Audio Processing

Aaron Zhao, Imperial College London

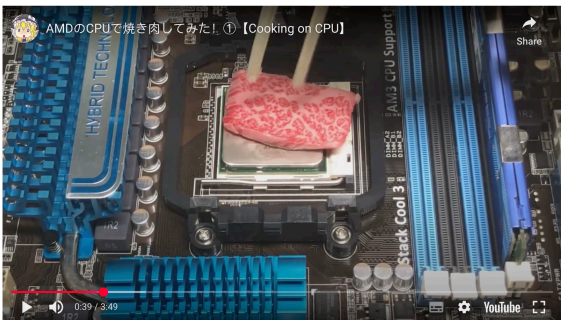
An Introduction to Lab2

What is in this lab?

- Handling audio input and output.
- Understand different modulation schemes.
- Understand how to handle a large hardware design.
- Build an understanding of the OverLay technology.
- Understand how to build multiple hardware IPs and integrate them to a larger design.

Audio processing in Software

- Use the BaseOverLay (no special hardware IPs), almost the same as how computers today you have at hands handle audio processing – **let the CPU do all of the work**

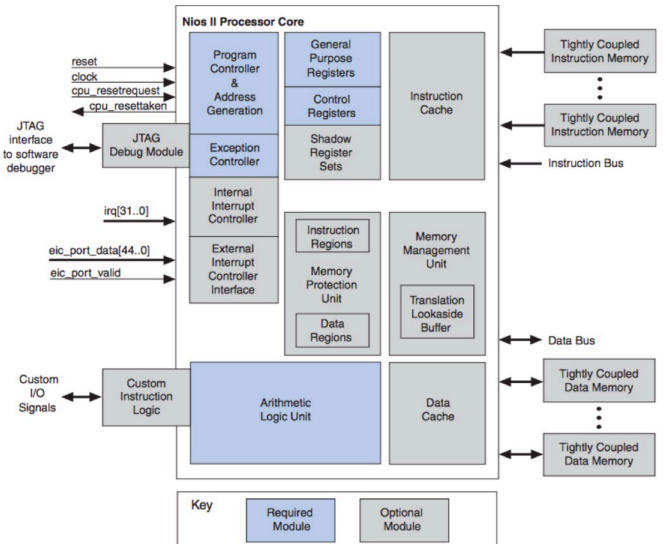


https://www.youtube.com/watch?v=wrmu_8LH88U&t=40s

Processor architecture

- Register files
- Arithmetic logic unit (ALU)
- iCache and dCache
- Instruction decoding
- Instruction bus and data bus

Processor architecture (ii)

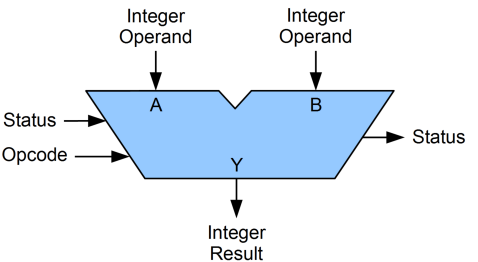


Processor architecture - Register files

- The ISA (Instruction Set Architecture) will always define a set of registers which are used to store and load data between memory and the functional units on the chip.
- Register file is a set of registers that can be accessed by the processor.
- Different types of registers
  - Data storage (eg. R0-R7 in ARM)
  - Instruction control (eg. PC, program counter)
  - Special status and control registers (eg. carry, zero and overflow flags)
- Possible optimizations
  - Register renaming

Processor architecture - ALU

- Normally a piece of hardware that performs arithmetic and logical operations, it normally takes
  - two inputs
  - an op-code
  - produces one output
  - maybe status in and outputs



Processor architecture - Cache system

- There are many design choices for the cache system
- Cache size, associativity, and replacement policy
- Data cache and instruction cache
- The general idea is to store frequently accessed data in the cache (which is closer to the processing unit) to reduce the time it takes to access the data

Processor architecture - Cache system

This can soon become very complex, especially when you consider the different levels of cache that are present in modern multi-processors.:

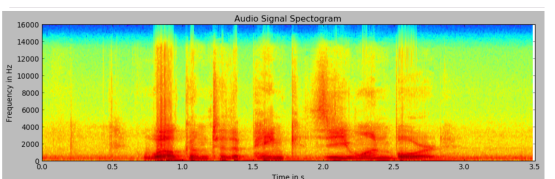
- Caching complicates the ordering of all operations
  - A memory location can be present in multiple caches
  - How can all processors see the same value? Or how can they see the same global order of all memory operations?
- It also affects the ordering of operations on a single memory location
  - A single memory location can be present in multiple caches
  - Makes it difficult for processors that have cached the same location to have the correct value of that location (in the presence of updates to that location)

Consistency and Coherence

- Memory consistency: Global ordering of accesses to all memory locations, it matters more to the programmer to reason about correctness in parallel systems.
- Cache Coherence: Local ordering of accesses to each cache block

The processor you will use

- An ARM processor on the PYNQ board (Zynq-7000 SoC)
- A lot of the time, on an IoT system, programming the CPU can be annoying because they DO NOT HAVE full system software stacks (like Linux) running on them.
  - Side note: this is how your seniors suffered previously with the NIOSII ecosystem with Intel.
- In this course, PYNQ's software stack comes with a variant of Linux (somehow a strange one…) that runs on the ARM processor.
- You will use Python to run code on the ARM processor to handle audio input and output.



Audio processing on the ARM processor

PDM (Pulse Density Modulation)

PDM is a form of modulation used to represent an analog signal with a binary signal. The relative density of the pulses corresponds to the analog signal's amplitude.

MEMS (Micro-Electro-Mechanical Systems) microphone on your PYNQ board records in the PDM format

PCM (Pulse Code Modulation)

PCM is a method used to digitally represent analog signals. This is done by sampling the amplitude of the analog signal at regular intervals and quantizing the sampled values into discrete levels.

It can be then processed in the digital domain more easily, such as applying filters, effects. More importantly, it can be then easily stored in your beloved format like WAV, MP3, FLAC, etc.

Audio processing on the ARM processor (ii)

PWM (Pulse Width Modulation)

PWM is used to encode a message into a pulsing signal. The width of each pulse corresponds to the amplitude of the analog signal at that time.

This is often used for **controlling power to devices**, such as motors or LEDs, but can also be used for audio output, where **the width of the pulses corresponds to the audio signal's amplitude**. This simplifies the hardware requirements such as the DAC design for audio output.

All of these modulation schemes should be familiar to you from your signal processing course!

In Task2A, you will do a software implementation of PDM to PCM conversion on the ARM processor (actually, you just run the code we provide).

Audio processing (Hardware)

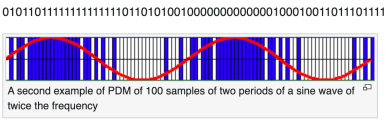
Task 2B: Create a audio PDM-to-PCM converter in hardware (on the FPGA fabric).

- Use Vivado's IP generator to create a Decimation Filter.
- Really just a fancy name for a low-pass FIR filter (to avoid aliasing noise) followed by down-sampling.
- Cascaded Integrator Comb (CIC) filter is a popular choice for this application.
- Your task is then to pack a new IP that performs the PDM to PCM conversion using this filter.

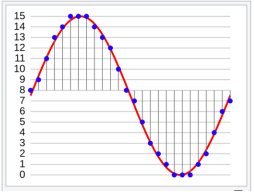
Audio processing (Hardware)

Task 2C: Handling PCM data

- PDM data is a bipolar 1-bit stream (high density of 1s means high amplitude).

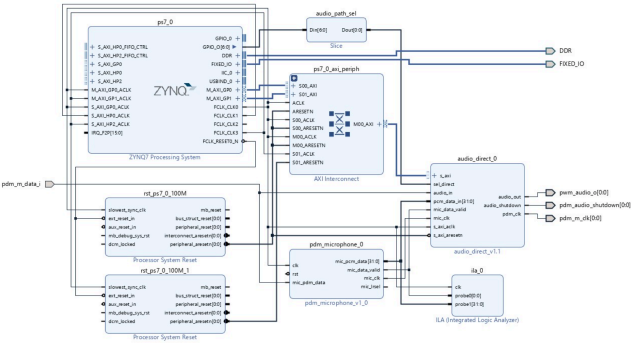


- PCM data is 32-bit signed integers (digital codes).
  - sampling and quantization



Audio processing (Hardware)

Putting it all together



- pdm_microphone: captures PDM data from the microphone
- audio_direct: handles PCM data to the audio output (PWM)
- you then have all three modulation schemes working together!

Understanding the OverLay technology

So in the previous lab, we have changed the Python lib and also ran the Makefile to build a new driver for our new hardware IP.

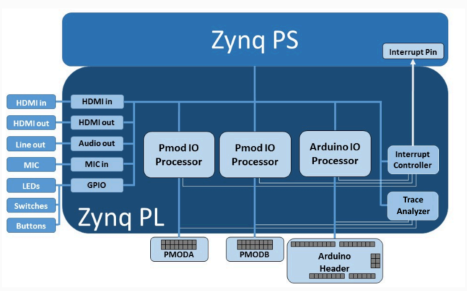
But how does this whole thing work?

- In a normal computer
  - all the hardware components are “there” – I mean they are either on the motherboard or packaged in the same chip (SoC).
  - From the software side, the OS just needs to load the appropriate driver to interact with the hardware.
- In an FPGA-based system
  - the hardware components are not “there” until you program the FPGA fabric to implement them.
  - This is where the OverLay technology comes in. It allows you to dynamically reconfigure the FPGA fabric to implement different hardware components as needed.

Understanding the OverLay technology

For example, on day 1, I can program the FPGA to implement a hardware accelerator for audio processing. On day 2, I can reprogram the FPGA to implement a hardware accelerator for image processing. This is very powerful because it allows you to have a flexible and adaptable system.

But then for all these different hardware components, the software side needs to know how to interact with them. This is where the drivers come in.



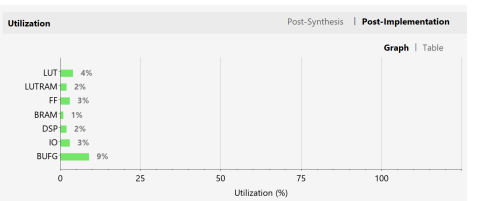
Evaluating hardware resource and performance

- FPGA resource usage
  - Look at the resource usage report generated by Vivado after synthesis and implementation.
  - Key resources to consider include:
    - Look-Up Tables (LUTs)
    - Flip-Flops (FFs)
    - Block RAMs (BRAMs)
    - Digital Signal Processing (DSP) slices
- Performance metrics
  - Latency: The time taken for a single data sample to be processed from input to output.
  - Throughput: The rate at which data samples can be processed, typically measured in samples per second (SPS) or Hertz (Hz).
  - **Maximum operating frequency**: The highest clock frequency at which the design can reliably operate, as determined by timing analysis.

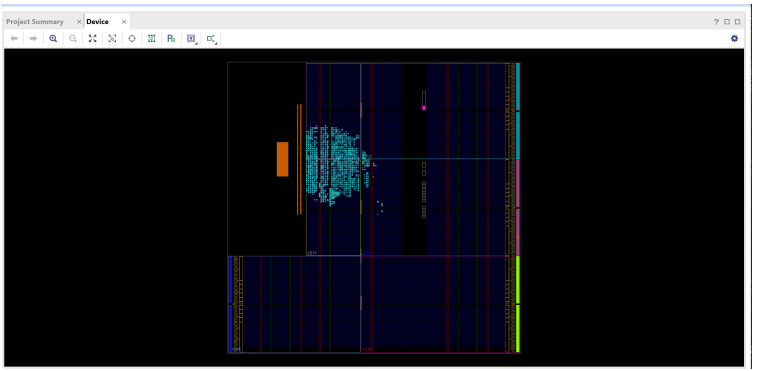
Evaluating hardware resource and performance

You can gain these information from:

- **Synthesis report**: Provides initial resource usage estimates.
- **Implementation report**: Gives detailed resource usage and timing analysis. The timing analysis report offers insights into the maximum operating frequency and potential timing violations.



Evaluating hardware resource and performance



- You can visualise the layout of your design on the FPGA fabric using Vivado's device view.
- Some hardcore FPGA designers may even go down to the level of floorplanning to optimise the placement of critical components for better performance (eg. minimise wiring delays).

An Introduction to Lab3

What is in this lab?

- Understand how to interact with external APIs (LLMs, ASR, TTS)
- A bit of the theory behind LLMs and speech processing
- Understand OpenAI API
- Understand LLM inference, tool use and agentic flows
- Build a simple voice assistant that can interact with the physical world
- A bit of 3D printing

ASR and TTS

ASR (Automatic Speech Recognition)

- Convert spoken language into text
- Popular models: DeepSpeech, Wav2Vec, Whisper
- Applications: Voice assistants, transcription services

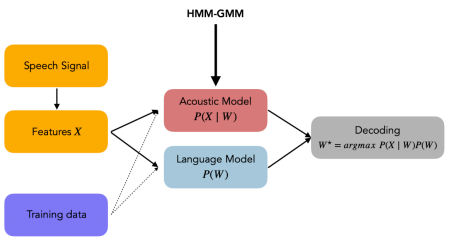
TTS (Text-to-Speech)

- Convert text into spoken language
- Popular models: Tacotron, WaveNet, FastSpeech
- Applications: Voice assistants, audiobooks

How were they done in the past?

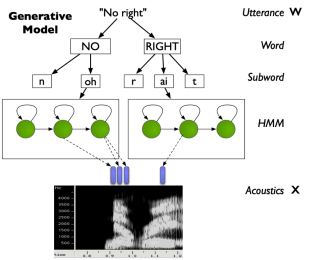
- ASR: Feature extraction (MFCCs), acoustic modeling (HMMs, GMMs), language modeling (n-grams)
- TTS: Concatenative synthesis, parametric synthesis (HMM-based)
- Limitations: Limited vocabulary, poor naturalness, difficulty handling accents and noise

HMM-GMM based ASR system (Hidden Markov Models and Gaussian Mixture Models)



How were they done in the past?

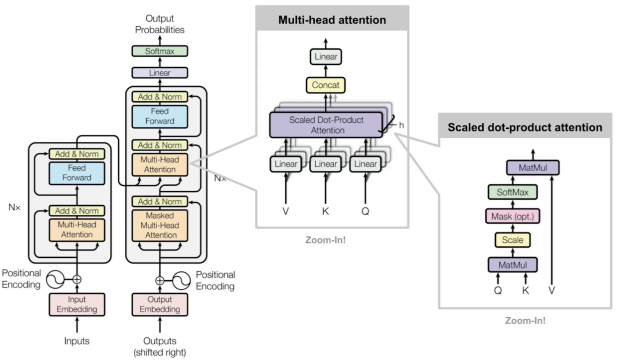
In the pre deep-learning era, ASR systems were typically built using a combination of statistical models and signal processing techniques.



- Find good features to represent the audio signal (eg. MFCCs)
- Use acoustic models (eg. HMMs, GMMs) to model the relationship between audio features and phonemes
- Same as old-school vision systems using hand-crafted features with statistical models and only deal with a small amount of data

Modern approaches to ASR and TTS

- Deep learning models (mainly Transformer backbones with different heads)
- The same base model architecture used in today's LLMs
- In fact, many multi-modal models (eg. GPT-4, Claude-3) can handle both ASR and TTS tasks, these are now "cooked in" capabilities of LLMs



Modern Transformers

- A transformer layer now is: an Attention layer + a Feedforward layer
  - **Attention layer**: allows the model to focus on different parts of the input sequence when making predictions
  - Different types of attentions: MLA in DeepSeek, GQA in LLaMA, etc.
  - **Feedforward layer**: processes the information from the attention layer to produce the final output for
  - Different styles such as choices of different activation choices (ReLU, GELU, SwiGLU, etc.), different sparsity patterns (Mixture of Experts, etc.)

The attention mechanism is arguably one of the most important innovations in deep learning in recent years.

This basically enabled the early success of Transformer models in NLP, and later on in multi-modal models.

Modern Transformers

Another, arguably equally important innovation is the use of large-scale pre-training on massive datasets.

This is known as the **Scaling Laws**: it basically states that the performance of deep learning models improves as the size of the model and the amount of training data increases.

- Purely empirical observation initially, later backed up by some "theoretical analysis".
- The early scaling laws observed that we should scale up **equally in model size, dataset size, given compute**.
- This powered the initial wave of large language models (GPT-2, GPT-3, etc.)

Calling to APIs

- Fortunately, you do not have to build these models from scratch or even deploying them on local GPU machines.
- We are going to call through OpenAI's API to use their hosted models for ASR, TTS and LLM inference.
- You will learn how to call these APIs using Python in the lab.
- The LLM API calls are getting more and more sophisticated, with support for tool use and agentic flows.

Calling to APIs

OpenAI has recently introduced a new endpoint called **Responses** that is intended to replace the older **Chat Completions** endpoint.

What you would need to do is to call into the endpoint with your input text, and the model will return a response based on the input.

```
from openai import OpenAI

client = OpenAI()
resp = client.responses.create(
    model="gpt-4.1-mini",
    # input can be a plain string:
    input="Hi.")
```

Tool Use

- Tool use allows LLMs to interact with external systems or APIs to perform specific tasks.
- This is done by defining a set of tools that the LLM can call, along with their input and output formats.

```
resp = client.responses.create(
    model="gpt-4.1-mini",
    input="What is the weather like in London?",
    tools=[
        {
            "name": "get_weather",
            "description": "Get the current weather.",
            "parameters": {
                "type": "object",
                "properties": {
                    "location": {
                        "type": "string"}},
                "required": ["location"]}}])
```

Agentic Flows

You can also define more complex agentic flows where the LLM can decide which tools to use and in what order based on the input.

OpenAI provides an AgentSDK for this purpose. You can also find a lot of standard tool protocols, such as the MCP protocol, that define how LLMs can interact with external systems.

In the lab, we will only play with simple LLM calls (no tool use or agentic flows), but you can explore these advanced features on your own later.

3D Printing

We will use 3D printers in the department to print out the case for your PYNQ board.

We make the case design files available for you to download and print. We also made the design minimal so that you can easily modify it if you want to add some custom features later on

Mid-term Lab Oral

How would you be graded?

1. You will be assessed as a team
2. You will be asked to demo a working chatbot (Your system works, 50%)
   - What is the end-to-end latency of your system (from speak-llm-listen)?
   - How did you evaluate this?
   - What improvements have you made to improve this end-to-end latency performance?
   - The better the system/evaluation is, the higher the mark is.
3. Technical questions (50%):
   - Like all other lab orals, we would ask you technical questions to check the team's understanding on several topics.
   - The questions will be taken from a pre-determined question-bank

Final Group Project

What is in your toolbox?

- IoT system dev (from drivers to integration to user program)
- FPGA hardware design (from basic Verilog to Vivado IP packging to system integration)
- Handling API calls (LLMs, Whisper and much more!)
- 3D printing
- Database

What is in the final group project?

- Use what you have learned in the labs to design a system that has information flowing from sensors to cloud server and then back to sensor or a local processing node.
- Minimum functional requirements:
  - Local processing of the data
  - Make use of the FPGA fabric
  - Establishing a cloud server to process events/information
  - Communicating information from the server back to the nodes in way that the local processing can be impacted.
  - Use of at least two nodes

What I hope you have learned?

In an abstract way, I would say 99% of the computation is about:

1. The movement of the data
2. The processing of the data

So hopefully in this course you have learned how systems can be designed to perform the above two aspects.