1. **Chuyển MediaBuffer thành short[]**
   * Mỗi cặp short trong mảng đại diện cho một frame audio:
     + short[i] → Kênh trái
     + short[i+1] → Kênh phải
2. **Tách dữ liệu thành 2 mảng riêng cho từng kênh**
   * leftChannel[]: chứa dữ liệu kênh trái
   * rightChannel[]: chứa dữ liệu kênh phải
3. **Xóa âm beep riêng biệt trên từng kênh**
   * Gọi removeBeep(leftChannel, leftOutput)
   * Gọi removeBeep(rightChannel, rightOutput)
4. **Ghép lại thành mảng short[] mới**
   * Xen kẽ dữ liệu từ leftOutput[] và rightOutput[].
5. **Chuyển short[] đã xử lý thành MediaBuffer**
   * Tạo ByteBuffer từ short[], gán vào MediaBuffer.

**1. Trích xuất dữ liệu từ MediaBuffer**

private short[] mediaBufferToShortArray(MediaBuffer mediaBuffer) {

ByteBuffer byteBuffer = mediaBuffer.getData().asReadOnlyBuffer();

byteBuffer.rewind();

int shortCount = byteBuffer.remaining() / 2; // Mỗi short 2 byte

short[] shortArray = new short[shortCount];

byteBuffer.asShortBuffer().get(shortArray);

return shortArray;

}

**2. Xử lý loại bỏ âm beep trên từng kênh**

private MediaBuffer processStereoAudio(MediaBuffer inputBuffer) {

// Chuyển MediaBuffer thành short[]

short[] inputData = mediaBufferToShortArray(inputBuffer);

int totalSamples = inputData.length;

if (totalSamples % 2 != 0) {

throw new IllegalArgumentException("Số lượng mẫu không hợp lệ (phải là số chẵn cho stereo)");

}

int numFrames = totalSamples / 2;

short[] leftChannel = new short[numFrames];

short[] rightChannel = new short[numFrames];

// Tách dữ liệu stereo

for (int i = 0; i < numFrames; i++) {

leftChannel[i] = inputData[2 \* i]; // Kênh trái

rightChannel[i] = inputData[2 \* i + 1]; // Kênh phải

}

short[] leftOutput = new short[numFrames];

short[] rightOutput = new short[numFrames];

// Xóa beep trên từng kênh

removeBeep(leftChannel, leftOutput);

removeBeep(rightChannel, rightOutput);

// Ghép lại thành dữ liệu stereo

short[] outputData = new short[totalSamples];

for (int i = 0; i < numFrames; i++) {

outputData[2 \* i] = leftOutput[i]; // Kênh trái

outputData[2 \* i + 1] = rightOutput[i]; // Kênh phải

}

// Chuyển lại thành MediaBuffer

return shortArrayToMediaBuffer(outputData);

}

**3. Chuyển short[] thành MediaBuffer**

java

CopyEdit

private MediaBuffer shortArrayToMediaBuffer(short[] shortArray) {

ByteBuffer byteBuffer = ByteBuffer.allocate(shortArray.length \* 2);

byteBuffer.asShortBuffer().put(shortArray);

MediaBuffer outputBuffer = new MediaBuffer();

outputBuffer.setData(byteBuffer);

return outputBuffer;

}

Audacity

**1. Ghi short[] ra file PCM**

private void savePCMFile(String filename, short[] data) {

try (FileOutputStream fos = new FileOutputStream(filename)) {

ByteBuffer buffer = ByteBuffer.allocate(data.length \* 2);

buffer.order(ByteOrder.LITTLE\_ENDIAN); // PCM 16-bit thường dùng Little Endian

buffer.asShortBuffer().put(data);

fos.write(buffer.array());

fos.flush();

Log.d("AudioDebug", "Saved: " + filename);

} catch (IOException e) {

e.printStackTrace();

}

}

**2. Tích hợp vào hàm processStereoAudio()**

Bạn lưu cả file **đầu vào** và **đầu ra** để so sánh:

java

CopyEdit

private MediaBuffer processStereoAudio(MediaBuffer inputBuffer) {

// Chuyển MediaBuffer thành short[]

short[] inputData = mediaBufferToShortArray(inputBuffer);

savePCMFile("/sdcard/input\_audio.pcm", inputData); // Lưu file đầu vào

int totalSamples = inputData.length;

if (totalSamples % 2 != 0) {

throw new IllegalArgumentException("Số lượng mẫu không hợp lệ (phải là số chẵn cho stereo)");

}

int numFrames = totalSamples / 2;

short[] leftChannel = new short[numFrames];

short[] rightChannel = new short[numFrames];

// Tách dữ liệu stereo

for (int i = 0; i < numFrames; i++) {

leftChannel[i] = inputData[2 \* i]; // Kênh trái

rightChannel[i] = inputData[2 \* i + 1]; // Kênh phải

}

short[] leftOutput = new short[numFrames];

short[] rightOutput = new short[numFrames];

// Xóa beep trên từng kênh

removeBeep(leftChannel, leftOutput);

removeBeep(rightChannel, rightOutput);

// Ghép lại thành dữ liệu stereo

short[] outputData = new short[totalSamples];

for (int i = 0; i < numFrames; i++) {

outputData[2 \* i] = leftOutput[i]; // Kênh trái

outputData[2 \* i + 1] = rightOutput[i]; // Kênh phải

}

savePCMFile("/sdcard/output\_audio.pcm", outputData); // Lưu file đầu ra

// Chuyển lại thành MediaBuffer

return shortArrayToMediaBuffer(outputData);

}

**Hướng dẫn mở file PCM trên Audacity**

1. Mở **Audacity**.
2. **File > Import > Raw Data**.
3. Chọn file **input\_audio.pcm** hoặc **output\_audio.pcm**.
4. Cấu hình:
   * **Encoding**: Signed 16-bit PCM
   * **Byte Order**: Little Endian
   * **Channels**: 2 (Stereo)
   * **Sample Rate**: 44100 Hz hoặc 48000 Hz (tùy nguồn audio của bạn).
5. Click **Import**, sau đó nghe thử file.

Case 2:

**1. Gộp tất cả MediaBuffer thành một mảng short[]**

* Lưu timestamp của từng MediaBuffer cùng với số mẫu (short[]) tương ứng.

**2. Xóa âm beep trên toàn bộ dữ liệu**

* Xử lý short[] với removeBeep().

**3. Chia lại dữ liệu đã xử lý thành MediaBuffer**

* **Giữ nguyên timestamp gốc** khi chia lại buffer.

**📌 Hiện thực**

**🔹 1. Gộp tất cả MediaBuffer và lưu timestamp**

private static class BufferInfo {

long timestamp;

short[] data;

BufferInfo(long timestamp, short[] data) {

this.timestamp = timestamp;

this.data = data;

}

}

private List<BufferInfo> mergeMediaBuffersWithTimestamps(List<MediaBuffer> buffers) {

List<BufferInfo> bufferInfos = new ArrayList<>();

int totalSamples = 0;

for (MediaBuffer buffer : buffers) {

short[] data = mediaBufferToShortArray(buffer);

bufferInfos.add(new BufferInfo(buffer.getTimestamp(), data));

totalSamples += data.length;

}

return bufferInfos;

}

👉 **Lưu timestamp của mỗi MediaBuffer để sử dụng lại sau khi xử lý.**

**🔹 2. Xử lý removeBeep trên toàn bộ dữ liệu**

private short[] processFullAudio(short[] inputData) {

short[] outputData = new short[inputData.length];

// Xóa beep trên toàn bộ audio

removeBeep(inputData, outputData);

return outputData;

}

**🔹 3. Chia short[] đã xử lý thành MediaBuffer có timestamp chính xác**

private List<MediaBuffer> splitIntoMediaBuffersWithTimestamps(short[] processedData, List<BufferInfo> originalBuffers) {

List<MediaBuffer> outputBuffers = new ArrayList<>();

int totalSamples = processedData.length;

int processedOffset = 0;

for (BufferInfo info : originalBuffers) {

int sampleCount = info.data.length; // Số mẫu của buffer gốc

if (processedOffset + sampleCount > totalSamples) {

break; // Tránh lỗi khi cắt dữ liệu vượt quá length

}

// Cắt đúng phần dữ liệu đã xử lý

short[] chunk = Arrays.copyOfRange(processedData, processedOffset, processedOffset + sampleCount);

processedOffset += sampleCount;

// Tạo MediaBuffer mới với timestamp gốc

MediaBuffer buffer = shortArrayToMediaBuffer(chunk);

buffer.setTimestamp(info.timestamp);

outputBuffers.add(buffer);

}

return outputBuffers;

}

👉 **Timestamp gốc được giữ lại chính xác khi chia lại buffer.**

**🔹 4. Hàm chính xử lý toàn bộ**

public List<MediaBuffer> processAllBuffersWithTimestamps(List<MediaBuffer> inputBuffers) {

// 1. Gộp tất cả MediaBuffer và lưu timestamp

List<BufferInfo> bufferInfos = mergeMediaBuffersWithTimestamps(inputBuffers);

// Chuyển thành một mảng short[] duy nhất

short[] fullInputData = bufferInfos.stream()

.flatMapToInt(info -> Arrays.stream(info.data))

.toArray();

savePCMFile("/sdcard/full\_input.pcm", fullInputData); // Debug file gốc

// 2. Xóa beep trên toàn bộ dữ liệu

short[] fullOutputData = processFullAudio(fullInputData);

savePCMFile("/sdcard/full\_output.pcm", fullOutputData); // Debug file sau xử lý

// 3. Chia nhỏ dữ liệu đã xử lý thành MediaBuffer với timestamp chính xác

return splitIntoMediaBuffersWithTimestamps(fullOutputData, bufferInfos);

}

**📌 Hướng dẫn sử dụng**

List<MediaBuffer> processedBuffers = processAllBuffersWithTimestamps(inputBuffers);

* **inputBuffers**: Danh sách MediaBuffer gốc.
* **processedBuffers**: Danh sách MediaBuffer đã qua removeBeep() nhưng **giữ nguyên timestamp**.

**Code Kotlin**

**🔹 1. Gộp tất cả MediaBuffer và lưu timestamp**

data class BufferInfo(val timestamp: Long, val data: ShortArray)

private fun mergeMediaBuffersWithTimestamps(buffers: List<MediaBuffer>): List<BufferInfo> {

return buffers.map { buffer ->

BufferInfo(buffer.timestamp, mediaBufferToShortArray(buffer))

}

}

👉 **Lưu timestamp gốc của từng MediaBuffer để giữ nguyên sau khi xử lý.**

**🔹 2. Xử lý removeBeep trên toàn bộ dữ liệu**

private fun processFullAudio(inputData: ShortArray): ShortArray {

val outputData = ShortArray(inputData.size)

removeBeep(inputData, outputData)

return outputData

}

**🔹 3. Chia ShortArray đã xử lý thành MediaBuffer với timestamp chính xác**

private fun splitIntoMediaBuffersWithTimestamps(

processedData: ShortArray,

originalBuffers: List<BufferInfo>

): List<MediaBuffer> {

val outputBuffers = mutableListOf<MediaBuffer>()

var processedOffset = 0

for (info in originalBuffers) {

val sampleCount = info.data.size

if (processedOffset + sampleCount > processedData.size) break // Tránh lỗi khi cắt quá size

val chunk = processedData.copyOfRange(processedOffset, processedOffset + sampleCount)

processedOffset += sampleCount

val buffer = shortArrayToMediaBuffer(chunk)

buffer.timestamp = info.timestamp // Gán timestamp gốc

outputBuffers.add(buffer)

}

return outputBuffers

}

👉 **Mỗi MediaBuffer sau khi xử lý vẫn giữ đúng timestamp gốc.**

**🔹 4. Hàm chính xử lý toàn bộ**

fun processAllBuffersWithTimestamps(inputBuffers: List<MediaBuffer>): List<MediaBuffer> {

// 1. Gộp tất cả MediaBuffer và lưu timestamp

val bufferInfos = mergeMediaBuffersWithTimestamps(inputBuffers)

// Chuyển thành một ShortArray duy nhất

val fullInputData = bufferInfos.flatMap { it.data.toList() }.toShortArray()

savePCMFile("/sdcard/full\_input.pcm", fullInputData) // Debug file gốc

// 2. Xóa beep trên toàn bộ dữ liệu

val fullOutputData = processFullAudio(fullInputData)

savePCMFile("/sdcard/full\_output.pcm", fullOutputData) // Debug file sau xử lý

// 3. Chia nhỏ dữ liệu đã xử lý thành MediaBuffer với timestamp chính xác

return splitIntoMediaBuffersWithTimestamps(fullOutputData, bufferInfos)

}

**📌 Hướng dẫn sử dụng**

val processedBuffers = processAllBuffersWithTimestamps(inputBuffers)

* **inputBuffers**: Danh sách MediaBuffer gốc.
* **processedBuffers**: Danh sách MediaBuffer đã qua removeBeep() nhưng **giữ nguyên timestamp**.

Nếu xử lý 2 channel thực hiện trong remove shutter()

**📌 Cập nhật code bên ngoài**

Dưới đây là cách cập nhật lại pipeline xử lý:

**🔹 1. Gộp tất cả MediaBuffer thành một mảng ShortArray (stereo)**

fun mergeMediaBuffersWithTimestamps(buffers: List<MediaBuffer>): List<BufferInfo> {

return buffers.map { buffer ->

BufferInfo(buffer.timestamp, mediaBufferToShortArray(buffer))

}

}

👉 **Không cần tách stereo, chỉ cần gộp toàn bộ dữ liệu.**

**🔹 2. Gọi removeShutter() trực tiếp (đã xử lý 2 channel bên trong)**

fun processStereoBuffersWithTimestamps(inputBuffers: List<MediaBuffer>): List<MediaBuffer> {

// 1. Gộp toàn bộ dữ liệu từ MediaBuffer

val bufferInfos = mergeMediaBuffersWithTimestamps(inputBuffers)

val fullInputData = bufferInfos.flatMap { it.data.toList() }.toShortArray()

savePCMFile("/sdcard/full\_input.pcm", fullInputData) // Debug file gốc

// 2. Xử lý removeShutter() trực tiếp trên dữ liệu stereo

val fullOutputData = ShortArray(fullInputData.size)

removeShutter(fullInputData, fullOutputData)

savePCMFile("/sdcard/full\_output.pcm", fullOutputData) // Debug file sau xử lý

// 3. Chia lại dữ liệu đã xử lý thành MediaBuffer giữ timestamp chính xác

return splitIntoMediaBuffersWithTimestamps(fullOutputData, bufferInfos)

}

👉 **Không cần split/gộp thủ công, chỉ gọi removeShutter() trực tiếp.**

**🔹 3. Chia lại ShortArray đã xử lý thành MediaBuffer với timestamp chuẩn**

fun splitIntoMediaBuffersWithTimestamps(

processedData: ShortArray,

originalBuffers: List<BufferInfo>

): List<MediaBuffer> {

val outputBuffers = mutableListOf<MediaBuffer>()

var processedOffset = 0

for (info in originalBuffers) {

val sampleCount = info.data.size

if (processedOffset + sampleCount > processedData.size) break // Tránh lỗi khi cắt quá size

val chunk = processedData.copyOfRange(processedOffset, processedOffset + sampleCount)

processedOffset += sampleCount

val buffer = shortArrayToMediaBuffer(chunk)

buffer.timestamp = info.timestamp // Gán timestamp gốc

outputBuffers.add(buffer)

}

return outputBuffers

}

👉 **Timestamp được giữ chính xác theo từng MediaBuffer gốc.**

**Hàm lưu PCM vào file**

kotlin

CopyEdit

import java.io.File

import java.io.FileOutputStream

import java.io.IOException

fun savePCMFile(filePath: String, data: ShortArray) {

try {

val file = File(filePath)

val outputStream = FileOutputStream(file)

// Chuyển đổi short[] thành byte[] để ghi vào file

val byteBuffer = ByteArray(data.size \* 2)

for (i in data.indices) {

val value = data[i].toInt()

byteBuffer[2 \* i] = (value and 0xFF).toByte() // Byte thấp

byteBuffer[2 \* i + 1] = ((value shr 8) and 0xFF).toByte() // Byte cao

}

outputStream.write(byteBuffer)

outputStream.close()

println("PCM file saved: $filePath")

} catch (e: IOException) {

e.printStackTrace()

}

}