desired to create something novel through extending the capabilities of the video pipeline and utilising the PYNQ-framework, in a new way. Through the recent prevalence of offline communication, it was chosen to create a widening access feature of offline speech-to-text transcription. This would enable communication worldwide, while also providing security and privacy. Transcribed videos increase the viewer's concentration and aids comprehension, further improving education.

Using Rev.ai's Software Development Kit, speech-to-text functions were generated to recognise and transcribe spoken English from a microphone stream, or a file saved locally. Rev.ai was used as it has a higher accuracy rate than other free speech recognition models available. It also transcribes local files remotely, however, still requiring cloud connection for streaming transcription. For a locally stored file, language detection was also offered. All functionality was tested, and the word error rate was determined experimentally to observe the difference in accuracy pending the speaker's home location; mainland UK speakers had the lowest error rate.

The design used hardware to accelerate the visual data while simultaneously using the processing system to transcribe audio. This provided a real-time output. Within the processing system, text generated from speech was placed into a subtitling template (box at bottom of screen containing subtitles) and saved as an image. The webcam output had the templates overlayed onto them, Vivado's Video Mixer IP was used to merge the outputs together and produce an AXI4-Stream HDMI output.

Unfortunately, hardware limitations were reached as video data was too dense. Hence, the model could only run for a few seconds before "falling over", due to lack of contiguous memory. Methods to make the system operational were discussed: changing the resolution of the input or using a different audio and visual input port. However, the problem lies that the device did not have the required properties, so error would eventually repeat itself. The evolution of ZYNQ-7000 series to ZYNQ MPSoC was utilised and, with more time, the target device for the design would be ZCU-104 device, this is available within the StrathSDR lab.

Project was successful, with the unsuccessful parts originating in hardware limitations. The project devised a new way to process audio data within the device and established a way to create an offline speech-to-text transcriber. Clear progression has been highlighted within the further work section to improve device and implement speech-to-text within logic fabric. Thus, provided an achievable way to create an offline, low-power, interpreter.

Contents

List of Tables, Figures

Tables

Figures

1. Introduction

The motivation for this project is to create a low-power, offline speech-to-text transcriber which can be used anywhere in the world, only requiring a few components. The speech-to-text (S2T) device will be able to provide subtitled output of the spoken word, in real-time, to be beneficial for real life interactions between people. Design would require very few components, with only a small screen to output text, battery to power the low power board, and a microphone to take in the speech signals. From this front, an appliance which is transportable and easy to use can be created; hence, allowing many people to benefit from its uses. Given in Figure 1 is a diagram showing the desired small-scale design.

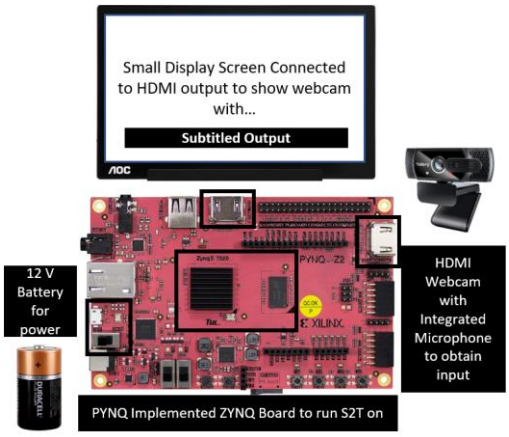

Figure 1 Desired Small-Scale Model for Design

The highlighted blocks and ports are the utilised components of the device. Figure 1 is set to highlight independent nature of the compact design. 12 V battery was chosen as the board draws 0.5 A and requires 7 V_{DC}, able to supply power for nearly 9 days. This design is the final design of the project. Currently (due to resources available) the project is laid out on a much larger scale: more connections, components, and ports in use. The number of components and the structure of interactions is explained in the hardware design section (3.3) of report.

1.1. Context

The low power remote device will have the capacity to help lots of people, through the implementation of S2T, it will allow deaf people to communicate, enhance quality of education and improve widening access features within offline broadcasting.

1.1.1. Deafness within the UK

Within the UK 87,000 people are deaf. Out of the population of 67 million, only 151,000 people can use British Sign Language meaning that a large proportion of deaf people have limited ways to communicate with the public. [1] Out of the 87,000 deaf people in the UK, it is estimated that 25,000 of them use BSL as their main form of communication, implying that 62,000 people rely on lip-reading, written word or using an interpreter. [2] Issues have been raised on numerous occasions that deaf

people feel uncomfortable with exchanging personal details (for example, medical and financial concerns) via an interpreter. Frequently interpreters are family members or close friends, access to care is not prioritised for deaf people which leaves this familiar support network having to act as carers, often young carers. [3]

Along with people being fully deaf, 1 in 5 adults in the UK live with mild hearing impairments requiring them to wear hearing aids; on top of that, 1.2 million adults have hearing loss greater than 65 dB HL: categorised severe. [4] However, as the national life expectancy increases so too the number of complications involving hearing impairments. More than 40% of people over 50 have some form of hearing loss with that growing to more than 70% of people over 70, with hearing loss growing exponentially as a person's age increases. [1]

This is why the innovation of speech-to-text was so enabling. For reasons previously mentioned, it has protected deaf people's privacy, allowed communication without sign language, and has ensured that comprehension surrounding conversations and spoken word within videos is present.

Speech and voice recognition was first tampered with by the U.S Department of Defence and universities within America in the 1970s. A robot called 'Harpy' was invented which could comprehend 1,011 words. Development stagnated until the 1990s when faster microprocessors became available to enable speech software: 'Dragon Dictate' was the world's first speech recognition software which could accept 100 words per minute, at the spritely cost of $695. [5] Now, Voice Assistants are very common with 3.1 billion searches being made monthly and most S2T modules being free of charge, thus there is plenty of access to innovation. [6]

1.1.2. Educational Enhancements

One of the main applications of speech-to-text is subtitles. Having captions on videos increase video viewing by 80%; while ensuring all viewers can consume content, it has been shown that they help improve viewers' attention and increase the likelihood of the video being watched in full. [7] Subtitling, while important for engagement and focus, is crucially needed for deaf and hard of hearing people. The National Institute for Deaf People are calling for the government to make subtitles a requirement on all video services. In 2021, Channel 4 failed to meet its subtitling quota: it is required that 90% of programming must have subtitles associated with them but only 85.41% of their programmes had access to them. [8] This is meant to be conditional to the channel's broadcasting license, demonstrating that subtitling has generally yet to be a priority with respect to delivering video and audio content.

However, what is not required is accurate subtitles. Ofcom only requires 100% accurate subtitles on pre-recorded videos and there is more concern surrounding the

pace of the words in live broadcasts: display should not exceed 160 to 180 words per minute and the delay in speech to text must not exceed 3 seconds. [9]

1.1.3. Offline Broadcasting: *Neutral Wireless*

Remote broadcasting is essential for communication; especially when not all necessities are available to the broadcaster. Remote broadcasting removes the need for mains electrical power, cloud-based servers and fixed filming locations. Hence, low power (renewable energy or batter powered), and portable private networks are deployed. This is specifically useful for geographical locations unconnected to the internet or have no radio or film broadcasting houses for widespread communication. These areas have their education impacted upon due to these limitations as well. [10]

Strathclyde spin-out company *Neutral Wireless* is working within the forefront of 5G remote production through their innovation of "Network in a Box"; live production deployment method for broadcast within a private, portable, and flexible server. [10]

Areas of engineering design which proved to be challenging were developing architecture and workflows, efficient testing of equipment and demonstrating that 5G-NPN radio can be utilised for global production. This uses Low Earth Orbit Satellite for backhaul. Backhaul is the section of the network which connects the core network to smaller subnetworks: that is extend internet connection from elsewhere. [11] Low Earth Satellite had to be proven to provide enough geographical data for remote communications. Shown in Figure 2 is the high-level workflow for the design.

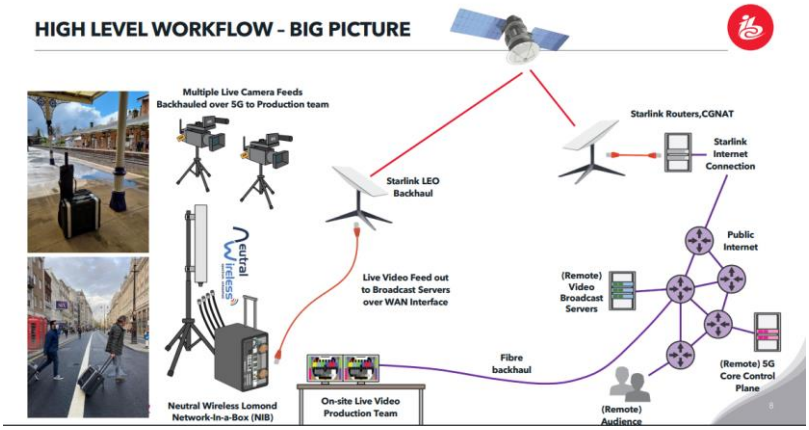

Figure 2 Operation of 5G-in-a-Box [10]

Neutral Wireless' "Lomond Network-in-a-Box" design has already been used in various locations such as broadcasting traditional music festival 'The Homecoming' in August 2022 in Ireland, Pitlochry highland games, and was even introduced at Silverstone Moto Grand Prix last year. The design has also been used within school in New Zealand to make a documentary for the Māori TV studios and live stream talks for Wildlife Conservation charity 'Ol Pajeta' in Kenya. [12]

The similarities hosted between all the present examples of 5G remote broadcasting are that they have been held in areas away from large cities, hence the likelihood of their connectivity being adequate is slim and the events have been relatively small scale, therefore not financially viable to have a full-scale portable television studio brought in. Remote broadcasting is also used frequently in nature documentaries, broadcasts from war-stricken areas and events such as concerts, sporting matches or special circumstances like Royal Weddings. [10] This is because they offer security.

This won *Neutral Wireless* "Project of the Year" at the IBC Accelerators 2022 award show. This is an awards show dedicated to promoting excellence and innovation within the design of broadcasting. [12]

There is a requirement for remote broadcasting and working outside the realms of highly powered interconnected systems: necessary for secure, remote, and confidential communications. But what about their subtitling system?

1.2. Sustainability in Practise

Speech-to-text is a well-established method within the digital world, but it is a communication method rarely used in real life. S2T is mainly used within smart assistants in phones and home devices, call centre transcriptions, and voice commanding. [13] Most speech-to-text APIs contact cloud-based servers or use large computing farms to provide the computation of speech signals; CO_2 emissions, globally, from cloud computing range from 2.5 to 3.7% which is larger than the 2.4% which come from commercial flights. [14]

This design is used to target various of the sustainable development goals (SDGs) created by the United Nations to ensure everyone is living with prosper in peace, by 2030, targeted sustainable development goals are given in Figure 3. [15]

Figure 3 Targeted SDGs within Project [15]

With regards to no poverty, it will not necessarily reduce poverty. However, it will ensure that people living in poverty will not be discriminated against due to this. As there are few elements to the device, the cost will be relatively low and will not require further payments such as supplying electricity to charge, Wi-Fi or other subsequent fees to generate the service or specific storage conditions.

As the system is fully remote, it can be used anywhere which means there is no set requirement for the user to be in a well-connected area. This would allow rural areas to access the feature even if there is no available Wi-Fi or cellular connections. Thus, providing them with a connection service which they can use to operate S2T.

Expansion in industry was spoken about in section 1.1.3 with regards to utilisation within television and radio broadcasting, this is relevant for outside streams which require portable travel arrangements. Another feature which could be exploited by industry is the privacy of this design: as all communication is purely internal this is a vital requirement of the defence sector and other companies with strict non-disclosure agreements.

Quality education is a key factor which has been the focus for development. Within Strathclyde University, it is a legal requirement that all online video content has subtitling associated with it. Not only does having subtitles improve the number of members of the public who can use it, it also improves the quality of the media digested by the viewer. Various features of having subtitles are beneficial for education, such as:

- Providing clarity of the spoken subject;
- Maintaining concentration for longer periods of time;
- Aiding comprehension when the spoken dialogue is fast or mentions various factors without visual aids, such as following on with a string of thought. [16]

As mentioned previously, live-time subtitles do not have the requirement to be absolutely correct, however, they are required to be comprehensive with regard to the media shown. The reason for the slight looseness of how accurate the subtitles have to be is purely due to the lack of computational time to provide subtitles, in real-time.

Due to the various benefits of using subtitles, a hopeful application of the project is for the S2T service to be displayed within lectures at the university, thus providing subtitled output for all forms of media digested by the student.

Also, from a secretarial standpoint, as the S2T function generates a local file containing a transcription of the spoken word, this would eliminate the need to take minute notes or query what had been said within a meeting.

1.3. Aims

The majority of aims for this project focused on understanding embedded system design and the implementation of the PYNQ framework within the processing system of the board. Given in Table 1 is an overview of the aims and their status of completion.

Table 1 Project Aims and their Progress

Aim	Completion Status
1. Gain familiarity with the ZYNQ architecture, the PYNQ framework, and the composable pipeline design released by AMD-Xilinx.	Completed in Full
2. Conduct background research to understand the various functional blocks implemented as part of the video pipeline.	Completed in Full
3. Use the composable overlay to process video and generate results, and appropriately analyse them	Composable overlay understood
4. Write efficient Speech to Text detection scripts which can be used on all forms of audio (video, audio files, live input, etc).	Completed in Full
5. Analyse the capabilities and limitations of the composable overlay and method and accelerate software through hardware design of new IP	IP generation understood
6. Extend capabilities of design through providing more analysis, i.e., for photograph identification, text translation and speech recognition to other languages.	Language detection and translation implemented

The progress by which the aims were achieved is expanded upon in the discussion (section 5.2) of report.

2. Background Theory

2.1. Development of Speech-to-Text

2.1.1. Operation of Speech-to-Text Model

Recognition of speech utilises deep learning algorithms to create a single ended end-to-end model. Rev.ai uses a specific type of Neural Network (NN) called a Recurrent Neural Network (RNN), this is perfect for speech as it is designed to handle sequential data. This is due to its ability to "remember" what came before and use the previous output to inform what the next move should be. Words generally form in a similar structure within the context of a sentence, the deep learning algorithm can use previous examples to recognise that "you" generally comes after "thank". [17]

A NN mimics the operation of a brain to recognise relations between datasets. Through passing input data through a stream of nodes, all with different weights corresponding to their likeliness to occur, find the most relevant output. [18]

RNN simply add the immediate past to the present therefore operate similarly to a NN just with two inputs. Resulting in Network learning from its previous inputs for later improvement in accuracy. Diagram showing the output interacting with the input layer is given in Figure 4.

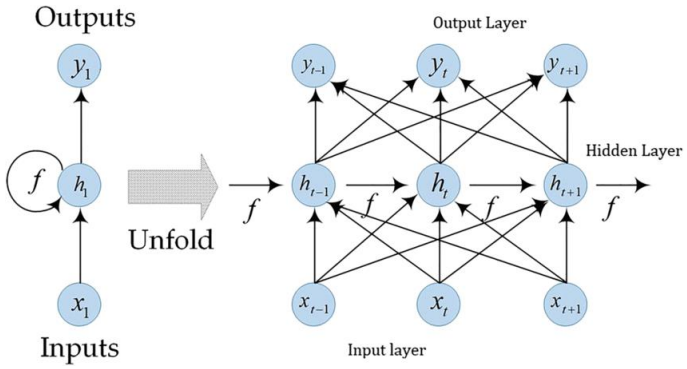

Figure 4 Recurrent Neural Network Operations [19]

However, the main issue with speech recognition is classifying text. Within Machine Learning image classification, it is simple to feed in various images of various things in order to train a model, for instance training different colours or types of dogs. Classifying speech is difficult as each person speaks within a range of their own tones and will pronounce words differently to how they seem; also, a person's rate of speech and words used depends upon whom they are speaking with. Thus, a Connectionist Temporal Classification (CTC) is deployed. [17]

CTC uses probability to align the labels words with the audio training data. [17] It is designed to be used for tasks where alignment between sequences is required but it is hard to do so. Therefore, it uses the CTC Loss algorithm where it calculated the difference between a continuous time series and a target sequence; it sums all the potential alignments of the input to target and uses probability to differentiate

between the values to find most optimum. It then maps the inputs “many-to-one” which means that target sequence must be equal or less than the input. [20]

Thus, it means that networks can be trained from pairs of words and sounds without having to specify the specific position of words, using CTC loss and the CTC decoder transforms the output of a NN into final text. Given in Figure 5 is a diagram showing how the ground truth words are compared to their pace.

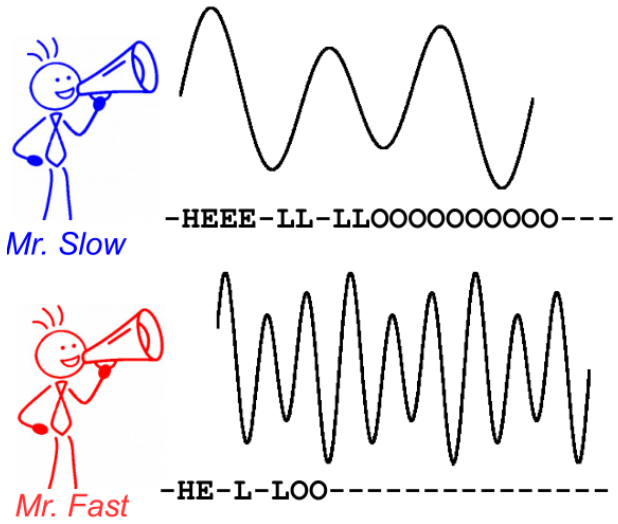

Figure 5 Diagram Showing How Different Pace of Words Affect Algorithmic Understanding [18]

2.1.2. Use of Subtitling Within Strathclyde University

Currently, within Strathclyde University, it is a legal requirement that all online video content has subtitling associated with it. To gain a more comprehensive understanding of the educational requirements of subtitled text, a meeting with Craig Bishop, from the Engineering Online Centre (EOC), was arranged. The educational structure of the EOC allows students to complete their degree while in conjunction with an industrial partner. To allow this cross-disciplinary learning, much of the teaching takes place online with only one day on campus each month. Due to the majority of learning being online in nature, certain educational enhancement tools are used, namely subtitling.

The meeting took place within the recording studio of the EOC; as suggested by its name, the studio is where lecture material is recorded by academics. Shown in Figure 6 is a screengrab from a recording of a 2nd year maths course, noting that subtitles are displayed on-screen, and a transcript is available for this video.

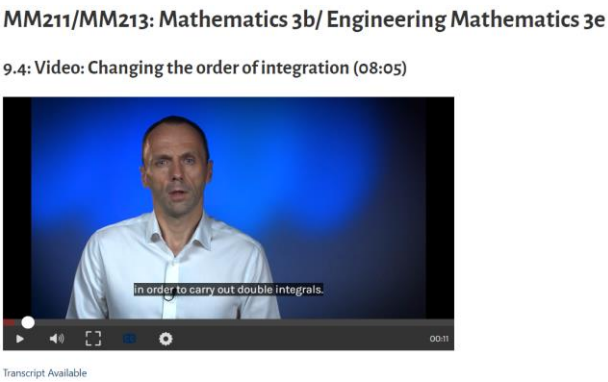

Figure 6 Screenshot of a Recording Made by the EOC

Subtitling is generated and managed through SeaCow (a Strathclyde built open-sourced front end) where educational content and the multiple subtitling methods are held. Within the UI, there are many features, key ones are:

1. A list of drop-down menus of modules used by the EOC, within the folders, it contains the various academic-created videos and their subtitling status (complete, awaiting review, not suitable for students);
2. If a certain video has not yet been subtitled, there will be an option to choose the subtitling method;
3. A content-creator approval service is needed where the respective academic has to review the subtitles from their video to ensure they are correct.

So, for every lecture video which needs uploaded to MyPlace, it must be approved by academics and pass SeaCow's constraints to have its upload approved. Given in Figure 7 is a screen grab of the UI.

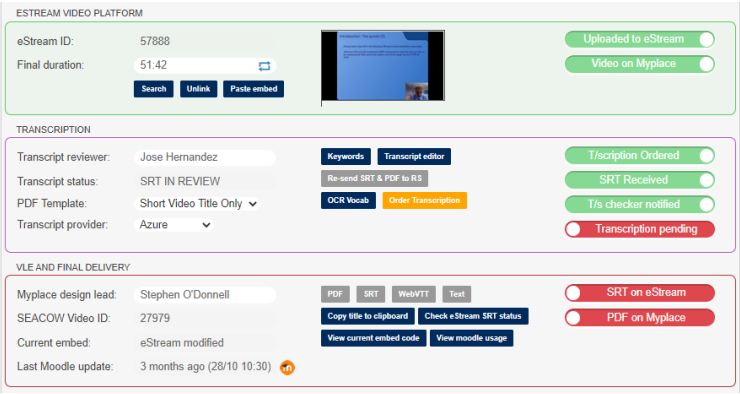

Figure 7 UI of SeaCow showing the Transcription Status of a Video

From the figure, it is apparent that Azure was the S2T API of choice however it has not been sent for transcription yet. Implying that the academic needs to review its contents. There are various methods used through the SeaCow interface: online APIs and human reviewed premium products. Shown in Table 2 below are the different subtitling methods used by the SeaCow interface.

Table 2 Subtitling Types used by the SeaCow

Subtitling Feature Name	**Subtitling Type**
3PlayMedia	Human Reviewed Premium Product
Rev.ai	API predicted subtitles using AI
Azure	API predicted subtitles using AI
SyncWords	API predicted subtitles using AI
CATFISH	API predicted subtitles using AI
TakeNote	Human Reviewed Premium Product

The method is purely dependent on each department's budget; such that, the human provided subtitling features are of a higher premium than online APIs. For departments to spend their funds most appropriately, SeaCow automatically tracks the number of corrections made by the academic for each video, per department. It also asks each academic to rate their experience with using the facilities, so that others can benefit from their experience. That is, Rev.ai might be the most accurate service for EEE whereas MAE are more beneficial to use 3PlayMedia. Shown in Figure 8 is the feedback given from SeaCow:

id	video_id	srt_provider	srt_lines	words	changes_made	time_taken	smile_level
127	27729	1	31	304	1	108	80
128	27729	1	31	304	1	113	80
129	27730	1	155	1501	3	399	80
130	27730	1	155	1501	3	400	80
131	25391	1	88	739	4	344	70
132	25393	1	253	2326	24	977	70
133	25394	9	50	705	49	1117	50
134	25395	1	26	237	0	84	80
135	25397	1	72	596	6	244	70
136	25398	1	173	1579	6	477	60

Figure 8 Corrections manager from SeaCow

Where the "Smile_level" is an indicator of the academic's experience with using the API, i.e., if it was too time expensive or an efficient service. One feature of SeaCow which cannot be implemented in real-time is human transcription. Shown in Figure 9 is an example where the speech is guessed by the transcriber and where the transcription service is unsure what the output actually is.

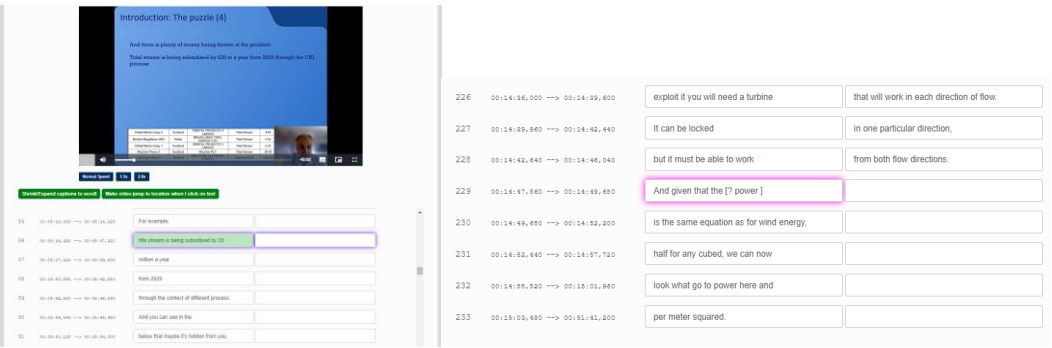

Figure 9 Predicted Speech from 3PlayMedia and highlighted ambiguity

During the conversation, it was apparent that the focus of the EOC was not only to provide the legal compliance associated with subtitling but go above and beyond. Within the compliance, it is only necessary to have subtitles, but Bishop ensures they are intelligible too. To provide this, all transcripts and subtitling features are reviewed internally to ensure they are accurate before they are sent out.

Bishop mentioned that frequent mistakes within 3PlayMedia have been that formulae have been incorrectly digested by the stenographer. Thus, with the video required to be transcribed, slides showing the content are also sent over to ensure cohesive understanding of the text.

One crucial factor Bishop mentioned was that the whole experience is time consuming and can afford to be. This is because it is concerned with providing the most comprehensive pre-recorded subtitled educational content. Therefore, unable to be used within lecture halls or meetings in real life, thus, this provided inspiration and ideas to provide a well-supported real-time service.

Figures 6 to 9 are screengrabs of the SeaCow UI, provided by Craig Bishop

2.1.3. Utilised Subtitling APIs

There were various subtitling APIs mentioned within the SeaCow section, however, some of these are paid products and not Python passable. Hence, what was wanted was an instantiable API to be used within a Python environment. Two S2T packages were used:

1. Rev.ai;
2. Google speech recognition.

Rev.ai was used within the majority of the final design as it was easier to use, provided a higher-quality transcription service and could be used offline without the addition of another model like within Google's speech recognition. Rev.ai has a lower word error rate than Google and Microsoft: 14.22%, 17.10% and 16.51%, respectively. [21]

Google's package could be configured to use various models, the utilised package was *Google Cloud Speech API* and *Microsoft Azure Speech*, which both worked online. Vosk API was briefly looked at, it works remotely through the download of a small language model. This was not utilised as the model is around 50 Mb, hence if installed onto the processor of PYNQ-Z2, there would be a severe memory usage. [22]

2.2.1. Video Data Structure

A video is a series of images. An image is a two-dimensional array of pixels, which are the smallest item of information within an image. [23] Image processing is the supreme handing of data running in parallel, an HD image has the pixel map of 1080 x 1920 which is 2,073,600 pixels. Also, as each pixel requires their colour channel, 8 bits for red, green, and blue. Overall, there are 49,766,400 bits per single HD image. [24] Given in Figure 10 is a pixel matrix representing an image.

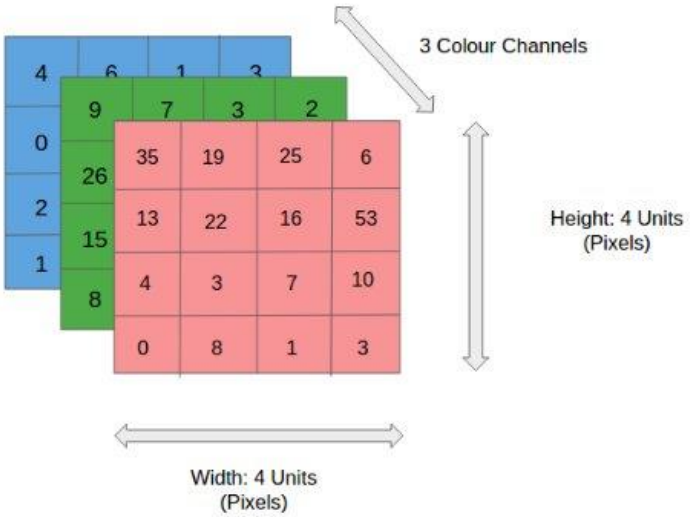

Figure 10 Pixel Matrix [22]

Each pixel is composed of three colour channels to give a resolution of RGB for the overall image. A RAW image has no processing performed and keeps a hold of all camera data obtained through sensors and requires 8 MB of storage memory; the same image requires 2 MB of memory to store as JPEG. [25] When a video is sampled at the standard 30 fps, each second of film accepts 30 images. Sampling and processing videos have a large effect on the Central Processing Unit of a computer which requires it to have downtime to not overload. [26] Most video and image processing applications are concerned with image and video compression due to the scale. [24] Here lieth the inspiration for performing the video manipulation within hardware.

2.3. PYNQ

Python Productivity for ZYNQ (PYNQ) is an open-source software framework implemented on the processing system of selected ZYNQ devices, PYNQ-Z2 was used within project. PYNQ was created to simplify the process of creating applications on ZYNQ devices: improving the productivity of designers working with ZYNQ (programme using Python instead of C) and reduces the complexity of hardware design to ensure new users can create. [27]

PYNQ framework complements pre-existing Xilinx tools (such as ZYNQ devices) to provide acceleration between the software/ hardware co-design of embedded systems through reducing the complexity of interfacing between programmable logic (PL) and processing system (PS). It has been designed for competency within:

- Parallel hardware execution;
- Video processing at high frame rate;
- Hardware accelerates algorithms;
- Low latency control and real-time signal processing;
- High bandwidth IO. [28]

Hence, a good platform to perform the project's desired features.

The PYNQ interaction and overall system methodology is given in Figure 11.

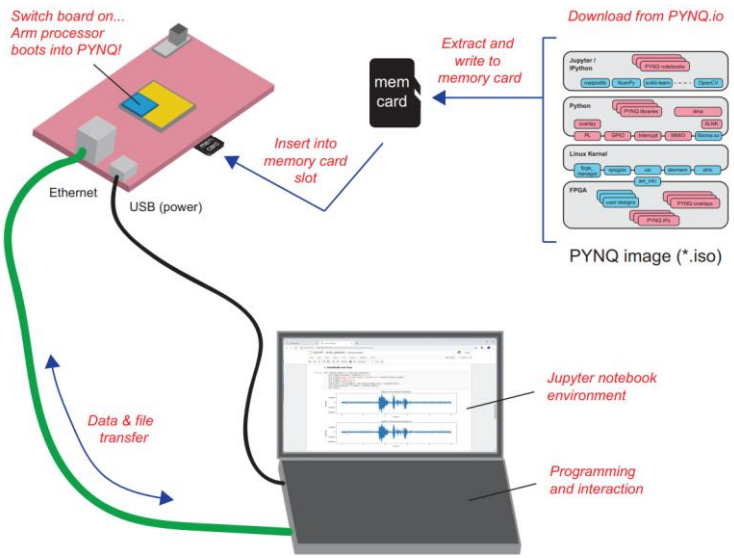

Figure 11 PYNQ operational sequence [24]

The PYNQ image states the target board, with its respective peripherals, which is loaded onto an SD card. When placed onto the board, it is instantiated within the PS and can be accessed remotely from a laptop driving Jupyter Notebooks. Each step will be explained further in the following section and later in the method write up.

2.3.1. PYNQ Framework

Given in Figure 12 is a diagram showing PYNQ's framework layers.

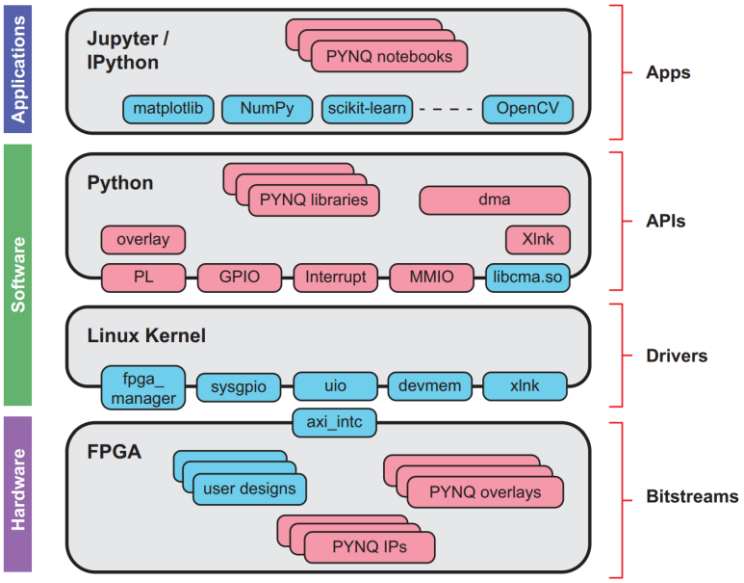

Figure 12 PYNQ Framework Stacks [25]

The top layer (Applications) is the user interface through designing and running Python code within Jupyter Notebooks, which is hosted within ZYNQ's ARM processer on the device, this is the PS. However, within the notebook, code can be offloaded from the processing to hardware models operation within the logic fabric of device; user can use hardware blocks in a similar manner to how you can call functions within software. [27]

Software layer consists of drivers and functions to utilise components within the hardware within the Jupyter Notebook, such as downloading bitstream files, communicate with peripherals on board and generate PL interrupts handling, accessing memory to move large amounts of data through processor to programmable logic and back again. The lower middle layer is composed of starting the user interface within the design. [27]

Bottom layer is the user defined hardware design. This takes a bitstream file (a binary file which describes the hardware design) and calls it within the design using an "overlay". An overlay is the method of programming a hardware system into the programmable logic: this allows the user to access specific IPs within the design, specific to the target board. [27] Full definitions in section 2.3.3.

2.3.2. PYNQ Image

In order to use PYNQ within a ZYNQ device, a PYNQ SD card image has to be downloaded onto the inserted micro-SD card used within device. Project utilised PYNQ-Z2 version 2.7 as that image was compatible with the version of Vivado (2020.2) used within the StrathSDR lab.

2.3.3. Base Overlay

Base overlay is specific to the device used and essentially describes all logic interfacing to external ports such as GPIO, HDMI ports, audio ports, Pmod interfaces and Raspberry Pi and Arduino headers. Figure 13 is a block diagram for PYNQ-Z2. [29]

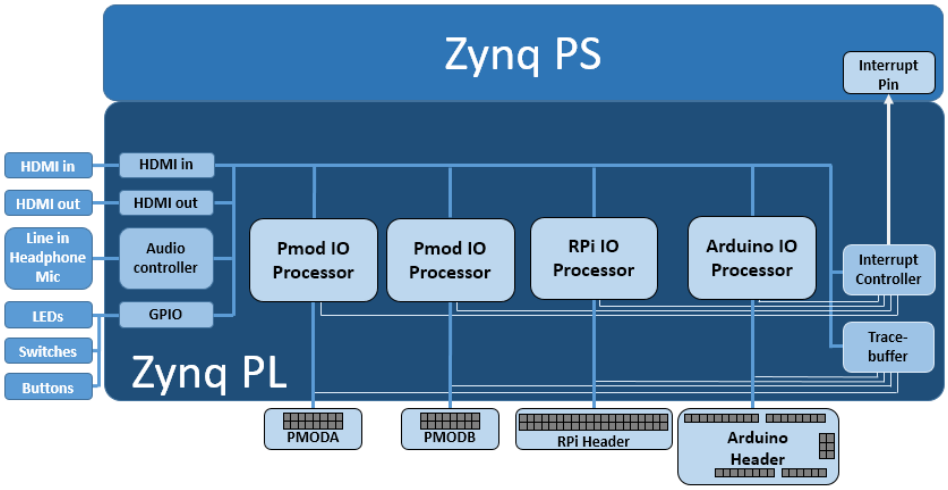

Figure 13 Block Diagram for PYNQ-Z2 Base Overlay [29]

The design process for project involved removing unnecessary blocks from base overlay and adding in different relevant blocks to be used within design; this was performed within Vivado and a new bitstream was generated. The new bitstream file was loaded into the design via Jupyter Notebooks as an Overlay. However, within the Notebook, code can be run to remove blocks or place blocks within pipeline, this is explained more thoroughly through PYNQ Composable.

Bitstream is a sequence of bits which creates a binary sequence. A bitstream is used within an FPGA to describe Hardware Descriptor Code to perform arithmetic operations, within the programmable logic. [30]

An overlay hardware implementation of a circuit, within software. [31] PYNQ Overlay is created from bitstream files which are wrapped within PYNQ Python API to be accessed from Jupyter Notebook. This allows software programmers to alter the hardware and create specialised designs, without needing to design hardware design first. [32]

2.3.4. PYNQ Composable

PYNQ Composable Pipeline is similar to the base overlay within the aspect that IP cores can be removed as and when, hence providing run-time configuration as well as hardware composability. It was designed to allow filters to be implemented onto videos with ease and provide editing of pixel quality within video, without delay. Given in Figure 14 is a block diagram of the composable video pipeline.

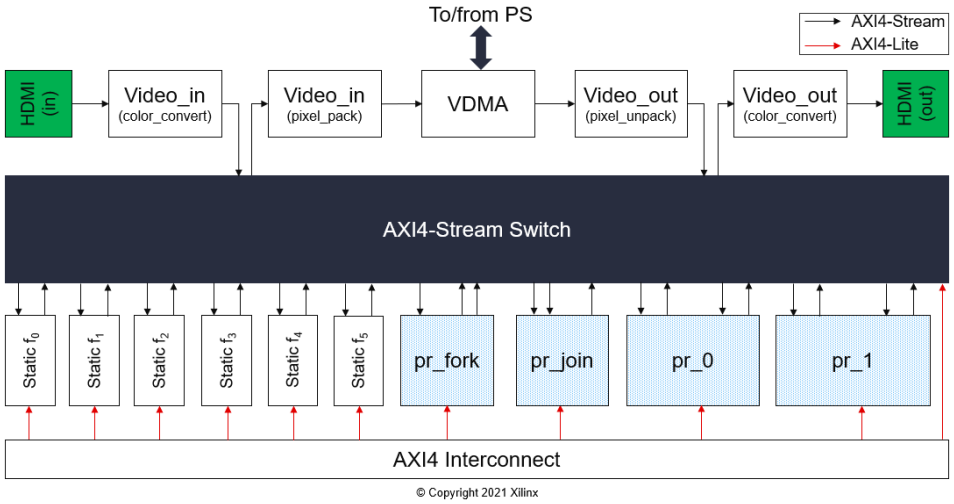

Figure 14 Composable Video Pipeline [27]

There are three main components:

- AXI4-Stream Switch defines data transfer between IP cores, hence allows design modification without redesigning overlay; [33]
- Dynamic Feature Exchange allows modification of blocks through partially downloading bit files while the remainder of logic continues uninterrupted run; [34]
- Python controllable API to alter and control pipeline. [33]

AXI is defined and explained in section 2.5.1. In Table 3, there are a list of implementable vision functions.

Table 3 IP Cores Designed to be Used Within Composable Overlay

Static Functions	**Dynamic Functions**	
colorthresholding	absdiff	erode
filter2d	add	fast
gray2gray	bitwise_add	fifo
lut	cornerHarris	filter2d
rgb2gray	dilate	rgb2xyz
rgb2hsv	duplicate	subtract

PYNQ Composable was not used directly within design as it was decided to have more of a hardware focus and design functionality within Vivado. However, it did provide inspiration for the creation of the design. This is expanded upon within the report discussion, section 5.2.1.

2.3.5. PYNQ Composable Labs

Labs are provided within the PYNQ Framework Jupyter Notebook once they have been collected from within a terminal. The labs are a guided walk through creating a composable pipeline and implementing video filters within. Given in Figure 15 is the provided labs within the Notebook.

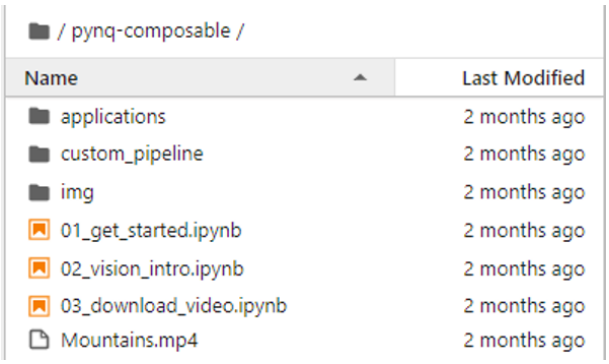

Figure 15 PYNQ Composable Overlay Labs

Each lab was examined in turn, applications provide pre-built functions, and the custom pipeline holds demonstrations as to how to generate pipelines. Applications is a user controllable set up to implement various functions using either peripherals on board such as buttons or widgets on notebook. [33] Given in Figure 16 is the list of labs available within the composable pipeline tutorials.

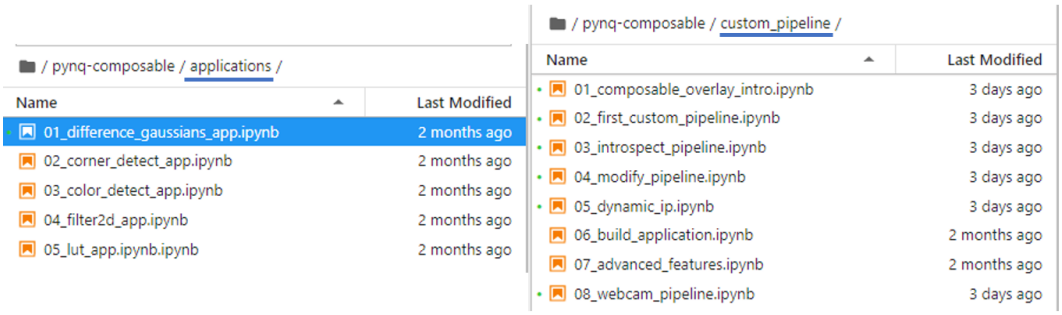

Figure 16 Composable Labs [33]

The most relevant labs were the custom pipeline as it provided an understanding of how a pipeline can be altered within software, such that it could later be reproduced within hardware. That is, the removal of insignificant components within the base overlay. An overview of the modification pipeline (lab 4) will be run through. Given in Figure 17 is the code used to create the pipeline of multiple filters.

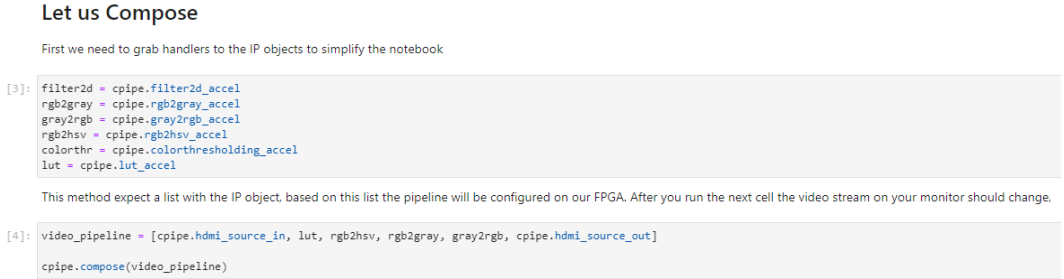

Figure 17 Pipeline Definition [33]

From the pipeline, the output was driven to the HDMI port, given in Figure 18 is the output display from the various operations.

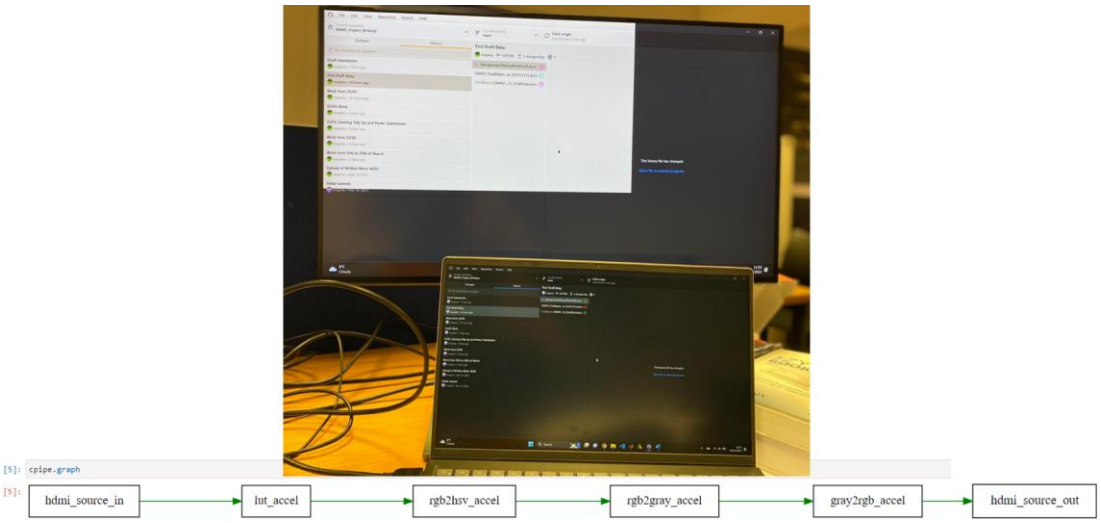

Figure 18 Manipulation of input Signal Through Composable Pipeline

It was then investigated the following manipulations which can be performed within, such as the "tap" function which essentially removes certain blocks. Given in Figure 19 is when only the first four blocks are utilised.

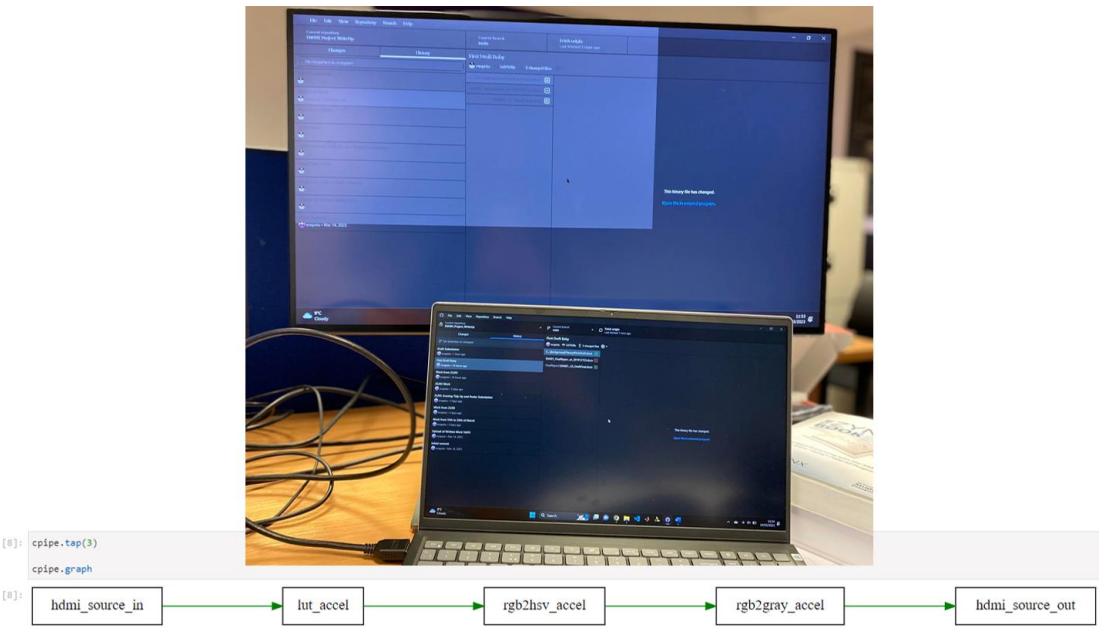

Figure 19 Altered Pipeline Output

The lab was beneficial as it was demonstrated how easily components can be substituted within design and additional IP cores removed and added.

2.4. ZYNQ Architecture

Within this section, areas of interest will consist of the components used within Vivado to model the design and the physical layout of PYNQ-Z2 device.

2.4.1. PYNQ-Z2

The target device used within project was PYNQ-Z2. It is a board with the FPGA fabric of a ZYNQ-7020 SoC which was designed for use within Xilinx University Programme to help teach PYNQ to students. [35] PYNQ-Z2 is an evolution on the first board created for learning PYNQ, PYNQ-Z1. Both boards have similarities, however the main difference is PYNQ-Z2 has full ADI audio codec with microphone and headphone port as well as a 40-pin Raspberry Pi header which PYNQ-Z1 does not have. [36] Given in Figure 20 is a comparison between the two devices.

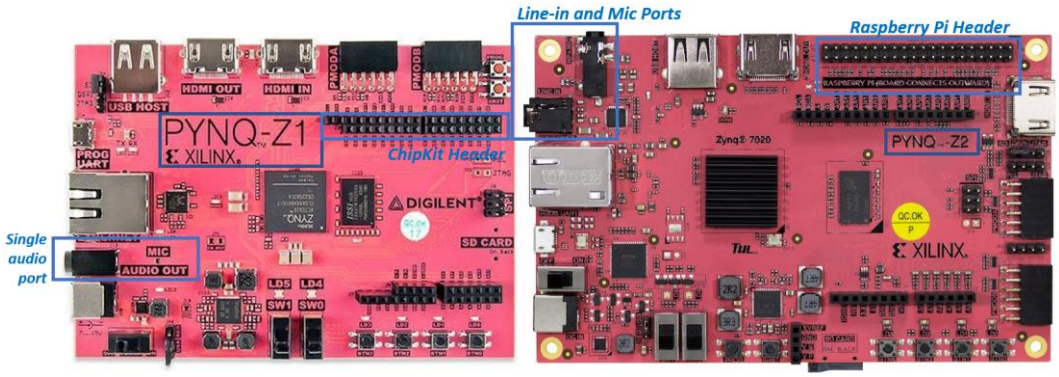

Figure 20 Comparison of PYNQ-Z boards

Within the PYNQ-Z2, as there are additional peripherals there is slightly more complexity to the device. Although there are additional ports, both devices have the same logic fabric and processor. Table 4 holds the memory and logic properties of the 650 MHz dual-core Cortex-A9 processor and the Artix-7 FPGA fabric. [35]

Table 4 Programmable Logic and Processing System Properties [29]

Property Type	Property
Logic	• 13,300 logic slices, containing four 6-input LUTs and 8 flip-flops each; • 4x clock management tiles containing phase locked loop and mixed-mode clock manager; • 630 kB of fast block RAM; • 220 DSP slices; • On-chip analogue-to-digital converter (XADC).
Memory	• 512 MB DDR3 with 16-bit bus @ 1050Mbps; • 16 MB Quad-SPI Flash; • MicroSD.

A diagram of PYNQ-Z2 device and the utilised ports within design are given within the software implementation section of project method, section 3.2.

2.4.2. ZYNQ MPSoC vs ZYNQ 7000 Series

ZYNQ MPSoC is the evolution of the ZYNQ 7000 series which provides double the processing system abilities and an increase of programmable logic structure from the 7000. MPSoC PS has a dual or quad Arm Cortex whereas the 7000 series PS has single or dual; 7000 series utilise 28 nm silicon for PL and MPSoC uses 16 nm. [37] Smaller silicon means transistors can be more densely packed across fabric.

ZYNQ-7000 was the first SoC which combined PL and PS, from Xilinx. ZYNQ MPSoC takes the pre-existing architecture and provides more sophisticated PL and PS along with larger PL area. Along with these improvements, ZYNQ MPSoC also has enhanced security measures and power management capabilities: device is split into four separate power domains for individual operation meaning that unused areas can be off. [27]

A comparison of PYNQ-Z2 to the available MPSoC devices is drawn within the further work section of the report, section 6.2.

2.5. Vivado Hardware Design

Vivado was used to alter the base overlay to create a design specific to the operation required for the project. This composed of removing unnecessary functionality from the default design, whilst augmenting the video pipeline to include Video Mixer IP block used to overlay subtitled images on top of webcam output.

2.5.1. AXI Interfacing

Belonging to ARM AMBA, AXI is a protocol for interconnect specifications, specifically allowing the connection and management of controllers and peripherals within a master design. [38]

AXI features of importance are:

- Separation between data and address and control phases;
- Byte width does not have to be constant, i.e., widths can be buffered within data transfers;
- Read and write data channels are separated, enables Direct Video Access;
- Transactions are not sequential. [24]

There are three types of AXI, given in Table 5, each with their separate purpose. [24]

Table 5 AXI Types

AXI Type	Read and Write Channels	Application
AXI4	Read and write	Memory mapped communications, **256 data** cycles per address phase.
AXI4-Lite	Read or write	**Single** memory mapped transactions; Does not support burst
AXI4-Stream	Read and write	**Unlimited** burst size for memory communications.

AXI4-Lite is mostly used for control data, such as interrupts or enable signals whereas AXI4-stream is mostly used if the destination of the signal in memory is not specific. For AXI4, the destination in memory is required. [39]

2.5.2. IP Core Generation

A Semiconductor Intellectual Property Core (IP core or block) is a hardware specification used to configure FPGA logic fabric to perform a certain function or task. They can either be soft or hard IP cores, with soft cores allowing user alteration whereas hard cores have set processes. Hard cores are normally firmly implemented on silicon therefore cannot be changed, whereas soft cores can either be altered from the block design or further down into the original Register Transfer Level (RTL) code. [24]

There are two well-established methods for creating own IP core, HDL coder or System Generator. System Generator is a Xilinx product run on Simulink to generate HDL whereas HDL coder is a MathWorks product which uses Simulink for creation. Therefore, HDL Coder creates synthesisable RTL code from a Simulink model but System Generator actually targets a block library within Simulink for specific Xilinx IP cores. [40]

System Generator and HDL Coder were investigated through the learning of classes *Further VHDL and FPGA design* and *DSP Implementation with FPGAs for System Generator and HDL coder*, respectively. HDL coder was prioritised due to its widespread usage and non-specific implementation. The generated RTL from HDL coder is automatically transferred to Vivado environment, through the Workflow Advisor, for implementation of IP creation onto target device.

The four HDL Coder labs from the *DSP Implementation with FPGAs* course were undertaken in order to understand the operation of creating an IP core within Simulink. The topics included:

- Arithmetic Operations: complex, floating point and fixed point;
- Clock Routing.

These are relevant operations required for any IP core generated.

An example design was to generate a decimal to binary converter which did not succumb to overflow conditions. The Simulink model is given in Figure 21.

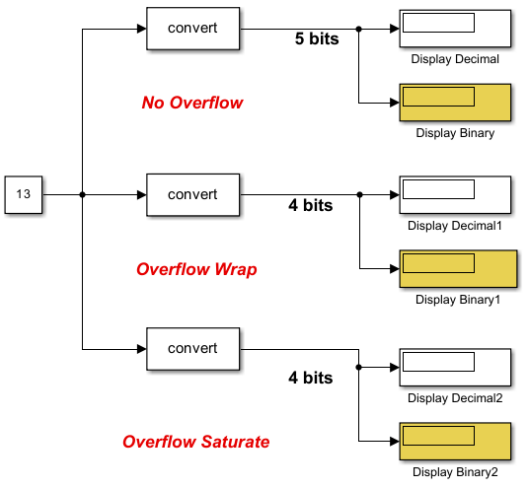

Figure 21 Overflow detector design

Overflow is when an integer value is incremented to a value that is too large to be stored in the appropriate representation. Example of an odometer (analogue scale used within car) being overflowed is given in Figure 22.

Figure 22 Overflowed Example [41]

Again, the design of an overflow was not directly used within design however it should be used within any IP core which has a counter associated. Within project design, it was not required to generate an IP core, this is addressed further within the discussion, section 5.2.2.

2.5.3. IP Cores Used in Design

Pre-existing IP Cores are beneficial for the designer as they reduce project design development time and remove the need for IP core testing. [24] Four IPs were used for augmenting the video pipeline; two additional IPs were used for video processing and two for connectivity. Video IPs:

- Video Direct Memory Access (VDMA);
- Video Mixer.

VDMA provides memory access between memory and AXI4-Stream type to allow video data to target peripherals, this is how the subtitled overlay was able to be defined as stream within design, see hardware design section of project method, section 3.3. It is a two-dimensional direct memory access which can supply asynchronous read and write channel operations. [42]

Video Mixer was the IP responsible for overlaying the subtitles onto the webcam frames. It allows up to 8 4k resolution memory or streaming layers with alpha blending available. Alpha blending allows for transparency and opaqueness at certain regions of the overlays. [43]

The two specific IPs used for AXI conversion and data transfer.

- AXI Interconnect;
- AXI-Stream Switch.

AXI Interconnect manages transactions between AXI slaves to AXI masters; that is, connecting a single master to multiple slave units or multiple masters to one slave unit. Given in Figure 23 is a pair of diagrams showing how many-to-one interactions is performed for AXI Interconnect. [44]

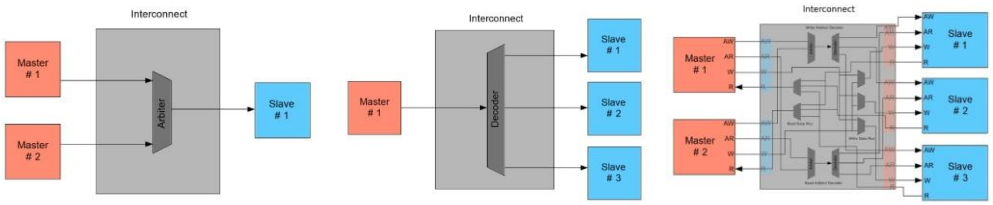

Figure 23 AXI Interconnect Internal Diagram [45]

The following figure, Figure 24, shows the Vivado IP block.

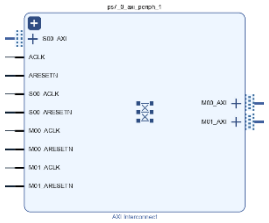

Figure 24 Vivado IP for AXI-Interconnect, diagram taken from project vivado block design

AXI-Interconnect would be used if the designer requires more than one master routed from the same slave unit but timed under different clock frequencies or reset pulses, or vice-versa. [44] Given in Figure 25 is the Vivado IP block for the AXI-Stream Switch block.

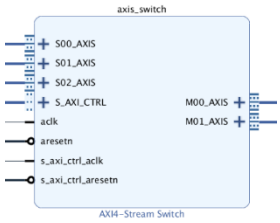

Figure 25 Vivado IP for AXI-Stream Switch

AXI-Stream Switch's operation is similar to AXI-Interconnect but is used for streaming data instead of regular AXI4 or AXI4-Lite ports. AXI-Stream Switch are normally controlled via the PS of the device, rather than purely from logic. [46]

2.5.4. Video Pipeline

Video pipeline is the capture, processing and display of an HDMI input through Vivado's methodology. Given in Figure 26 is the example video pipeline provided by AMD. [47]

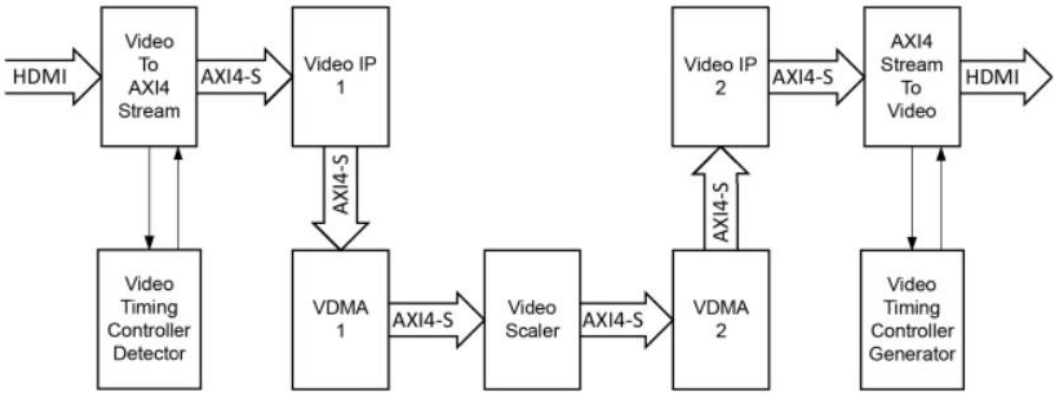

Figure 26 Xilinx's Video Pipeline

The pipeline transfers the HDMI input to AXI4-Stream before being passed through the VDMA and processing of data. Within design the first IP is a AXI subset converter to downsize the bit width from 32 bits to 24 bits before being passed into the second IP which is actually the Video Mixer.

2.6. Literature Review

A large volume of project inspiration and learning how to undertake the project came from learning resources, open-source projects and online software/hardware tutorials. However, an important factor was reviewing the work undertaken by others. Three papers were investigated to determine the relevance for the project:

1. OpenCV Based Road Sign Recognition on ZYNQ [48];
2. Convolutional Neural Network-based ECG Classification on PYNQ-Z2 Framework [49];
3. A Dual-Channel Dehaze-Net for Single Image Dehazing in Visual Internet of Things Using PYNQ-Z2 Board [50].

These three conference papers took real issues and used embedded systems to solve; primarily involving image processing and large-scale data handling.

The first paper implemented a road sign recognition function on Zedboard device (Xilinx Zynq-7020 chip, same as PYNQ-Z2) for scanning 1080 x 1920 images. Key component for autonomous driving and assistance. Using a similar format to the project, the shape recognition was performed within PS and pre-processing of images within PL. The paper found that system took roughly 5 s to process one frame. [48]

Conditions affecting the heart can be monitored and diagnosed using an electrocardiogram. Manual classification can be time consuming and very prone to human error. The paper created a deep convolutional neural network to identify ECG beats into their 5 classes. The model was deployed onto PYNQ-Z2 board and achieved an accuracy of 95.6% and has an improved performance and accuracy, compared to other technologies. [49] CNN is similar to RNN, ML algorithm as used within S2T.

Final paper looks at dehazing images taken outdoors in inadequate weather; thus, they acquire atmospheric scattering from environmental pollution which causes low-contrast and colour shifted images. Through utilising a dual-channel deep neural network, the transmission map can be estimated and utilised to compute the level of natural atmospheric light to generate dehazed image. PYNQ-Z2 board was used as a CPU to source a wide availability of test data while implementing he neural network within the programmable logic. Results from experiments show that it produces expected images, however, only under supervised conditions, which is not always possible. [50] This is similar to wanting to implement a RNN within the PL and CTC classifier within the PS, discussed further within future work, section 6.3.

Key take aways were, PYNQ-Z2 has the capacity to run neural networks using the software/hardware overlay, OpenCV functions can be easily implemented, and a full machine learning and classification system can be instantiated, on whole device. Within these relevant domains, good results were delivered. Hence, the technology and required software used within project has scope to produce qualifiable output.

3. Project Method

Project was undertaken from an introductory understanding of embedded system design. However, student had strong Python skills. Hence, to gain an understanding of what the final system should be able to produce, a software model was created to figure out how this could be attained. The project was built in three sections and will be explained as such:

1. Design of the Software Model;
2. Integrating Software to run within PYNQ Framework;
3. Design of Hardware System.

Due to the structure of PYNQ, having a well-established software design was beneficial for the reusability of code: a lot of functions were able to be uploaded directly to the framework, however areas which utilised peripherals had to be adjusted. Within the framework, the video processing element was extracted from software and transferred to be performed within the programmable logic of the device. This led to some complications and slight issues, explained within the main body of text.

3.1. Software Model

A software model was designed to perform S2T functions: transcribing audio data in live time and for pre-recorded videos. From the collection and implementation of the S2T functions, the focus was then on creating an efficient display procedure to show the subtitled output on top of what was being subtitled.

3.1.1. Speech-to-Text API

The chosen S2T API was Rev.ai, this is because it had the ability to fit criteria necessary for project design. Given in Table 6 is the required features of the S2T algorithm and a comparison to Rev.ai features.

Table 6 Capabilities of Rev.ai Compared to Desired Qualities

Desired Feature	**Rev.ai's Solution**
Callable within Python programme	Can be installed as a Python SDK to computer
Accurate and reliable	Around 90% accurate for native English Speakers [51]
Ability to operate offline	Rev.ai can transcribe local files offline but live speech requires internet connection
Transcribe non-English Speech	Rev.ai offers a language detection feature
Accept Speech from more than one Speaker	Key speaker identification can be programmed within local file, however, only supports mono speech in real-time
Free to use	Once account is made, user has first 15 hours of transcription free

Rev.ai was also of the lowest word error rate (WER) compared to other available online S2T packages. Given in Figure 27 is a comparison between the available packages.

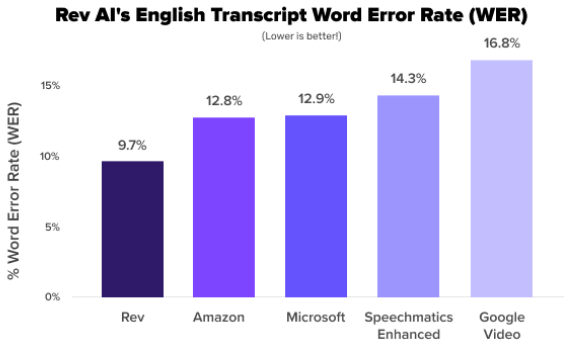

Figure 27 Comparison of WER of other APIs [51]

Rev.ai can be installed as a Python SDK directly via a terminal from the command given in Figure 28.

```
pip install --upgrade rev_ai
```

Figure 28 installation command for Rev.ai's SDK

From this installation (which was also performed within the Jupyter Environment of the PYNQ framework) Rev.ai can be utilised for understanding speech. Upon signing up to Rev.ai, an access token is provided which is the only requirement for operation. Given in Table 7 are the Python functions used within project to use Rev.ai.

Table 7 Rev.ai Python Functions

Python Function Callout	Use Within Design
`apiclient.RevAiAPIClient(token)`	Make connection to API to transcribe
`client.submit_job_local_file(filePath)`	Local File Transcription
`RevAiStreamingClient(access_token, example_mc)`	Streaming Transcription (real-time)
`LanguageIdentificationClient(access_token)`	Language Detection

As the model is trained on noisy data, its accurate and able to transcribe data from many differing sources. Rev.ai can transcribe video (.mp4, .mov tested) and audio data (.mp3 and .wav). The method for receiving locally transcribed functions is less methodical than for live-time subtitles, given in the next paragraph. Figure 29 is the list of functions required to subtitle a local file.

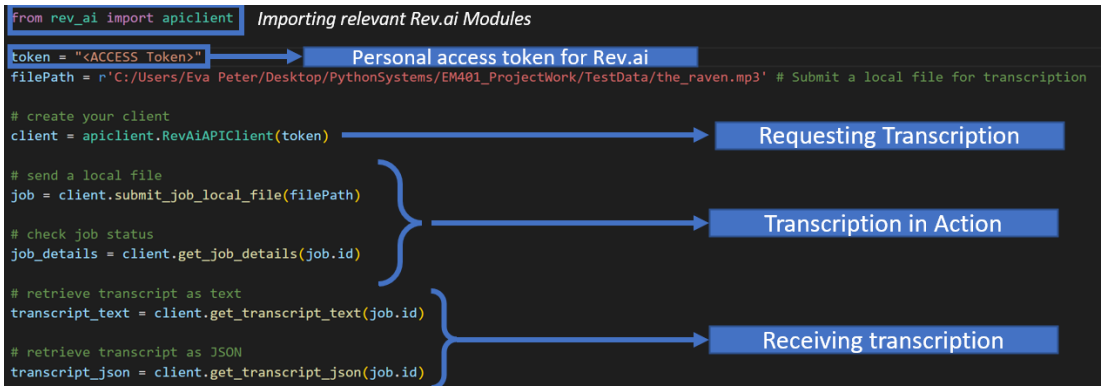

Figure 29 Local File Transcription

For the live transcription, it involved opening the microphone source and accepting in audio at its specific frequency and audio segment size. Given in Figure 30 is a demonstration for inputting a microphone stream to receive a real-time subtitled output.

Figure 30 Real-time Subtitling Functions

Rev.ai writes the text stream into json and text files. An example json file is shown in Figure 31. The file was taken from the transcription of Edgar Allan Poe's "The Raven".

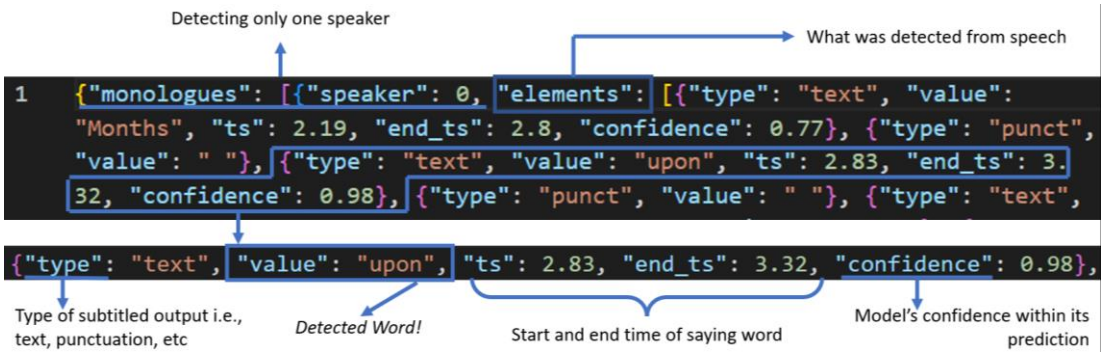

Figure 31 Example json file and its explanation

The same transcription as shown in Figure 31 also produces a text file, file is given in Figure 32.

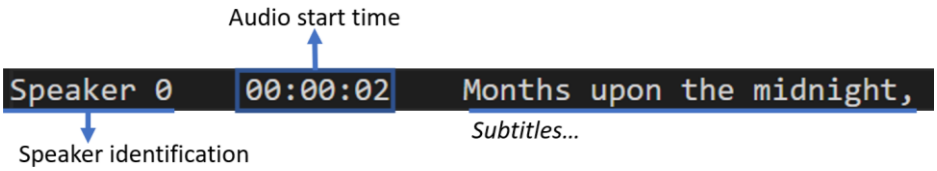

Figure 32 Example Text file and its explanation

As can be identified from this section, Rev.ai provides a quality service, however, the output requires some post processing to get the subtitles into a usable format. The format developed stemmed from the requirements set by *Ofcom* for supportive broadcasting, these include:

- Subtitles are on the screen for no less than 6 s;
- Subtitles are cohesive with the content on the screen: i.e., changed when topic of conversation is altered;
- Subtitles are of size and number of words which is digestible for user: no more than 180 words per minute. [9]

Hence, from the guidelines, the following script was created to output the correct number of words in a readable format. The script and its explanation are given in Figure 33.

Figure 33 Subtitle Structuring Script

3.1.2. OpenCV Display Functions

OpenCV was used to write the subtitles onto the webcam frames. Initially, for the software design, subtitles and their display box were written directly onto the webcam frames. Within the hardware design, this could not be performed and instead the subtitles had to be set within a secondary layer. Thus, a faux transparent frame of identical parameters to the webcam input had the content written on it. In the PL, this was extracted from memory and overlaid on top of the webcam frame.

For a local file, once the subtitled had been generated, the S2T functions provided an array of subtitled words as well as their start and end times. From this, subtitles were able to follow Ofcom's guidelines and be displayed onto the frame. Thus, a subtitled version of the local file. Start time was converted into start frame; programme iterated through frames therefore was more likely to reach a specific frame rather than a specific time. Time was converted to frames through extracting the value of frames per second in the video and performing a simple conversion. Shown in Figure 34 is the transfer from seconds to frames.

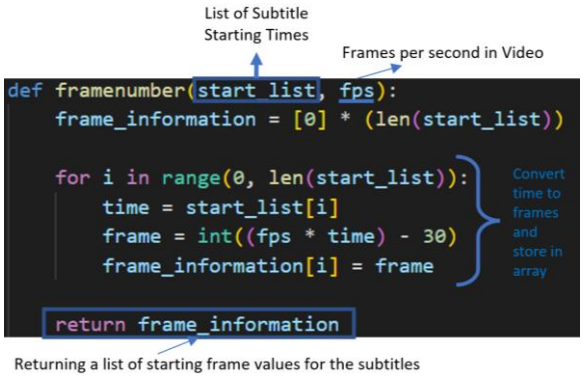

Figure 34 Function to convert time to frames

Figure 35 has the function used to write subtitles onto their specific video frame.

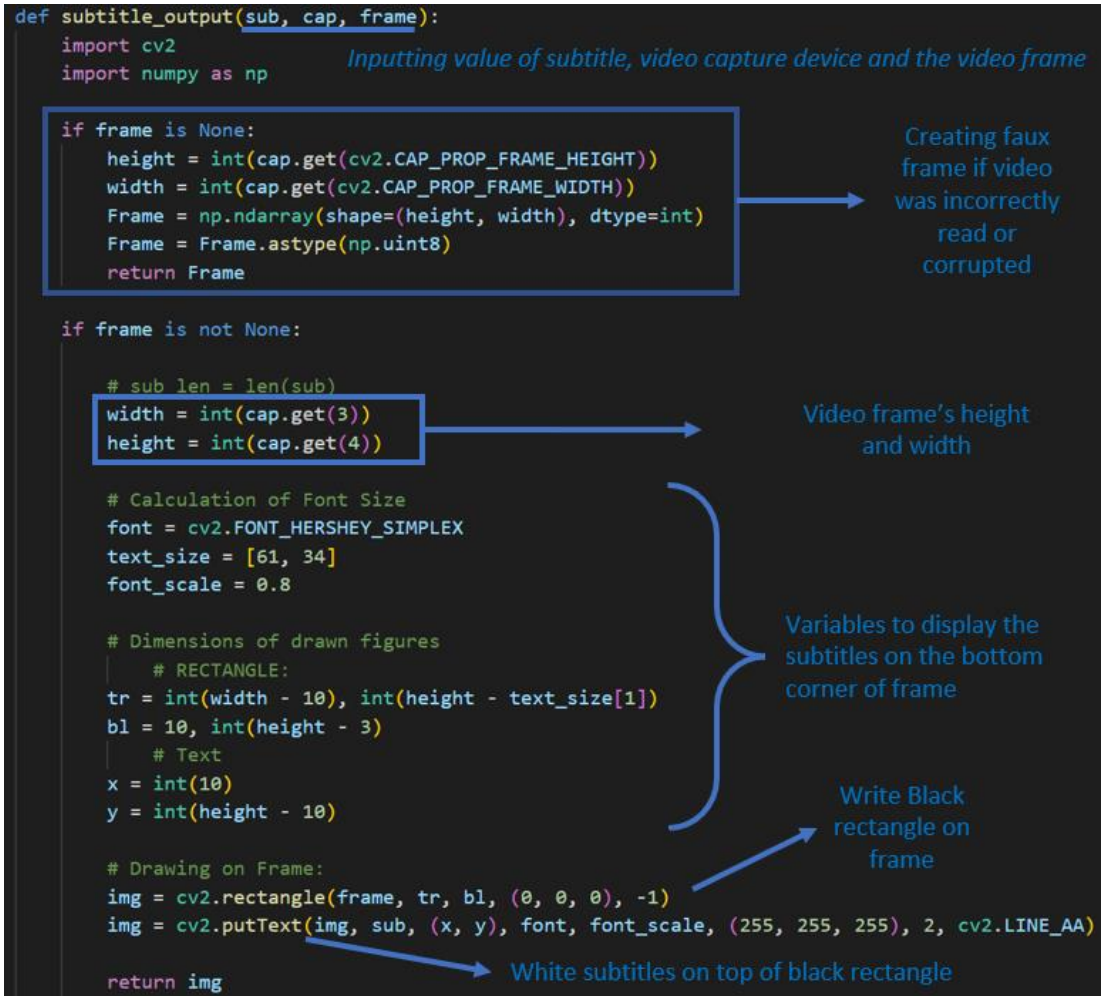

Figure 35 OpenCV to write on Video Frame

The output from using both these functions meant that a local video could be transcribed. Given in Figure 36 is a still from a downloaded video in which subtitles were applied to.

Figure 36 Screenshot from the generated transcribed output of The Raven

For a live stream, subtitles were immediately shown on screen, hence a slight lag.

As previously mentioned, the "write on frame" function could not be used within hardware. Instead, a faux frame of the same dimensions as the incoming webcam input was created. This faux frame was completely transparent except for the black rectangle containing the subtitles, it was opaque; this was enabled through utilising an alpha channel within the RGB image. This meant that the faux frame was written on, instead of the actual video frame. Given in Figure 37 is the code used to generate that.

```python
def overlaid_image(frame):
    import cv2
    import numpy as np
    # Getting the height and width of the dummy frame:
    height, width = frame.shape[:2]
    # Defining an alpha column for transparency
    # Defining the false overlay image:
    data = np.zeros((height, width, 4), dtype=np.uint8)

    # area of interest, where subtitles are placed:
    tr = int(width - 10), int(height - 34)
    [a, b] = tr
    bl = 10, int(height - 3)
    [c, d] = bl

    img = cv2.rectangle(data, tr, bl, (0, 0, 0), -1)

    for h in range(0, height):
        for w in range(0, width):
            if c <= w < a and b <= h < d:
                data[h, w, 3] = 255
            else:
                data[h, w, 3] = 0
    return img
```

Transparent everywhere except rectangle

Img -> Faux_frame

```python
def sub_output(sub, faux_frame):
    import cv2
    # Making a Copy of the Frame in order for no Overwriting:
    clone = faux_frame.copy()

    # Frame Dimensions:
    height, width = faux_frame.shape[:2]

    # font constraints for opencv put text:
    font = cv2.FONT_HERSHEY_SIMPLEX
    font_scale = 0.8

    # Image Variables to display text on rectangle:
    x = int(10)
    y = int(height - 10)
    image = cv2.putText(clone, sub, (x, y),
                        font, font_scale, (255, 255, 255),
                        2, cv2.LINE_AA)
    return image
```

Same as before, writing on copy instead of video frame

Figure 37 False Image Overlay

Used to highlight the slight differences required by a hardware system operational when comparing to software.

3.1.3. Sourcing Relevant Test Data

Test data was required to determine the accuracy of Rev.ai experimentally. Due to poems being easy to read aloud and understand, they were selected to perform the comparisons. Also, as a poem is a set piece of text, it was easy to observe the accuracy of S2T.

If the poet had died more than one hundred years ago, their work is within the public domain therefore free to use. Many educational resource videos were used and poems from Keats, Poe, Wordsworth, and Austen were all transcribed.

All Python code was created within the first semester of project and the majority was utilised within the PYNQ Framework also.

3.2. Integrating Software to Run within PYNQ Framework

For the PYNQ Framework design, various ports had to be used. Given in Figure 38 is a diagram highlighting every peripheral used on the PYNQ-Z2 throughout the entirety of project.

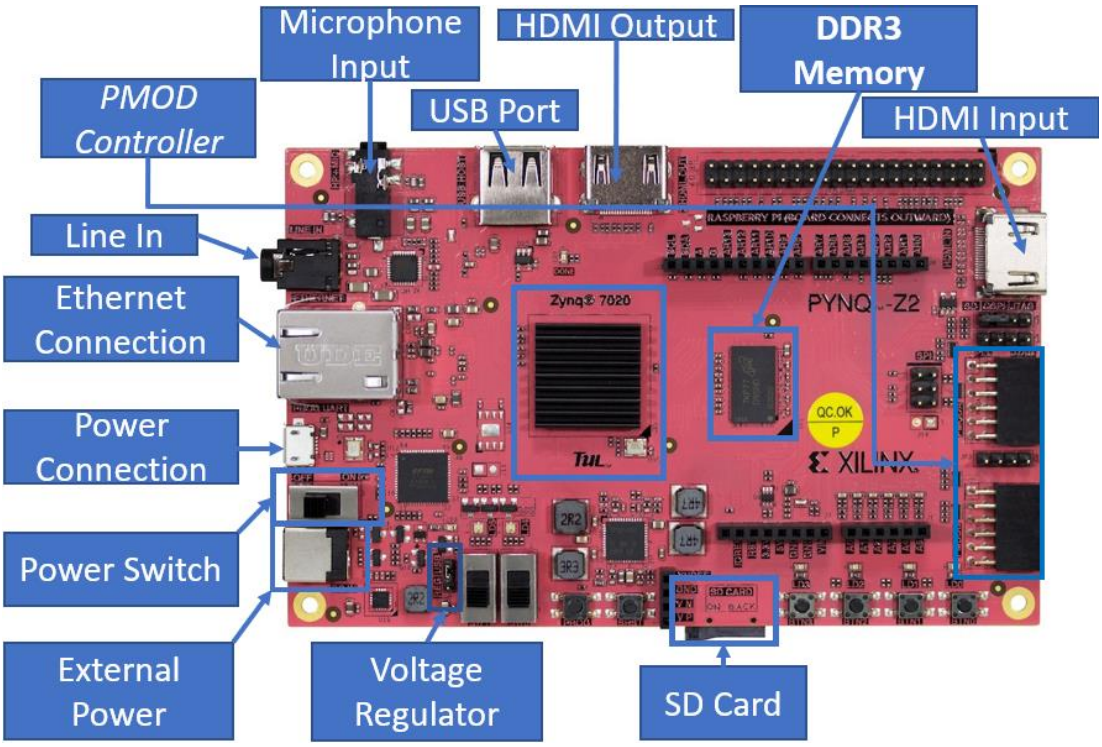

Figure 38 Utilised peripherals and components of PYNQ-Z2

Given in Table 8 is the area of the design which the peripherals are used for.

Table 8 PYNQ-Z2 Peripheral Usage

Component	**Design Utilised**	**Purpose**
SD Card	PYNQ Framework and Hardware	Store bitstream and Jupyter Notebooks code
Voltage Regulator	PYNQ Framework	Regulate Device to accept more than 7V
External Power	PYNQ Framework	Provide more than 7 V if necessary
Power Switch	PYNQ Framework and Hardware	Turn device on or off
Power Connection	PYNQ Framework and Hardware	Micro-USB connection to laptop
Ethernet Connection	PYNQ Framework and Hardware	USB connection to laptop, use Wi-Fi
PMOD Controller	Hardware in Future	Real Time Clock Routing
Microphone input	PYNQ Framework and Hardware	Provide Audio Data
Line In	PYNQ Framework and Hardware	Provide Audio Data
USB Port	PYNQ Framework	Accept USB Camera Input
DDR3 Memory	PYNQ Framework and Hardware	Double Data Rate 3x Synchronous DRAM
HDMI Input	Hardware	Provide Input Video Data to Device
HDMI Output	Hardware	Display Created Output from Device

As hinted upon by the OpenCV Display section (3.1.2), there are slightly different requirements necessary for hardware systems compared to software. Two main differences found within the conversion to software run within the PYNQ Framework, was obtaining an audio and a webcam input.

3.2.1. Physically Inputting Audio onto Device

There are two jacks which act as physical audio inputs to device: microphone and line in. Microphone jack was used for inputting live-audio and the line-in was used for playing audio files from a separate device. The line-in could be activated from just a cable connecting the jack to a laptop or phone to accept audio whereas the MIC+HP peripheral required a microphone. What was used was a pair of Apple earphones with an integrated microphone. Shown in Figure 39 is the PYNQ-Z2 set up.

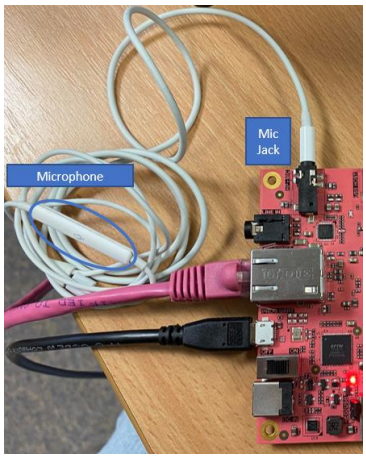

Figure 39 Section of PYNQ-Z2 showing microphone connection

Local audio, stored within memory of the Jupyter Environment, was also able to be used. Therefore, files could be uploaded and stored within a Notebook and accessed in the same way local files were accessed within the software system. One major difference between the software and PYNQ system was that audio had to be an unencrypted wave file. Therefore, when an audio file was read in, the audio file had to be converted into the appropriate structure. Shown in Figure 40 is the function to alter and decode the file.

```
with wave.open(filepath, 'r') as wav_file:
    raw_frames = wav_file.readframes(-1)
    num_frames = wav_file.getnframes()
    num_channels = wav_file.getnchannels()
    sample_rate = wav_file.getframerate()
    sample_width = wav_file.getsampwidth()
```

Figure 40 Wave File Decoding Operation

The operation took 65.4 ms of CPU time for a 3 s audio clip and 75.1 ms for a 60 s clip, therefore, not much alteration even with a larger file. However, once operation was performed, audio could be manipulated in any manner necessary. That is, it can be used as a normal .WAV file within a Python environment.

3.2.2. Audio Processing within PYNQ

PYNQ audio (pAudio) provides a method to read audio from the input microphone then read or write audio files as well as playing them back. Given in Figure 41 is the block diagram for the audio module within the base overlay.

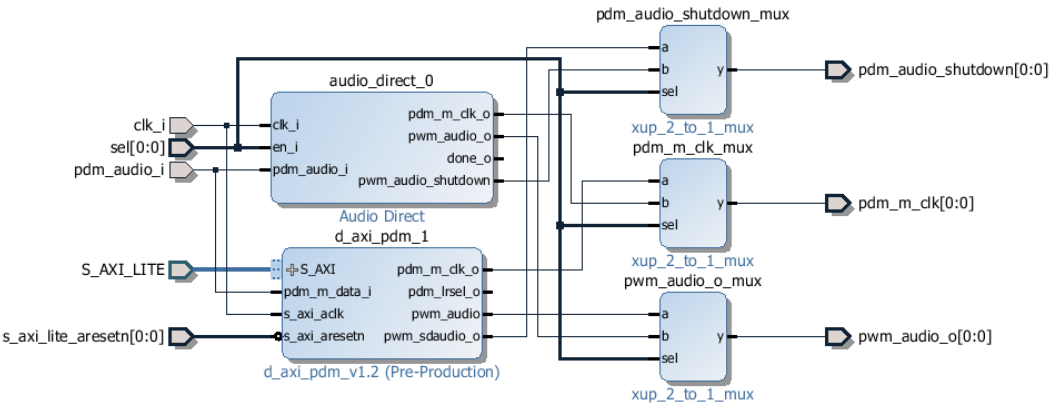

Figure 41 Audio Block Diagram Within Base Overlay [52]

The base overlay is a design to allow PYNQ to use the peripherals on a board without any hardware design, they are installed when the image is read onto SD card. [52] Here it is being utilised to allow audio processing to be performed.

Therefore, pAudio can be set to accept line-in or microphone as an input. Given in Figure 42 is example code block for accepting microphone as the input audio quantity.

```
[2]: pAudio = base.audio
     pAudio.select_microphone()
     pAudio.set_volume(20)
```

Figure 42 Still from a Notebook setting the audio contraints

Once, peripherals have been defined, microphone can record audio onto the board; audio inputted to board is stored as a buffer. The buffer's padding was determined experimentally through trial and error to be padded by 8 bits, before the most significant bit. Given in Figure 43 is a plot of the raw audio data.

```
while not self.stopping:
    self._pynq_audio.record(self.record_time)
    self._buffer.put(self._pynq_audio.buffer << 8)
```

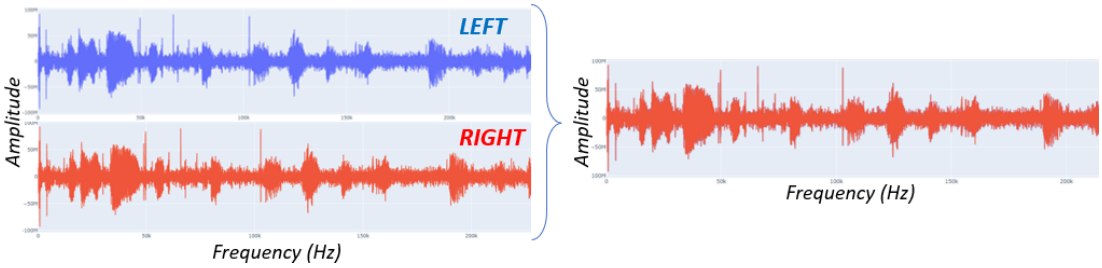

Figure 43 Audio Input to board

The audio codec for PYNQ-Z2 is ADAU1761; low power stereo audio. The audio provided is signed 32-bit, 96 kHz interleaved raw audio. [53] Hence, left and right data are both sampled at 48 kHz. Within Figure 44, the inputted audio is defined and Rev.ai streaming transcription service is called.

```
[9]: rate = 48 * 1000
     example_mc = MediaConfig('audio/x-raw', 'interleaved', rate, 'S32LE', 2)
     streamclient = RevAiStreamingClient(access_token, example_mc)
```

Figure 44 Audio Specifications for PYNQ-Z2 using Rev.ai

The recorded audio was then written into a thread and streamed constantly when called within the live-time S2T determination. This provided a real-time output of the spoken word. The computation was performed within the PS of PYNQ-Z2, which it remained, even within the software-hardware system. The same programme was used to call the live-time transcription function as was shown in Figure 30.

3.2.3. USB Webcam Display

A USB webcam was initially used to directly "copy" the work done over the course of the previous semester. However, work could not be directly copied due to the difference between running code in hardware against within software.

The plan for the USB webcam was to create a purely software system run within the processing system of PYNQ-Z2. This would be used to compare the frame rate of the processing system of Z2 and previously created software model, section 4.2.1. Given in Figure 45 is an example output called from Jupyter Notebooks utilising the USB webcam.

```
[1]: from PIL import Image as PIL_Image

     orig_img_path = '/home/xilinx/jupyter_notebooks/common/data/webcam.jpg'
     !fswebcam  --no-banner --save {orig_img_path} -d /dev/video0 2> /dev/null

     img = PIL_Image.open(orig_img_path)
     img
```

[1]:

Figure 45 USB Webcam output

An integrated system taking an audio input to transcribe and display on top of the webcam input was developed. The real-time subtitling utilised the microphone input to have S2T functions performed, within the PS. Also, within the PS, the webcam was streaming its input and having subtitles written on the frames with every changing subtitle. Given in Figure 46 is the layout for performing the live-time subtitling.

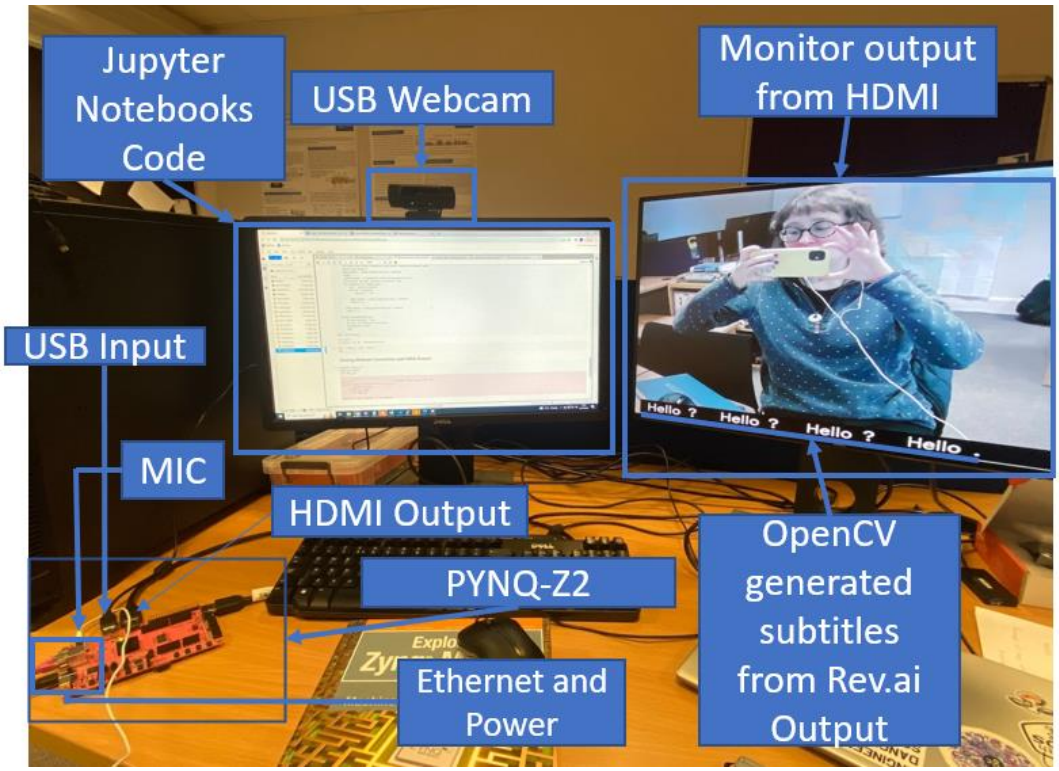

Figure 46 Layout of Running S2T system with USB input

An example video of the programme operating and transcribing speech in real time can be accessed through the QR code, Figure 47.

Figure 47 QR code to demonstrate S2T operating in live time

A webcam input was initially hard to obtain until it was noticed that the webcam would display frames for around one second before flashing, stalling then eventually

cutting out, requiring the PYNQ-Z2 device to be restarted. This was because the webcam required an external 12 V supply.

Webcam required an external 12 V supply; this is undesirable for our system design as we require low power operation. Not that 12 V is a large power to offer the board but just that having another input is undesirable for the board. Given in Figure 48 is the board when the webcam was connected.

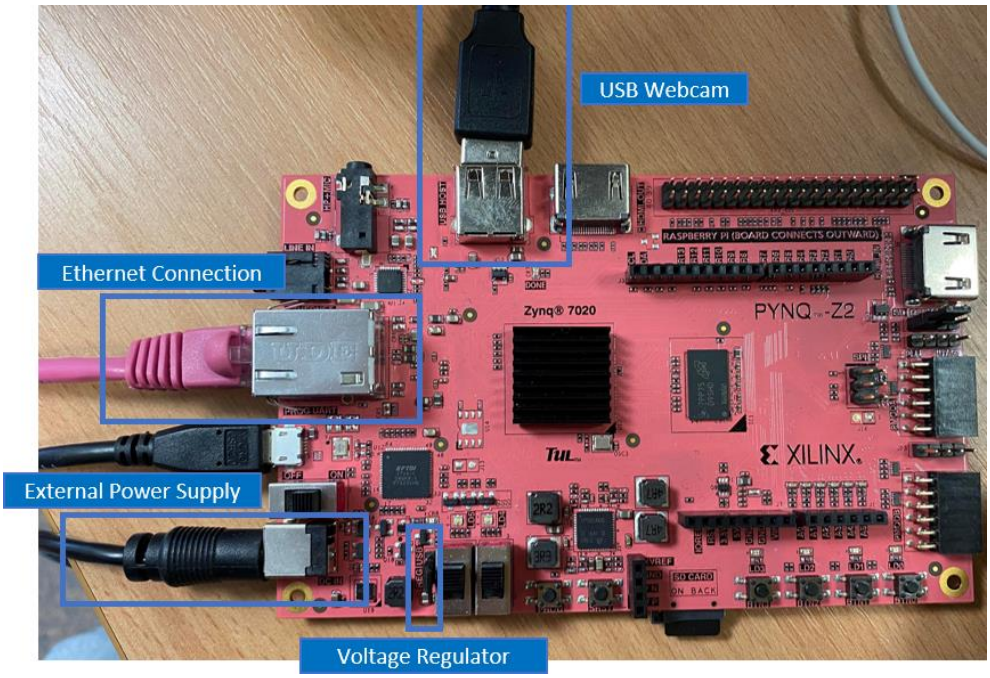

Figure 48 Utilised Ports when USB webcam is operating

Note that within the figure, the voltage regulator had to be used to not to fry the device. The main concern with having an external power supply is that there is still an ethernet USB connection to a laptop. The laptop should be able to power the device but when webcam is used, we require a connection to mains or an independent 12 V supply.

This is why it was looked into using HDMI as that would mean that the laptop was able to:

- Provide ethernet connection;
- Power the device;
- Provide an input signal.

Due to the base overlay for ZYNQ-7020 (PL for PYNQ-Z2) having HDMI routed through the programmable logic anyway, it seemed intuitive that HDMI was used in order for video processing to be hardware accelerated. There is also scope for future work as HDMI carries audio data also, meaning that it wouldn't be required to use a separate microphone, thus reducing the physical complexity of the design further, mentioned in section 6.1.

3.3. Hardware Acceleration of Video Processing

Hardware design was purely focused on the video processing component of the design, not the audio. That is, within the PS, the S2T functions were still being used to generate the false frame images containing, the subtitled output. Given in Figure 49 is an example faux image of a subtitled output.

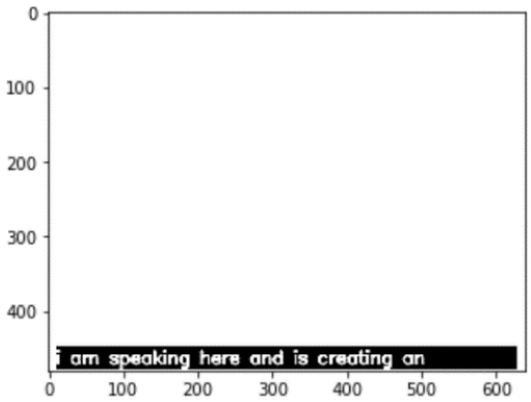

Figure 49 Example Faux Image

So, after the false image is created within the PS it was written into the PL and go into the video mixer block. Given in Figure 50 is a system overview diagram for the software-hardware design

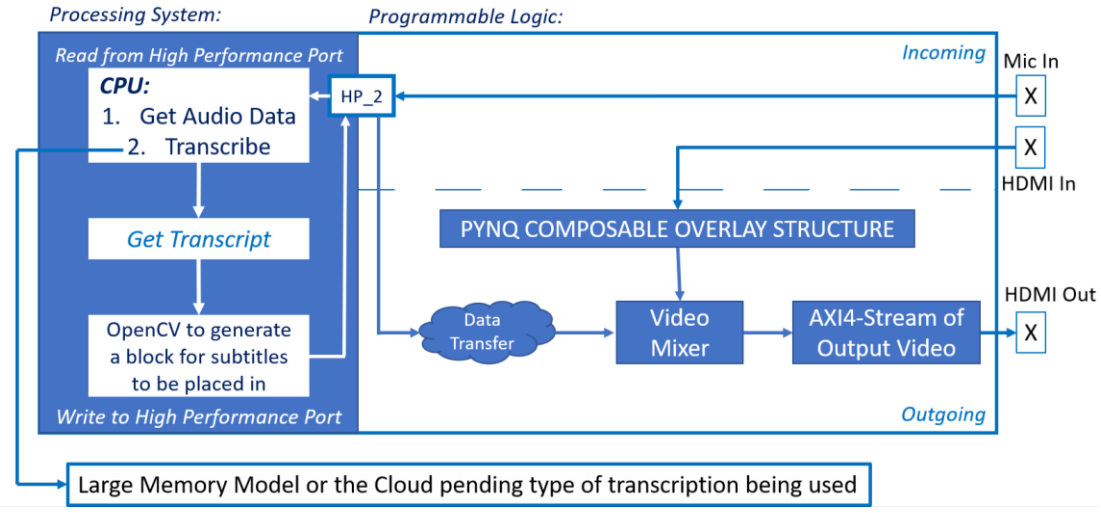

Figure 50 System Overview Diagram of Design

The figure shows a simplified projected design of the software-hardware overlap. The design process for hardware development followed stages of sanity checking everything, such that if anything went wrong, it was easier to identify the issue and attempt to sort and prevent it. From this, the video mixer design was composed of creating a design for a single layer to just stream the input frames directly to ensure operation was sound before adding a secondary layer which contained the subtitled overlay.

3.3.1. First Layer Construction

Given in Figure 51 is the block diagram for the original design. This was a single layered design taking the HDMI input and passing it through the logic to receive hopefully the same output being shown on the HDMI monitor.

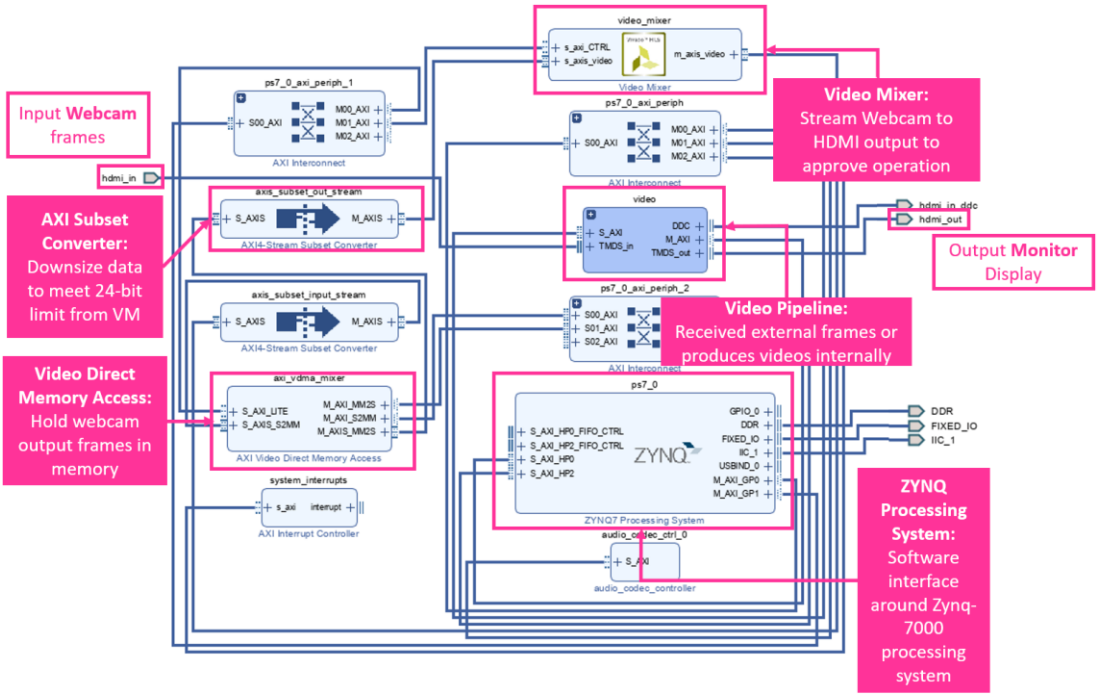

Figure 51 Initial design, annotated

The initial design's bitstream was then generated and uploaded to the Jupyter Environment to allow the processing system to utilise the design. For this to be performed, a class of the video mixer was created. Within a class, all programmable registers must be defined: such as control values, layer enabling defining transparency values, etc. A list of the programmable registers is given in Figure 52.

```
[ ]: video_mixer_regs = {
    'CTRL':
        {'address_offset': 0, 'size': 32, 'access': 'read-write', 'description': 'Control signals', 'fields': {
            'ap_start': {'bit_offset': 0, 'bit_width': 1, 'description': 'Control signals', 'access': 'read-write'},
            'ap_done': {'bit_offset': 1, 'bit_width': 1, 'description': 'Control signals', 'access': 'read-only'},
            'ap_idle': {'bit_offset': 2, 'bit_width': 1, 'description': 'Control signals', 'access': 'read-only'},
            'ap_ready': {'bit_offset': 3, 'bit_width': 1, 'description': 'Control signals', 'access': 'read-only'},
            'flush_pending': {'bit_offset': 5, 'bit_width': 1, 'description': 'Flush pending AXI transactions', 'access': 'read-write'},
            'flush_done': {'bit_offset': 6, 'bit_width': 1, 'description': 'Flush Done', 'access': 'read-only'},
            'auto_restart': {'bit_offset': 7, 'bit_width': 1, 'description': 'Control signals', 'access': 'read-write'}}
        },
    'GIER': {'address_offset': 4, 'size': 32, 'access': 'read-write', 'description': 'Global Interrupt Enable Register', 'fields': {
        'Enable': {'bit_offset': 0, 'bit_width': 1, 'description': 'Global Interrupt Enable Register', 'access': 'read-write'}}
    },
    'IP_IER': {'address_offset': 8, 'size': 32, 'access': 'read-write', 'description': 'IP Interrupt Enable Register', 'fields': {
        'ap_done': {'bit_offset': 0, 'bit_width': 1, 'description': 'IP Interrupt Enable Register', 'access': 'read-write'},
        'ap_ready': {'bit_offset': 1, 'bit_width': 1, 'description': 'IP Interrupt Enable Register', 'access': 'read-write'}}
    },
    'IP_ISR': {'address_offset': 12, 'size': 32, 'access': 'read-write', 'description': 'IP Interrupt Status Register', 'fields': {
        'ap_done': {'bit_offset': 0, 'bit_width': 1, 'description': 'IP Interrupt Status Register', 'access': 'read-only'},
        'ap_ready': {'bit_offset': 1, 'bit_width': 1, 'description': 'IP Interrupt Status Register', 'access': 'read-only'}}
    },
    'Width': {'address_offset': 16, 'size': 32, 'access': 'read-write', 'description': 'Active width of background'},
    'Height': {'address_offset': 24, 'size': 32, 'access': 'read-write', 'description': 'Active height of background'},
    'background_r_or_y': {'address_offset': 40, 'size': 32, 'access': 'read-write', 'description': 'Red or Y value of background color'},
    'background_u_or_g': {'address_offset': 48, 'size': 32, 'access': 'read-write', 'description': 'Green or U value of background color'},
    'background_g_or_u': {'address_offset': 56, 'size': 32, 'access': 'read-write', 'description': 'Blue or V value of background color'},
    'layer_enabled': {'address_offset': 64, 'size': 32, 'access': 'read-write', 'description': 'Layer enable', 'fields': {
        'master_layer': {'bit_offset': 0, 'bit_width': 1, 'description': 'Master layer is enabled/disabled', 'access': 'read-write'},
        'overlay_layer_1': {'bit_offset': 1, 'bit_width': 1, 'description': 'Overlay Layer 1 is enabled/disabled', 'access': 'read-write'},
        'logo_layer': {'bit_offset': 23, 'bit_width': 1, 'description': 'Logo layer is enabled/disabled', 'access': 'read-write'}
    }},

    'layer_1_alpha': {'address_offset': 512, 'size': 32, 'access': 'read-write', 'description': 'Alpha blending value for layer 1'},
    'layer_1_start_x': {'address_offset': 520, 'size': 32, 'access': 'read-write', 'description': 'X position of the top left corner of layer 1, relative to the background layer'},
    'layer_1_start_y': {'address_offset': 528, 'size': 32, 'access': 'read-write', 'description': 'Y position of the top left corner of layer 1, relative to the background layer'},
    'layer_1_width': {'address_offset': 536, 'size': 32, 'access': 'read-write', 'description': 'Active width (in pixels) of layer 1'},
    'layer_1_stride': {'address_offset': 544, 'size': 32, 'access': 'read-write', 'description': 'Active stride (in bytes) of layer 1'},
    'layer_1_height': {'address_offset': 552, 'size': 32, 'access': 'read-write', 'description': 'Active height  (in lines) of layer 1'},
    'layer_1_scale_factor': {'address_offset': 560, 'size': 32, 'access': 'read-write', 'description': 'Scale factor for layer 1'},
    'layer_1_plane_1_buffer': {'address_offset': 576, 'size': 32, 'access': 'read-write', 'description': 'Start address of plane 1 of frame buffer for layer 1'}
}
```

Figure 52 Video Mixer's relevant programmable registers

The registers were extracted from the user guide for the video mixer. [43] A class utilising these registers was then created. This involved setting the control and layer enable registers while defining the height and width of the input frame. Given in Figure 53 is the Video Mixer class definition.

Figure 53 Video Mixer Class Definition, annotated diagram

Figure 54 is the code required to utilise the class shown in Figure 53.

```
[5]: # 1920 x 1080 Monitor but that is larger than the
     ol.video_mixer.base_height = 1080
     ol.video_mixer.base_width = 1920

     ol.video_mixer.base_height = frame_height
     ol.video_mixer.base_width = frame_width

     ol.video_mixer.en_layer = 1  # if base layer alone

[6]: hdmi_in = ol.video.hdmi_in
     hdmi_out = ol.video.hdmi_out

[7]: hdmi_in.configure()
     hdmi_out.configure(hdmi_in.mode)

     hdmi_in.start()
     hdmi_out.start()

[7]: <contextlib._GeneratorContextManager at 0xa1beec88>

[8]: hdmi_in.tie(hdmi_out)

[9]: hdmi_out.close()
     hdmi_in.close()
```

Figure 54 Main body code for using Video Mixer Class

Once the class had been instantiated, code could be run. To test the operation, the HDMI input was taken from a laptop. From the laptop's point of view, it was extending its display to a second monitor, via PYNQ-Z2. From observation, it was apparent that the output moved in real-time; however, there were bars across the screen recognised as data loss. Given in Figure 55 is a comparison between the monitor output when the screen is normally extended verses when it is passed through the video mixer.

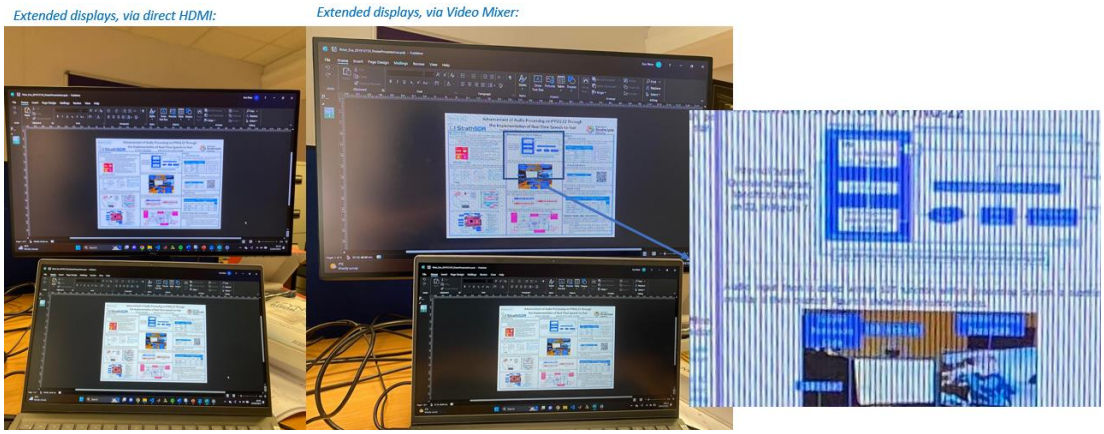

Figure 55 Demonstration between HDMI output and Video Mixer Output

From analysis of the monitor output, it seemed that it was dropping samples every 8 bits. The problem was assumed to be due to the AXI subset converter: it accepted an input signal of word length 32-bits then converted into an output of 24-bits, which entered the Video Mixer. There was found to be a secondary error, due to resolution, but that will be explained fully in the next section. The down sampling was accepted but slightly ignored as the system was operational in real-time. So, the next stage was started.

3.3.2. Enabling an Overlay Layer

The Video Mixer IP was the first change to the design, this involved changing its structure to accept a second layer. The settings used to accommodate this are shown in Figure 56.

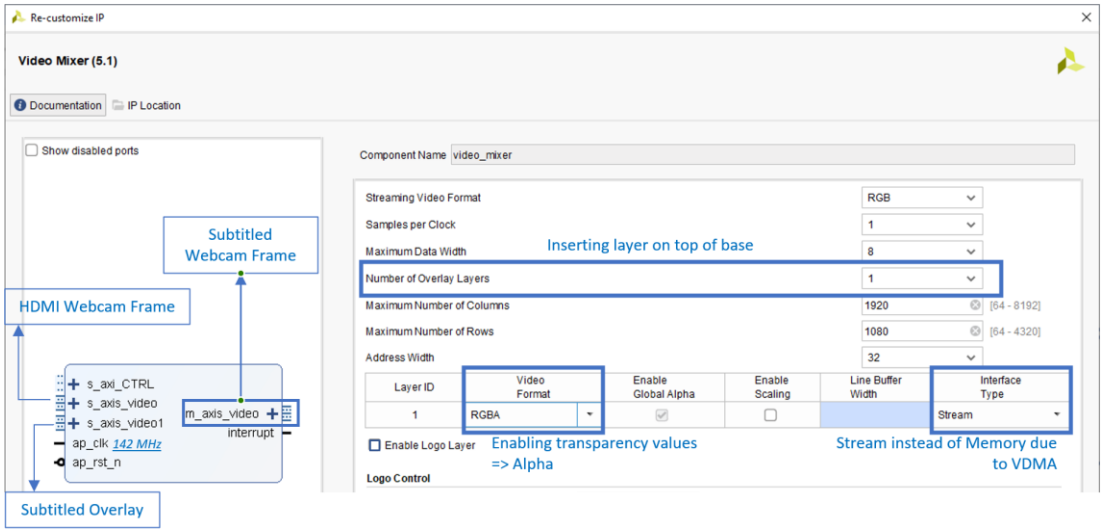

Figure 56 Video Mixer Settings to allow Overlay

RGBA (Red, Green, Blue, Alpha) was used as the overlay to “enable global alpha”, that is so that the layer could be defined to be transparent apart from the subtitled section. That is, the first three channels supply the colour data while the fourth gives a transparency value. To store the subtitled overlay and read the image from memory a VDMA was used. This also meant that the secondary port did not have to be

memory mapped hence fewer registers had to be enabled from this streaming port. Figure 57 holds the annotated updated diagram.

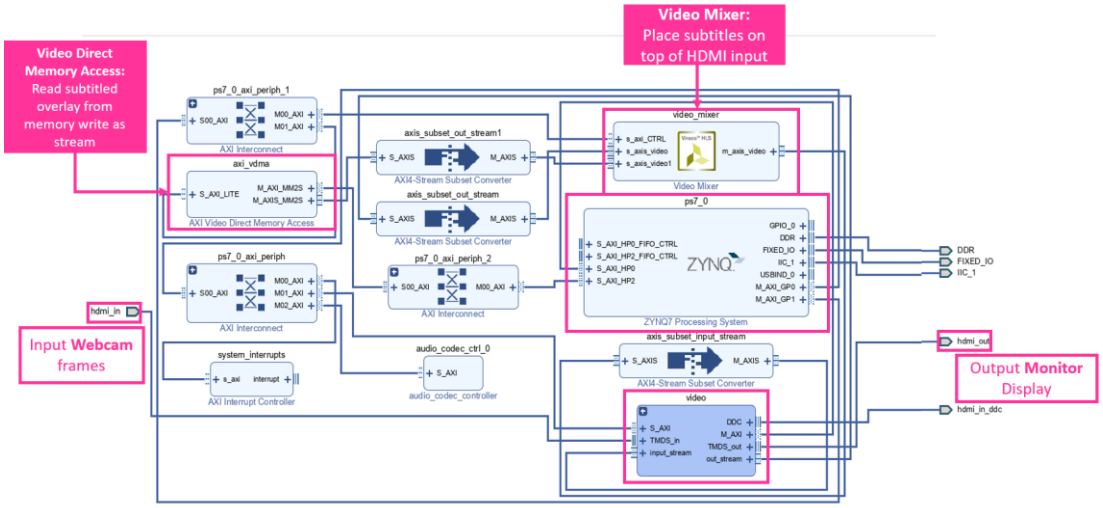

Figure 57 System with enabled 2nd layer

The code was updated as well, the class had to have various additional registers programmed, shown in Figure 58.

```
## 1st Overlay Properties:
    ## Alpha Value of Overlay:
@property
def first_alpha(self):
    return self.read(0x200)
@first_alpha.setter
def first_alpha(self, alpha_value):
    if (alpha_value < 0) or (alpha_value > 256):
        raise ValueError('Alpha Value Outside Conditions')
    self.write(0x200, alpha_value)
```

Defining Transparency

```
    ## Defining the Coordinates of X and Y Positions of the Overlay:
@property
def start_x(self):
    return self.read(0x208)
@start_x.setter
def start_x(self, x_value):
    if (x_value < 0) or (x_value > self._MAX_WIDTH):
        raise ValueError('x value outwith width constraints')
    if type(x_value) != int:
        raise TypeError('Bad Height Datatype!')
    self.write(0x208, x_value)
@property
def start_y(self):
    return self.read(0x210)
@start_y.setter
def start_y(self, y_value):
    if (y_value < 0) or (y_value > self._MAX_HEIGHT):
        raise ValueError('y value outwith height constraints')
    if type(y_value) != int:
        raise TypeError('Bad Height Datatype!')
    self.write(0x210, y_value)
```

Defining where top left corner of the overlay should be, relative to the base

```
    ## Defining width and height of overlaid image:
@property
def layer_width(self):
    return self.read(0x218)
@layer_width.setter
def layer_width(self, lw_value):
    if (lw_value < 0) or (lw_value > self._MAX_WIDTH):
        raise ValueError('Bad Width!')
    if type(lw_value) != int:
        raise TypeError('Bad Width Datatype!')
    self.write(0x218, lw_value)

@property
def layer_height(self):
    return self.read(0x228)
@layer_height.setter
def layer_height(self, lh_value):
    if (lh_value < 0) or (lh_value > self._MAX_WIDTH):
        raise ValueError('Bad Width!')
    if type(lh_value) != int:
        raise TypeError('Bad Width Datatype!')
    self.write(0x228, lh_value)
```

Defining height and width of the overlay

Figure 58 Additional Class Registers to Enable an Overlay

Then, extra lines within main body code also are needed; they are given in Figure 59. Also, note here that what was written to the overlay port was just the rectangle with the subtitles, alpha was still necessary to ensure that that was fully opaque.

Defining the Output Layer Operations:

```
[13]: ol.video_mixer.en_layer = layer_number

      ## Physical Shape of the Frame:
      ol.video_mixer.base_height = frame_height
      ol.video_mixer.base_width = frame_width

      ol.video_mixer.start_x = topleft_x
      ol.video_mixer.start_y = topleft_y

      ol.video_mixer.layer_height = overlay_height
      ol.video_mixer.layer_width = overlay_width

      ol.video_mixer.first_alpha = alpha_val
```

Defining the Input and Output HDMI Blocks:

```
[14]: hdmi_in = ol.video.hdmi_in
      hdmi_out = ol.video.hdmi_out
```

```
[15]: hdmi_in.configure()
      hdmi_out.configure(hdmi_in.mode)

      hdmi_in.start()
      hdmi_out.start()
```

Figure 59 Internal Code Operation for additional Layer

However, when this configuration was given an input, no output was found, and the device had to be restarted after every attempt. That was because the VDMA was being read from before there was the ability to write to it. Hence, the VDMA would have to be tied to itself and the HDMI input and outputs therefore requiring it to have read and write ports. Given in Figure 60 is a block diagram showing the desired configuration.

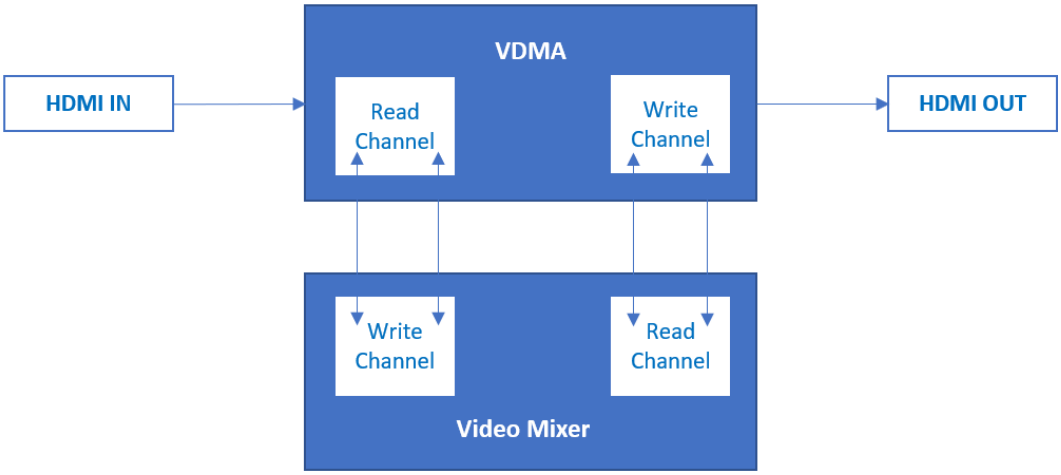

Figure 60 Method to Stream Memory Efficiently

Hence, there was the requirement to update the VDMA. Therefore, VDMA which read the overlaid files from memory and the VDMA which accepted the streaming port from the HDMI input both needed these alterations. New configuration given in Figure 61.

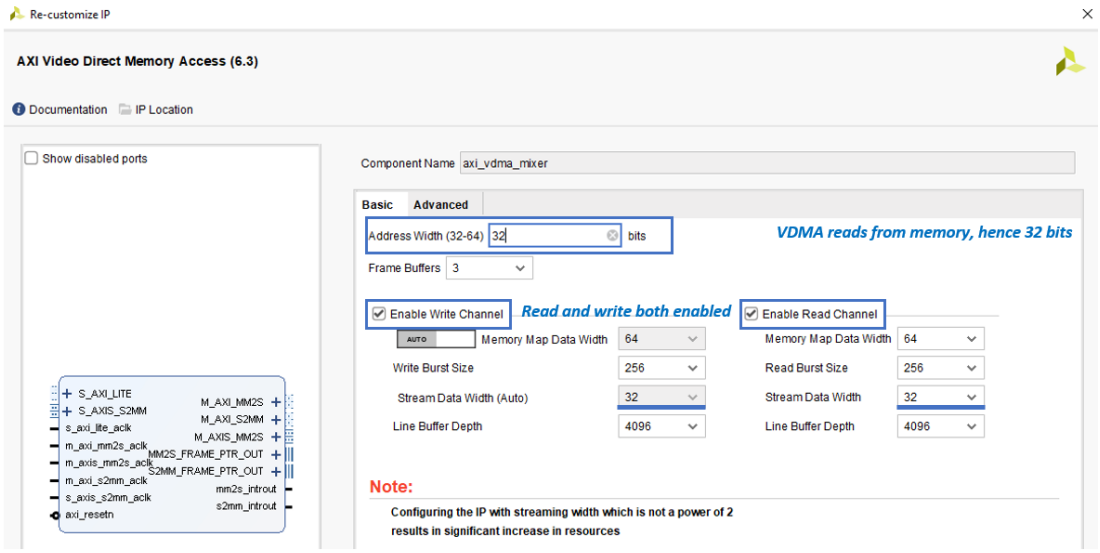

Figure 61 VDMA Read and Write Ports

The design had to be recreated and it was resolved that only one layer should be used for the redevelopment of the design.

3.3.3. Reconfiguration of Design

Given in Figure 62 is the design of the system, with one layer and the correct streaming set up.

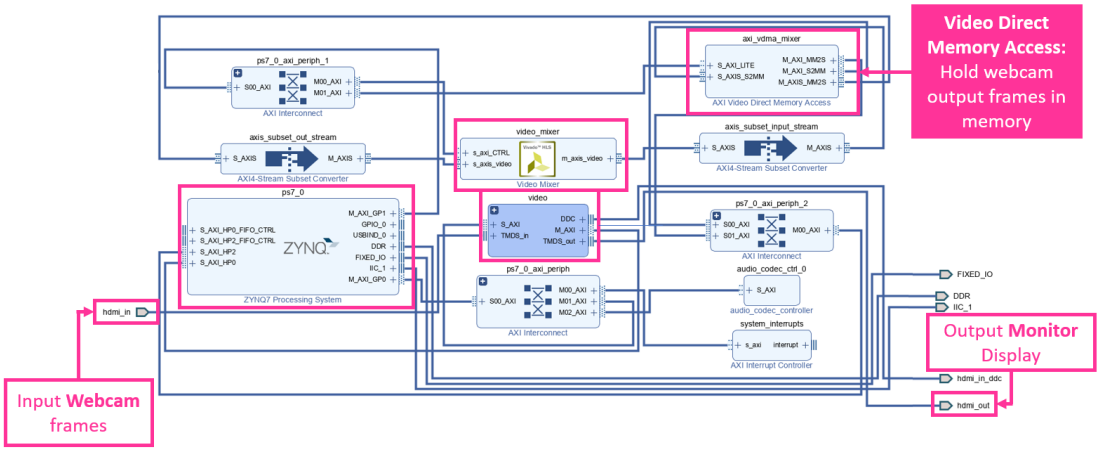

Figure 62 Improved design for one layer operation

Note, the VDMA which accepts the HDMI input is stored within the video pipeline so not visible within Vivado block diagrams.

As a single layer was being used again, the code used to provide an output for the mono layer result was repeated; with the addition of the VDMA initialisations. The output was the same as the output from the first configuration. Hence a secondary layer was inserted. Block diagram is shown in Figure 63.

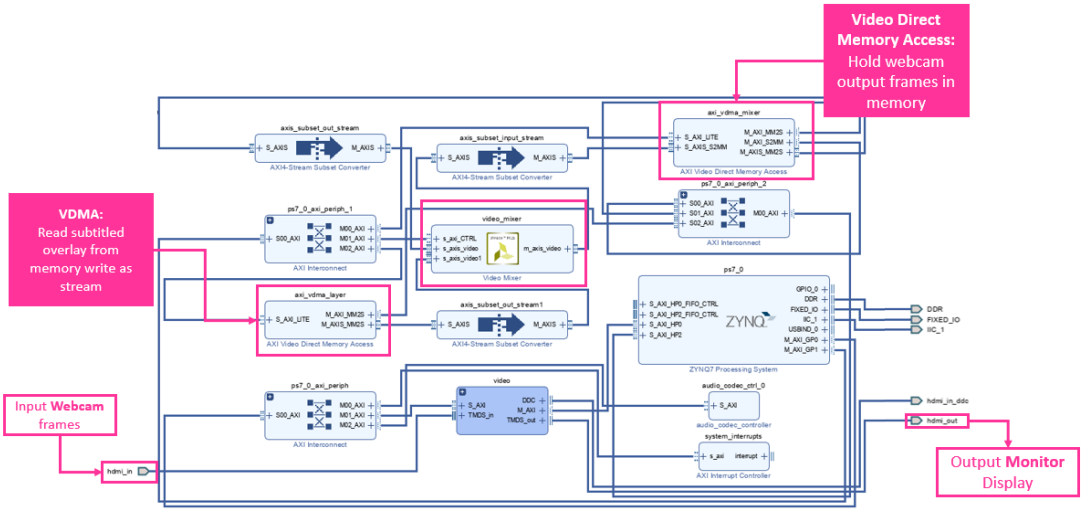

Figure 63 Insertion of a second layer

This code initially ran, however, after a short period of time passed, the output began to flash before immediately cutting out. It was found to be due to resolution. The device could not accept 1080 by 1920 pixels at 60 fps. This was This was because of the sheer memory required to store this as there was more data in one burst than the device could handle. Hence the down sampling from the first layer establishment. Within the Discussion, this is spoken about, and a justification is provided, see section 5.1.

Unfortunately, this realisation of error came too late within the project scope so it could not be resolved within adequate time to present results within report. Therefore, the hardware design was successful, even although the PYNQ-Z2 implementation was unable to be performed. All code and hardware designed were saved and maintained for it to be reproduced for a different target device. Within the Discussion and Future Work section (5 and 6, respectively) of the report, solutions are presented and justified as to which was used.

4. Testing of Design

4.1. Accuracy of Rev.ai

4.1.1. Transcribing English

To qualify the accuracy of S2T Rev.ai's Software Development Kit (SDK), a passage was read individually by a group of people. Of course, the word error rate (WER) for Rev.ai has not been improved through the development of this project; however, it was deemed important to show it is dependable. This is especially useful if the contents of the project will be used within education or another widening access features, it is important to have justified its accuracy. WER formula is given in Equation 1.

Equation 1 WER formula

$$WER = \frac{S + I + D}{\#\ of\ Words\ Spoken} \quad (1)$$

Where:

- S is the number of incorrectly substituted words;
- I is the number of incorrect insertions: e.g., "SAT" to "essay tea";
- D is the number of incorrect deletions.

Table 9 holds key information about the speakers which will have an influence on how they speak.

Table 9 Key vocal features of test group

Speaker ID:	Gender:	Home Location:	Native English Speaker:
A	Female	West Coast of Scotland	Yes
B	Male	West Coast of Scotland	Yes
C	Female	English Midlands	Yes
D	Female	Luxembourg's border with Germany	No
E	Male	Donegal County of Ireland	Yes
F	Male	A Coruña in Spain	No
G	Male	Perugia in Italy	No

The speakers were chosen as they each have a unique trait to their voice; that is, they are all from different places and are of different genders. Also, speakers A and B were chosen to observe whether the S2T SDK had a different accuracy based upon the speaker's gender. Otherwise, it had to be determined whether the SDK had a bias

towards certain accents, hence why Scottish, English and Irish accents were chosen to be compared with central European accents. The pace of speech was also evaluated as native speakers will have a different timbre and intonation of words than people whose second language is English.

The test passage was two stanzas of William Wordsworth's "I Wandered Lonely as a Cloud", its text is given in Figure 64, below.

I wandered lonely as a cloud
That floats on high o'er vales and hills,
When all at once I saw a crowd,
A host, of golden daffodils;
Beside the lake, beneath the trees,
Fluttering and dancing in the breeze.

Continuous as the stars that shine
And twinkle on the milky way,
They stretched in never-ending line
Along the margin of a bay:
Ten thousand saw I at a glance,
Tossing their heads in sprightly dance

Figure 64 Test passage for speakers

In turn, each speaker read out the passage: they were also provided with the passage prior to recording so were able to practise fully. The speaker was recorded, and two tests were performed. While they were speaking, the SDK was called to transcribe the audio, in real time. Then, the speaker's recorded output was transcribed using the local file transcription. The results are held within Table 10.

Table 10 Accuracy results from Rev.ai test Speakers

Speaker	Live Transcription				Local File Transcription				Spoken Time (s)
	Number of Incorrectly Interpreted Words			WER	Number of Incorrectly Interpreted Words			WER	
	Insertion	*Deletion*	*Substitution*		*Insertion*	*Deletion*	*Substitution*		
A	2	2	36	0.579	3	0	26	0.42	31
B	11	0	17	0.4	0	0	7	0.09	32
C	2	1	19	0.3	0	0	9	0.12	36
D	0	1	24	0.39	0	2	9	0.14	33
E	0	13	28	0.79	0	0	12	0.16	29
F	1	4	23	0.37	1	0	8	0.12	39
G					0	1	10	0.13	41

Speaker G could only send a pre-recorded video therefore was not able to have a streaming WER analysis performed.

The local file transcription was more accurate than the live audio for every speaker. This is because the live and local transcription methods vary for their S2T detection: live transcription contacts a cloud-based server to do so whereas the local transcription uses its large offline language model for comparison of results. Each speaker had interesting results from the tests, these are expanded on below:

- Speaker B has a notable drag to their voice, that is they have a tendency to speak quickly for the majority of a sentence then pause before starting the next sentence. This was a trait also shared by speaker E. However, the local transcription service offered a very low word error rate, implying that the speakers were clear, their local accents were unable to be picked up in real time.
- Speakers with a "neutral" accent were more likely to be interpreted correctly rather than strong Scottish and Irish accents. This is purely because of the tempo of speech and certain qualities such as sharp vowels and rolling "R" sounds.
- The speakers that spoke the slowest were the best interpreted, in live time.

Given in Table 11 are various incorrectly interpreted words and the speaker:

Table 11 Commonly Misinterpreted Words in Passage

Speaker	**Word Read**	**Word Interpreted**
B	"Milky Way"	"Mull kiwi"
A, B and D	"Breeze"	"Priest"
E and F	"Hills"	"Health"
A and B	"Lake"	"Leak"
C	"Fluttering"	"Flushing"

Hence, there are varying degrees of misinterpretation. Also, from the table it is easy to see the link between male and female speakers as they have dissimilar errors picked up, this is also noticeable as the two Scottish speakers had the same incorrect word twice: "Breeze" and "lake".

4.1.2. Translation Feature:

As speakers C, F and G were the only non-native English speakers in the test group, they were also used to qualify the accuracy of the language detection. In this instance, it had to be checked that the service could detect the European languages.

From this, several different nursery rhymes were used; German rhyme “Hoppe Hoppe Reiter”, which loosely translates to “Bumpety bump rider”, Spanish “Un Elefante” and “Stella Stellina” for an Italian contribution. Given in Table 12 are the lines of the nursery rhymes which were read and their corresponding English translation.

Table 12 Language Test Cases and Their English Translation

Speaker’s First Language	German	Spanish	Italian
Test Passage:	*Hoppe hoppe Reiter* *wenn er fällt, dann schreit er,* *fällt er in den Teich,* *find't ihn keiner gleich.* *Hoppe hoppe Reiter* *wenn er fällt, dann schreit er,* *fällt er in den Graben,* *fressen ihn die Raben*	*Un elefante no es elegante* *no usa sombrero corbata ni guantes* *La trompa muy larga, la panza gigante,* *las patas muy gruesasy orejas muy grandes* *Pero eso, ya sabes, no es lo importante* *pues ya es especialser un elefante*	*Stella stellina* *La notte s’avvicina* *La fiamma traballa* *La mucca è nella stalla* *La mucca e il vitello* *La pecora e l’agnello* *La chioccia e il pulcino* *Ognuno ha il suo bambino* *Ognuno ha la sua mamma* *E tutti fan la nanna*
English Translation:	*Bumpety bump, rider,* *if he falls, then he cries out* *should he fall into the pond,* *no one will find him soon.* *Bumpety bump, rider...* *should he fall into the ditch,* *then the ravens will eat him.*	*An elephant* *Is not elegant* *he does not wear a hat* *tie or gloves* *His trunk is too long,* *His belly is giant,* *His legs are very thick* *and ears very large* *But that, you know,* *is not important* *since it’s already special* *to be an elephant*	*Star Little Star* *The night is coming* *The flame is trembling* *The cow is in the stable* *The cow and the calf* *The sheep and the lamb* *The mother hen and the chick* *Each has their little one* *Each has their mom* *And everyone sleeps*

Simple nursery rhymes were used as the language detection feature is a secondary component of Rev.ai SDK. That is, the language service was initially designed for full English detection and comprehension and now it is being expanded for other languages. Currently, Rev.ai only offers a language detection feature rather than a full translation.

Rev.ai’s language detection service was used in conjunction with Google’s Speech Recognition API for translation. That is, Rev.ai would detect the language and Google would translate and transcribe based on that language.

The results from the test are given in Table 13.

Table 13 Results from Rev.ai's language detection service and Google's Speech Recognition Package

Speaker	Language Spoken	Language Detected	Number of Incorrectly Interpreted Words	Word Error Rate for Extract	Speaking Time (s)
D	German	German	5	0.11	20
F	Spanish	Spanish	5	0.09	24
G	Italian	Italian	6	0.13	19

Results from this test are not concerned with the WER, as it was not produced by Rev.ai. So, the focus of this test was that all the correct languages were detected, which they were. The WER for the local file translation was lower than for the English file, there are a few reasons for this. A different S2T API was used for the transcription and fewer complex words were used within the European poems. Nursery rhymes were used due to its basic language and easily transcribable nature, speakers were also asked to perform their nursery rhyme more deliberately.

It was proven earlier in the report (section 2.1.3) that Rev.ai was a more reliable package along with the fact that it has more advancement within the realms of offline detection, therefore will be used in future.

4.2. Comparison of Designs

Within a software design, all packages and function files have to be loaded in before operation can commence, the same is true for a hardware design. As well as function packages, the hardware downloads bitstreams and layout files as well as defining the streaming ports used for the design. Supplementary to the downloading of bitstream files, bitstreams have to be designed from a hardware design and take a while to generate to become compatible. However, once this is completed, the actual programme can run with very few limitations. Unlike software, where running in parallel is hard to be produced due to software having run time constraints.

Hardware accelerated components of the design were focused upon overlaying images and providing a real-time webcam output. Within the PYNQ-Z2, all the S2T functions were run within the processing system, therefore still software.

4.2.1. Frame Rate

Frames per second (fps) of the outputted webcam image was compared for software and hardware components. The fps rate for software systems was predicted to be less than the fps for the hardware design as it was designed for components to run in parallel.

Given in Figure 65 is a comparison of the frame rate of two hardware accelerated designs (FPGA and GPU) compared to a purely software framework for image identification, taken from a conference paper.

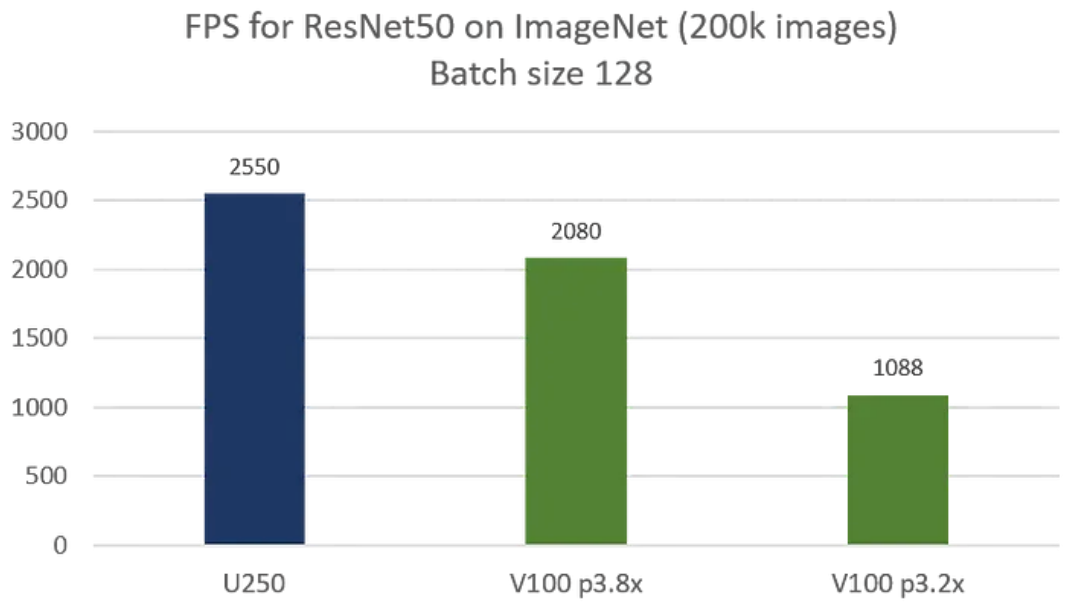

Figure 65 FPS for ResNet50 inference on ImageNet (200k images) with batch size 128 [54]

The human eye can detect between 30 and 60 fps. They cannot detect frame rates faster than 60 fps: resolution is not clearer above this value. However, buffers and

frame stills are noticeable below 30 fps. Given in Table 14 is a comparison between the hardware and software designs.

Table 14 fps for different design configurations

Design Configuration	fps
Python Software Alone	23.54
PYNQ Framework	25.1
PYNQ-Z2 Hardware Implementation	60

PYNQ-Z2 can pass webcam directly through the design, even when software is being run. This is because the PL does not have to wait for an input from the processing system before it can run. Unlike within the PYNQ framework as it waits for the result of secondary function output, i.e., S2T output. The same is true for Python software alone, however, software requires time to "catch up with itself" and reset memory therefore there is a slightly larger lag there due to its throttling values.

4.2.2. Rev.ai Subtitle Output

Previously mentioned that Rev.ai is ran in PS for both designs. Theoretically, they should run for the same volume of time. So, what was compared was the time taken to run whole script (call bitstreams, install libraries etc) to the time taken to run the local transcription alone. Both configurations were given the same three second ".WAV" file to transcribe, locally. With the same code to transcribe locally. Of course, PYNQ Framework had additional set up variables of calling bitstream and defining audio boundaries. Table 15 holds the result of this run-time comparison.

Table 15 Comparison of Script Run Time and Transcription Run Time

Design Configuration	Time to Run Full Local File Transcription Script (s)	Transcription Time (s)
Python Software Alone	45	17
PYNQ Framework	56	15

As predicted, PYNQ has a larger required set up time but is more efficient when up and running than software alone. What can also be noted is that there is a trade-off between transcription time and accuracy, as the local file is proven to be more accurate just extremely slow to run compared to the real-time subtitled which are essentially live.

4.3. Hardware Statistics:

Given in Table 16 is the resource table for the final design, shown in Figure 63. All FPGAs have a value of PL, routing, input/output, and memory resources available for the compiler to use to implement the HDL code.

A configurable logic block (Look-Up Tables, Flip-Flops, Multiplexers, etc) is a logic resource on an FPGA which are linked through routing resources to implement complex logic functions, memory functions and synchronise code on FPGA. Block random-access memory embedded throughout FPGA for data storage. Input and output ports are the target physical structures that allow connection between design system and the board. They translate analogue signals to a digital value to allow procedures within board. [55]

Table 16 Resource Table of Finalised Design

Resource Type	**Resource**		**Utilisation**	**Available**	**Utilisation %**
	Name	**Acronym**			
Programmable Logic	Look-up Table	LUT	19341	53200	*36.355*
	Digital Signal Processing	DSP	36	2200	*16.364*
	Flip Flips	FF	30743	106400	*28.893*
Memory Resources	Block RAM	BRAM	44.5	140	*31.786*
	LUT RAM	LUTRAM	1028	17400	*5.908*
I/O	Input Output	IO	29	125	*23.200*
Clock Routing	Global Buffer	BUFG	5	32	*15.625*
	Mixed mode clock management	MMCM	3	4	*75*

A large proportion of the resources were used as the design carries video data which is essentially multiple arrays of pixels contained within memory as frame buffers. This also makes sense as previously mentioned in Hardware Design (Section 3.3) that the board ran out of memory when data was inputted from HDMI.

The clock mmcm is essentially full due to three clocks required within the design: 100 and 200 MHz for PS and PL management and 142 MHz clock for HD Video Default, 1080 pixels at 60 fps.

5. Discussion

5.1. Hardware Constraints of PYNQ-Z2

Within PYNQ-Z2, there is 256 kB of internal memory used within a Jupyter Environment, there is 512 MB of external memory [35] which is insufficient for video data. Within the design, there are two Video Direct Memory Access (VDMA) blocks: one within the video pipeline and a second being used as storage for the Video Mixer.

For a resolution of 1920 x 1080 pixels, there are four columns due to RGBA input data being used. Therefore, in one frame there are 8.2944 Mbytes which corresponds to 66.3552 MBits. However, a VDMA requires four frames, and within design there are two VDMAs; requires 530.8 Mbits for initialisation alone.

Very quickly PYNQ-Z2 is inefficient for operation; although there is still memory available in the PL, there is no *contiguous* memory, required to establish VDMA. Contiguous memory is memory beside each other: processes are run on consecutive blocks of memory; memory is requested by the process (before execution) and this size is compared to the amount of contiguous memory available to do the following operation. If memory is available, it is allocated to the process and it begins; else, it is added to a queue until sufficient contiguous memory is free. [56]

Decreasing the resolution was not applicable at present as within the design, the HDMI input came from a laptop. When laptop's resolution was decreased to the lowest available it would only be decreased internally, and when read into device it was still read in as 1920 x 1080 pixels. Resolution would have to be manually reduced within the PS then read back into PL, which adds complexity and still caused the memory breakdown error.

Therefore, even although PYNQ-Z2 has been utilised within the realms of video processing, now evident that this has been within very simple systems requires only one small input.

5.2. Review of Project Aims

Each aim was used within the design process of the project; however, as the aims were created at the start of the project (when the concepts of PYNQ and ZYNQ devices were new) they have not all been utilised in their expected way. This is not true for every aim, as it was with learning about ZYNQ architecture and video pipeline it was understood that a composable overlay could not be used exactly within design or that a new IP was not required to be created. It will be with increasing development time that more and more features will be added to the S2T device, as the focus of the project was communication, it was deemed that translation was the most important factor from the list.

Each “uncompleted” aim will be spoken about further in the following section. Note that, the aim was not uncompleted, they were still understood and used, just that its intended implementation was not applicable within design.

5.2.1. Composable Overlay Design Implementation

Within the background theory of the report (section 2) composable overlays are defined and explained as well as a walk-through of the labs provided by PYNQ. At the initial definition of the project, it was not known that composable overlays primarily deal with the implementation of filters, which was not relevant for the project. It was understood initially that the composable overlay was a structure which video data can be inputted into design. This is actually how composable structure was used within design; PYNQ offer a tutorial to design a composable overlay and the structure of the overlay for inputting filter blocks is given in Figure 66.

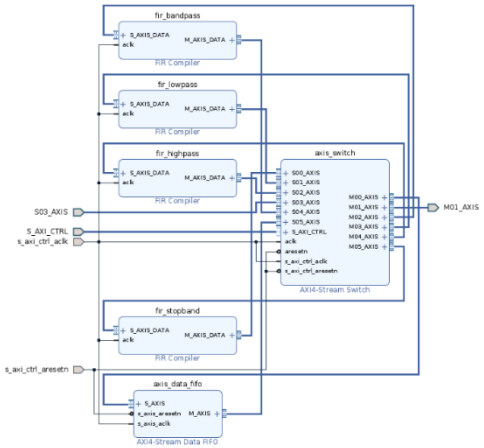

Figure 66 Suggested PYNQ Composable Pipeline Structure

The structure shown on figure was used within design. Instead of the filter function blocks, the video mixer was placed in the intermediate stage. As shown in Figure 67.

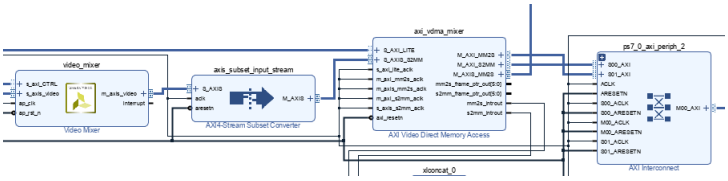

Figure 67 Composable Structure to Design

The design is not exactly like the suggested composable pipeline structure, as an AXI-Interconnect was used instead of an AXI-Stream Switch as the defined port was already AXI4-Stream.

The project is not concerned with filters, so it does not offer the typical composable structure, however as it provides image and video alteration it can be argued that it could be added as another component of the composable overlay. Thus, could be added within the composable overlay GitHub as open sourced for all to use.

5.2.2. IP Core Generation

On inspection of the video pipeline and composable overlay structure, as well as the issues faced with getting audio onto board, it was decided that (for the immediate design of project to get working prototype) audio was inputted separately to webcam data. This meant that there was no requirement to generate an IP for separating HDMI input into video frames and audio stream, as it was suggested at the interim stage.

In future, in order to make design more compact, it will be prevalent to only have one source. As indicated on Figure 1, an HDMI webcam which has a microphone attached would be the perfect solution. Initially, this was the plan as a USB webcam with a microphone was sourced. Therefore, audio and visual data stemmed from same port. However, it was later discovered that USB port cannot carry audio and visual data.

This extension will not require a long time as it was learnt within the project about using Simulink and Vivado through HDL Coder and System Generator to generate IP blocks. Labs from the 5th year class of Embedded System design were undertaken at the start of semester two so the skills are there to generate this, when there is sufficient time to do so.

5.2.3. Additional Features to Design

At the Statement of Intent, when the aims were to be defined, the additional features of the subtitled design included making the subtitled output into a closed caption service. Closed captions are subtitles which also mention the audio cues which are important to the situational environment. Hence, adding in features informing the viewer of who is speaking, subtitling sounds like doors opening, knocking, something falling over etc. Speaker identification would require frequency filters to try and separate the two speakers, this could be implemented, using the composable overlay within the PL. Given in Figure 68 are two different speakers, both saying the same word.

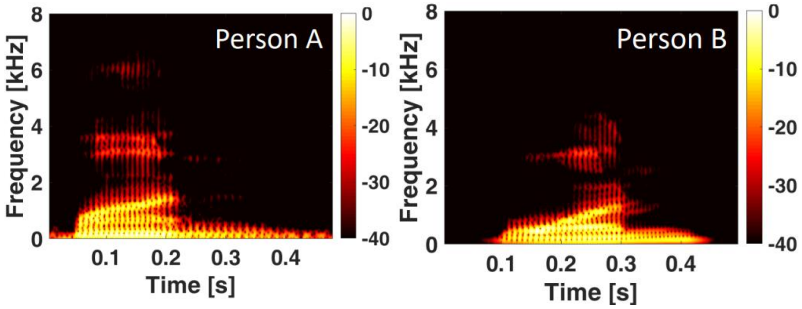

Figure 68 Comparison of speaker's oration [57]

Hence, two speakers could be easily separated, once it is apparent as to what divides them. This would require creating a frequency filtering device and using a machine learning algorithm to identify what the differences mean, instantly.

If background noise had to be added in, a secondary language model containing the sounds would have to be used alongside the S2T model. This implies that either the two models would have to be run in parallel or merged.

Another feature mentioned was image identification on the screen, this would be very useful from an educational standpoint: learning shapes and learning what physical objects are used for. It would also exploit the composable overlay structure and require an edge and feature detection components.

The translation feature was created and used. Rev.ai does not offer a direct translation feature but instead a language identification service. This was used in parallel with Google's S2T API in order to translate foreign languages into English. This was chosen as the most valuable as it not only allows deaf people to understand non-English speakers but also English speakers to understand non-English as well. This was prioritised as communication was the main focus within this project

6. Future Work

Future work for the project consists of developing the design to resemble the projected design from the introduction (diagram in Figure 1) more closely while also improving the quality of the outputted quantity.

6.1. Audio Input to Board

The audio being separate from the video stream is non-ideal due to this requiring more I/O ports within block design hence preventing the streamlining of ports. In future, audio should be read into the device via HDMI and routed through PL to allow for a more compact design.

6.2. Use of a Larger Device

As mentioned within hardware design (section 3.3) the design was operational and would provide a good output if PYNQ-Z2 had the available memory. Due to this, there were two options, rewrite design taking webcam input as USB and having the audio stream still enter through the mic jack of the board or, use a larger device which has the available memory to produce such output.

The reason why changing the device is more desirable than altering the design is because the USB input will not increase the board's capacity and when additional video frames are read in, more memory will be taken up by this hence the problem may still occur. Also, as USB cannot carry audio data, a microphone jack would still have to be used to obtain audio data. This is undesirable as very few devices have USB port, audio input and an HDMI output, so it would mean that the design would be specific to PYNQ-Z2, not very widespread and reusable.

The devices which are available to use within the Strath SDR lab are two ZYNQ MPSoC; this is important as ZYNQ MPSoC have larger available memory than ZYNQ-7000 series and an enhanced processing system. There is also a faster FPGA logic fabric with DSPs of wider word length therefore extended abilities. [27] A comparison of the two devices against PYNQ-Z2 is given in Table 17.

Table 17 Comparison of available MPSoC boards

Device	ZYNQ System	RAM	Peripherals	Power (V)
PYNQ-Z2 [35]	ZYNQ-7020	256 kB	• 2x HDMI 2.0 video input and output (3x GTH); • 1x USB 3.0; • 2x audio jacks	7
UltraScale96 [58]	ZYNQ MPSoC	2 GB	• 1x USB 3.0 Type Micro-B; • 2x USB 3.0; • 1x USB 2.0.	8
ZCU 104 [59]	ZYNQ MPSoC	38MB	• HDMI 2.0 video input and output (3x GTH); • DisplayPort (2x GTR).	12

The UltraScale96 has no HDMI connections, therefore could not be used in the final design; however, as it has the most memory, it would be able to perform the current design with some alterations. Alterations include that the design would have to be rewritten for USB inputs rather than HDMI inputs. As there is no microphone jack or HDMI there would be no way to carry audio data. Therefore, S2T would have to be performed completely off-board. This would have limited applications and would require an external processor to be run in parallel with the board, because the internal processor would not have the live inputs required. If not with a processor running in parallel, PYNQ Framework could be utilised for only transcribing local audio files, not fitting the criteria of real-time transcription.

Therefore, the targeted board would have to be ZCU 104. Board layout diagram is given in Figure 69.

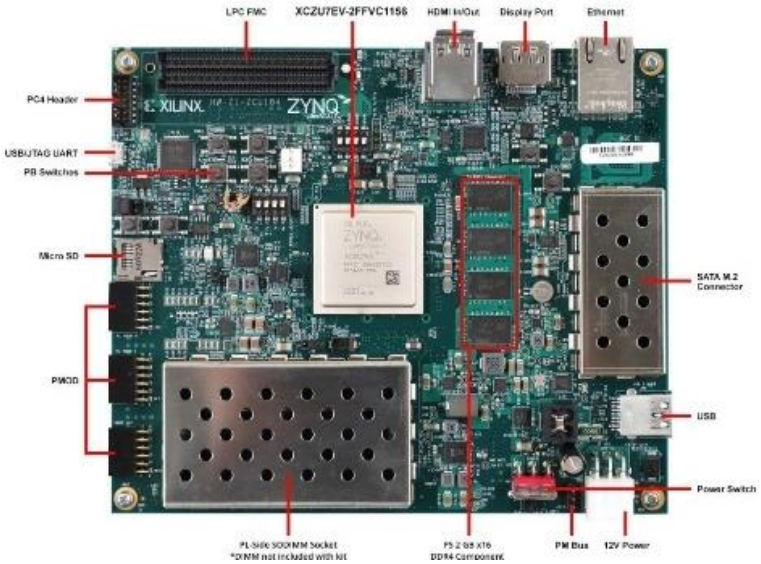

Figure 69 Board Layout of ZCU 104 board [59]

There will be necessary alterations to the design: only one HDMI port which will have to be the input and carry audio and visual data and the output will have to be written to a display port instead of an HDMI port. It also requires a larger voltage so the 12 V battery would only last over 5 days.

Also, as the audio is being routed directly onto the device, acceleration of S2T can be done within the PL, spoken about in following section. Hence, the evolution to MPSoC was the correct choice for the future operation of the design.

6.3. Implement S2T Function on Programmable Logic

One method to make the S2T function offline would be to implement the S2T algorithm onto the board, within the PL with frequent calls to the PS.

FPGAs can perform neural network operations at lower power and faster than GPUs. However, there is a lack of framework support within traditional FPGA structures meaning that may data scientists create a system of GPU and CPU. [60] So, the software-hardware implementation can be designed to be targeted on a ZYNQ MPSoC using PYNQ framework.

Through the generation of multiple IP blocks to perform statistical analysis comprising a Neural Network and the information being held within the PS, a design could be created to utilise both features adequately. An updated system overview diagram is given in Figure 70.

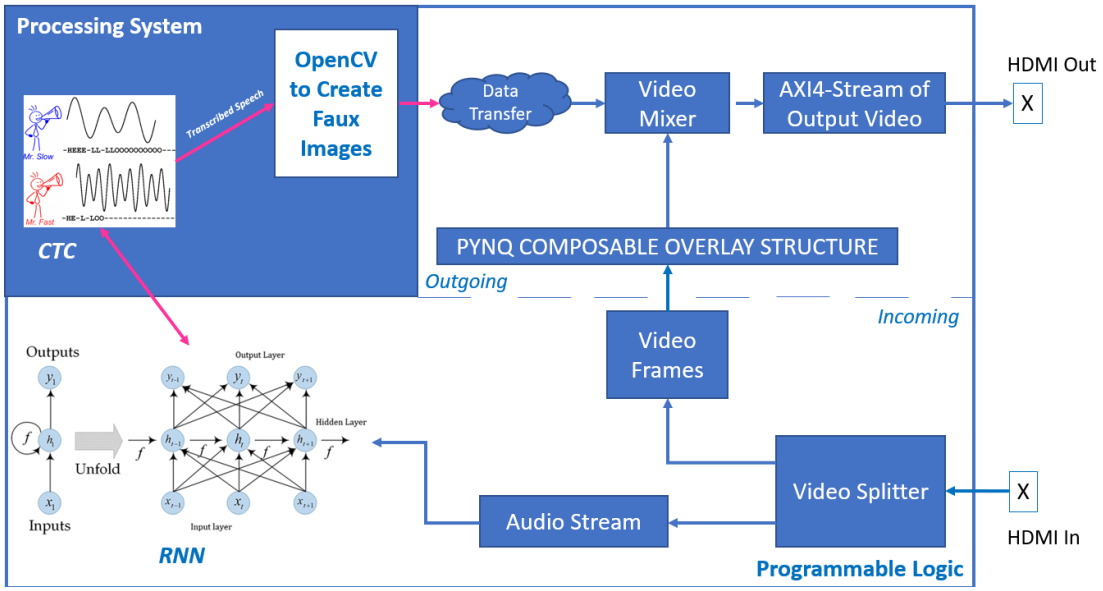

Figure 70 Implementation of S2T within PL

Hence, as can be observed, a large section of the programmable logic concerned with video processing remains as was, with the addition of an audio extraction IP (Video Splitter) which would be self-created. For the audio processing, a very high-level overview is given of how a machine learning algorithm and classifier pair would be instantiated within design. Note the double headed arrow between the RNN and CTC as the classified outputs are fed back into the Neural Network.

6.4. Downsizing of Physical Size of Project

Ideally, the project will be of portable size. That is, it must be lightweight, able to fit in rucksack and usable outside. As it will be moved around, the unit must be durable as well. This means that there are some requirements:

- Waterproof casing for the FPGA and power source, and their cables;
- Monitor with a display port and of similar size to FPGA: (179.1 x 149.8) mm [59];
- Small webcam with integrated microphone facing in the same direction as the monitor.

Also, within the design, the text outputs from all conversations would be stored within the Jupyter Notebook Framework and written to the SD card. Therefore, if a conversation would like to be looked at again, the user could plug the device into their laptop and download text file for future inspection.

7. Conclusions

Learned within project how to design an embedded system using PYNQ PS and ZYNQ PL and understand the operation of speech-to-text algorithms. An efficiently operating design was produced however hardware limitations prevented its utilisation; purely for the PYNQ-Z2 device. With the hardware design being reusable, it can be transferred and reproduced on a different PYNQ-supported device, with only slight alterations accounting for differing peripherals.

It was qualified that Rev.ai was a good choice of S2T SDK as it provides on and offline broadcasting. While the real-time broadcasting was online, which is not desirable for the final design, it provided nearly live subtitling and gave a clear indication of the project progress. In future, with additional time, the local transcription from the offline model will be implemented onto the device hence providing a clear method to accelerate the audio processing as well as the video processing, within the PL.

The project has a large scope for development as the unique concept of offline S2T has been identified. With a simple adjustment of evolving the device to a larger model there will be an online low power S2T system. Then, from this standpoint, the S2T can be separated between the PS and PL in order to accelerate further and create the offline model.

Hence, the work completed in this project has devised the strategy, highlighted technical qualities, and provided the hardware and software tools needed. This will enable, in the near future, a low power widening access transcription model to be created.

8. References

[1] British Deaf Associate, “BSL STATISTICS,” RNID, 2022. [Online]. Available: https://bda.org.uk/help-resources/#statistics. [Accessed 14 March 2023].

[2] RNID, “How to communicate with someone who is deaf or has hearing loss,” RNID, 2022. [Online]. Available: https://rnid.org.uk/information-and-support/how-to-communicate-with-deaf-people-hearing-loss/?gclid=Cj0KCQjwtsCgBhDEARIsAE7RYh3Fz9o0Ikxnp7JQrBrsKaM1MZJFAbmDmZJwwORP_LN9dRG81n_y9csaAq8wEALw_wcB. [Accessed 14 March 2023].

[3] E. James, “Caring alone,” 1 February 2019. [Online]. Available: https://www.barnardos.org.uk/sites/default/files/uploads/Barnardos%20Caring%20Alone%20report.pdf. [Accessed 14 March 2023].

[4] South East Hearing Care Centres, “What Level Of Hearing Loss Requires A Hearing Aid?,” South East Hearing Care Centres, 2023. [Online]. Available: https://www.hearingcarecentres.co.uk/what-level-of-hearing-loss-requires-a-hearing-aid/#:~:text=Hearing%20loss%20is%20categorised%20as,if%2056%20to%2070%20dB. [Accessed 14 March 2023].

[5] Adido, “History of voice search and voice recognition,” ADIDO, 15 April 2018. [Online]. Available: History of voice search and voice recognition. [Accessed 14 March 2023].

[6] ECH Alliance, “Forecast 2023: Top Five Trends in Voice AI for 2023,” ECH Alliance, 1 February 2023. [Online]. Available: https://echalliance.com/forecast-2023-top-five-trends-in-voice-ai-for-2023/. [Accessed 14 March 2023].

[7] 2022 Zubtitle LLC, “3 Reasons Why Adding Captions To Videos Is Important,” 2022 Zubtitle LLC, 23 September 2022. [Online]. Available: https://zubtitle.com/blog/3-reasons-why-adding-captions-to-videos-is-important#:~:text=Studies%20have%20found%20that%20captions,the%20video%20in%20its%20entirety.. [Accessed 25 November 2022].

[8] RN:ID, “Subtitle it!,” Royal National Institute for Deaf People, [Online]. Available: https://rnid.org.uk/get-involved/campaign-with-us/subtitle-it/. [Accessed 12 November 2022].

[9] Ofcom, “Ofcom’s Guidelines on the Provision of,” Ofcom, 3 February 2021. [Online]. Available: https://www.ofcom.org.uk/__data/assets/pdf_file/0025/212776/provision-of-tv-access-services-guidelines.pdf. [Accessed 12 November 2022].

[10] Neutral Wireless, “5G REMOTE PRODUCTION IN THE MIDDLE OF NOWHERE,” ACCELERATOR MEDIA INNOVATION PROGRAMME, London, 2022.

[11] J. Gordon, “Backhaul (Telecommunications) - Explained,” The Business Professor, 10 March 2022. [Online]. Available: https://thebusinessprofessor.com/en_US/communications-negotiations/back-haul-telecommunications-definition. [Accessed 14 March 2023].

[12] S. Hickey, “IBC ACCELERATORS: 2022 PROJECT OF THE YEAR AWARD ANNOUNCED,” IBC, 8 February 2023. [Online]. Available: https://www.ibc.org/news/ibc-accelerators-2022-project-of-the-year-award-announced/9434.article. [Accessed 6 March 2023].

[13] P. Farley, “Use cases for Speech-to-Text,” Microsoft, 18 July 2022. [Online]. Available: https://learn.microsoft.com/en-us/legal/cognitive-services/speech-service/speech-to-text/transparency-note. [Accessed 14 March 2023].

[14] H. Lavi, “Measuring greenhouse gas emissions in data centres: the environmental impact of cloud computing,” Climatiq, 21 April 2022. [Online]. Available: https://www.climatiq.io/blog/measure-greenhouse-gas-emissions-carbon-data-centres-cloud-computing. [Accessed 14 March 2023].

[15] United Nations, “THE 17 GOALS,” United Nations, 2015. [Online]. Available: https://sdgs.un.org/goals. [Accessed 30 March 2023].

[16] J. Antoniou, “The Benefits of Using Subtitles and Closed Captions,” Capital Captions, 25 February 2017. [Online]. Available: https://www.capitalcaptions.com/subtitles-and-captioning/the-benefits-of-using-subtitles-or-closed-captioning/#:~:text=Helping%20to%20learn%20different%20languages,of%20language%20as%20they%20go.. [Accessed 14 March 2023].

[17] C. Sanford, “Introduction to Speech Recognition Algorithms: Learn How It Has Evolved,” Rev.ai, 26 July 2021. [Online]. Available: https://www.rev.com/blog/speech-to-text-technology/introduction-to-speech-recognition-algorithms. [Accessed 20 March 2023].

[18] J. CHEN, “What Is a Neural Network?,” Investopedia, 21 September 2021. [Online]. Available: https://www.investopedia.com/terms/n/neuralnetwork.asp#:~:text=A%20neural%20network%20is%20a,organic%20or%20artificial%20in%20nature.. [Accessed 20 March 2023].

[19] I. Mustafa, “Analyzing gene polymorphism and metal folic acid interactions in neural tube defects using optimized deep recurrent neural networks,” *Personal and Ubiquitous Computing,* vol. 10, no. 7, p. 10, 2021.

[20] S. Bansal, “Study of Speech Recognition System Based on Transformer and Connectionist Temporal Classification Models for Low Resource Language,” *Speech and Computer,* vol. 3, no. 31, p. 10, 2022.

[21] Rev.ai, “Microsoft Azure Speech Recognition vs. Rev AI Speech to Text API,” Rev.ai, [Online]. Available: https://www.rev.com/blog/resources/microsoft-azure-speech-recognition-vs-rev-ai-speech-to-text-api. [Accessed 27 March 2023].

[22] N. V. Shmyrev, “Vosk Speech Recognition Toolkit,” 14 December 2022. [Online]. Available: https://github.com/alphacep/vosk-api/. [Accessed 31 January 2023].

[23] S. Sugathan and A. P. James, “Irregular pixel imaging,” in *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, Delhi, 2014.

[24] L. Crockett, R. Elliot, M. Enderwitz and R. Stewart, The ZYNQ Book, Glasgow: Strathclyde Academic Media, 2014.

[25] O'Reilly, “Image Resolution and Memory Capacity,” O'Reilly, [Online]. Available: https://www.oreilly.com/library/view/digital-photography-the/0596008414/ch01s02.html#:~:text=A%20RAW%20image%20isn't,saved%20as%20a%20RAW%20file.. [Accessed 27 March 2023].

[26] M. N. Al-Azam, M. M. Achlaq and C. Darujati, “Moving Image YOLOv3 Process Comparison with Multiple Resolutions and Frames-per-Second,” in *2022 International Seminar on Intelligent Technology and Its Applications (ISITIA)*, Surabaya, Indonesia, 2022.

[27] D. Northcote, L. Crockett, C. Ramsay and R. Stewart, Exploring Zynq MPSoC: With PYNQ and Machine Learning Applications, Glasgow: Strathclyde Academix Media, 2019.

[28] Xilinx, “What is PYNQ?,” AMD/Xilinx, 2019. [Online]. Available: http://www.pynq.io/home.html. [Accessed 1 November 2022].

[29] Xilinx, “Python Productivity for Zynq (PYNQ),” Xilinx, 2019. [Online]. Available: https://pynq.readthedocs.io/en/v2.3/pynq_overlays/pynqz2/pynqz2_base_overlay.html#:~:text=The%20PYNQ%2DZ2%20base%20overlay,out%20on%20the%20headphone%20output.. [Accessed 1 November 2022].

[30] R. Silva, “What Is Bitstream and How Does It Work?,” LifeWire, 22 July 2021. [Online]. Available: lifewire.com/what-is-bitstream-1846846. [Accessed 29 March 2023].

[31] PYNQ, “pynq.overlay Module,” PYNQ, 2023. [Online]. Available: https://pynq.readthedocs.io/en/v2.4/pynq_package/pynq.overlay.html. [Accessed 14 March 2023].

[32] PYNQ, “PYNQ Overlays,” PYNQ , 2023. [Online]. Available: https://pynq.readthedocs.io/en/v2.0/pynq_overlays.html#:~:text=PYNQ%20overlays%20are%20created%20by,to%20design%20an%20overlay%20themselves.. [Accessed 29 March 2023].

[33] M. Ruiz, “The Composable Video Pipeline,” Xilinx, 10 March 2022. [Online]. Available: https://discuss.pynq.io/t/the-composable-video-pipeline/2758/1. [Accessed 1 November 2022].

[34] Vivado, “High Level Design Features,” AMD Vivado, 2018. [Online]. Available: https://www.xilinx.com/products/design-tools/vivado/high-level-design.html#dfx. [Accessed 5 November 2022].

[35] Xilinx, “XUP PYNQ-Z2,” Xilinx, inc, [Online]. Available: https://www.xilinx.com/support/university/xup-boards/XUPPYNQ-Z2.html. [Accessed 27 March 2023].

[36] U. Farooq, “PYNQ-Z1 vs PYNQ-Z2 — Which is better?,” Medium, 15 November 2022. [Online]. Available: https://blog.umer-farooq.com/pynq-z1-vs-pynq-z2-which-is-better-213e3dc15527. [Accessed 5 December 2022].

[37] AMD/Xilinx, “Xilinx Adaptive SoCs,” AMD, 2023. [Online]. Available: https://www.xilinx.com/products/silicon-devices/soc.html. [Accessed 20 March 2023].

[38] arm Developer, “AMBA AXI3 and AXI4 Protocol Specification,” arm Developer, [Online]. Available: https://developer.arm.com/documentation/ihi0022/e/AMBA-AXI3-and-AXI4-Protocol-Specification. [Accessed 10 February 2023].

[39] FPGAs for Beginners, “AXI Introduction Part 1: How AXI works and AXI-Lite transaction example,” YouTube, 3 Febrary 2023. [Online]. Available: https://www.youtube.com/watch?v=p5RIVEuxUds&t=603s. [Accessed 5 February 2023].

[40] K. Kintali and Y. Gu, “Model-Based Design with Simulink, HDL Coder, and Xilinx System Generator for DSP,” MathWorks, 2012.

[41] L. Hardesty, “Better debugger,” MIT News, 24 March 2015. [Online]. Available: https://news.mit.edu/2015/integer-overflow-debugger-outperforms-predecessors-0324. [Accessed 9 February 2023].

[42] AMD/ Xilinx, “AXI Video Direct Memory Access v6.3,” 8 June 2022. [Online]. Available: https://www.xilinx.com/content/dam/xilinx/support/documents/ip_documentation/axi_vdma/v6_3/pg020_axi_vdma.pdf. [Accessed 9 March 2023].

[43] AMD Xilinx, “Video Mixer v5.2 LogiCORE IP Product Guide,” 2021. [Online]. Available: https://docs.xilinx.com/r/en-US/pg243-v-mix. [Accessed 1 February 2023].

[44] AMD Xilinx, “AXI Interconnect v2.1,” Vivado, 17 May 2022. [Online]. Available: https://www.xilinx.com/content/dam/xilinx/support/documents/ip_documentation/axi_interconnect/v2_1/pg059-axi-interconnect.pdf. [Accessed 3 March 2023].

[45] S. S. Michael, “AXI Interconnects Tutorial: Multiple AXI Masters and Slaves in Digital Logic,” All About Circuits, 28 August 2019. [Online]. Available: https://www.allaboutcircuits.com/technical-articles/what-are-axi-interconnects-tutorial-master-slave-digital-logic/#:~:text=An%20AXI%20Interconnect%20manages%20the,each%20of%20the%20five%20channels.. [Accessed 5 February 2023].

[46] AMD Xilinx, “AXI4-Stream Switch,” AMD Xilinx, [Online]. Available: https://docs.xilinx.com/r/en-US/pg085-axi4stream-infrastructure/Extra-Settings. [Accessed 2 March 2023].

[47] AMD Xilinx, “Video Pipelines,” 2018. [Online]. Available: https://docs.xilinx.com/r/en-US/ug1449-multimedia/Video-Pipelines?tocId=6acJvklT6D0DklPb_Vx4bg. [Accessed 27 February 2023].

[48] M. Russell and S. Fischaber, “OpenCV based road sign recognition on Zynq,” in *11th IEEE International Conference on Industrial Informatics (INDIN)*, Bochum, Germany, 2013.

[49] S. Tiwari and P. R. Muduli, “Convolutional Neural Network-based ECG Classification on PYNQ-Z2 Framework,” in *2022 IEEE International Conference on Electronics, Computing and Communication Technologies (CONECCT)*, Bangalore, India, 2022.

[50] G. Sahu, “A Dual-Channel Dehaze-Net for Single Image Dehazing in Visual Internet of Things Using PYNQ-Z2 Board,” *IEEE Transactions on Automation Science and Engineering,* vol. 10, no. 1109, p. 15, 2022.

[51] A. Mishra, “What is Rev AI's Accuracy?,” Rev.ai, 1 January 2022. [Online]. Available: https://help.rev.ai/en/articles/3813288-what-is-rev-ai-s-accuracy. [Accessed 5 November 2022].

[52] AMD/Xilinx, “Python Productivity for Zynq (Pynq),” AMD, 2019. [Online]. Available: https://pynq.readthedocs.io/en/v2.0/index.html. [Accessed 20 November 2022].

[53] Analog Devices, “SigmaDSP Stereo, Low Power, 96 kHz, 24-Bit Audio Codec with Integrated PLL,” 2018. [Online]. Available: https://www.analog.com/media/en/technical-documentation/data-sheets/adau1761.pdf. [Accessed 1 March 2023].

[54] InAccel, “Faster Inference: Real benchmarks on GPUs and FPGAs,” Medium, 10 July 2020. [Online]. Available: https://inaccel.medium.com/faster-inference-real-benchmarks-on-gpus-and-fpgas-fa3df04d51d7. [Accessed 22 March 2023].

[55] National Instruments, “FPGA Fundamentals: Basics of Field-Programmable Gate Arrays,” National Instruments, 23 March 2023. [Online]. Available: https://www.ni.com/en-gb/shop/electronic-test-instrumentation/add-ons-for-electronic-test-and-instrumentation/what-is-labview-fpga-module/fpga-fundamentals.html. [Accessed 24 March 2023].

[56] S. Prasad, Contiguous and Non-Contiguous memory allocation in Operating System, Ranchi: Mukherjee University, 2020.

[57] C. Carmine and C. Ilioudis, “EE469: Digital Signal Proccessing,” Strathclyde Academic Media, Glasgow, 2022.

[58] Xilinx, “Avnet Ultra96-V2,” Xilinx, inc, [Online]. Available: https://www.xilinx.com/products/boards-and-kits/1-vad4rl.html. [Accessed 17 March 2023].

[59] Xilinx, “Zynq UltraScale+ MPSoC ZCU104 Evaluation Kit,” Xilinx, inc, [Online]. Available: https://www.xilinx.com/products/boards-and-kits/zcu104.html. [Accessed 17 March 2023].

[60] R. DiCecco, “Caffeinated FPGAs: FPGA framework For Convolutional Neural Networks,” *International Conference on Field-Programmable Technology (FPT),* vol. 10, no. 1109, pp. 265-268, 2016.

[61] B. Chaulagain, “Plant leaf recognition,” *Bachelor in Computer Engineering,* vol. 10, no. 13140, p. September, 2017.