Bachelorarbeit im Studiengang Audiovisuelle Medien

Weiterentwicklung eines Soundgenerators für Elektrofahrzeuge

Softwaretechnische Umsetzung und Implementierung der Audio Engine *FMOD Studio*

Vorgelegt am 13. Januar 2017
an der Hochschule der Medien Stuttgart
zur Erlangung des akademischen Grades
„Bachelor of Engineering“

Erstprüfer: Prof. Oliver Curdt
Zweitprüfer: Dipl.-Ing. Jochen Lüling
Autor: Max Kersten
Matrikelnummer: 27080

Eidesstattliche Versicherung

Hiermit versichere ich, Max Kersten, an Eides statt, dass ich die vorliegende Bachelorarbeit mit dem Titel

„WEITERENTWICKLUNG EINES SOUNDGENERATORS FÜR ELEKTROFAHRZEUGE – SOFTWARETECHNISCHE UMSETZUNG UND IMPLEMENTIERUNG DER AUDIO ENGINE *FMOD STUDIO*“, selbständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinne nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Stuttgart, den 13.01.2017 _____

Ort, Datum

_____

Max Kersten

Kurzfassung

Diese Arbeit befasst sich mit der softwareseitigen Neuentwicklung eines Soundgenerators für Elektrofahrzeuge in Kooperation mit der Firma *GIGATRONIK Stuttgart GmbH*. Basis dafür ist ein bereits vorhandener Soundgenerator, welcher in einem, auf Elektroantrieb umgebauten, Quad installiert ist. Dieser verwendet zur Klangerzeugung einen Softwaresampler mit Midi-Ansteuerung. Der Fokus dieser Arbeit liegt auf der Umsetzung des neuen Soundgenerators mit Hilfe der Audio Engine *FMOD Studio,* welche normalerweise in Videospielen zum Einsatz kommt. Der Soundgenerator soll in seiner Authentizität verbessert und in seinem Funktionsumfang erweitert werden. Die Entwicklung eines Motorklangerzeugers, welcher dynamisch auf die jeweilige Fahrsituation reagiert, ist Kernaufgabe dieser Arbeit.

Abstract

This thesis is focusing on an innovative software development for a sound generator for electric vehicles in cooperation with *GIGATRONIK Stuttgart GmbH*. The development is based on an already existing sound generator which has been installed in a Quad bike that has been modified to be electric powered. It uses a software sampler with MIDI control for sound generation. The implementation of the new sound generator using the *FMOD Studio* audio engine, which is normally used in video games, is the primary focus of this work. The sound generator will need to be improved in its sound authenticity and extend its functionality. The core task of this work is the development of an engine sound generator, which dynamically reacts to the respective driving situation.

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

Diese Bachelorarbeit befasst sich ausschließlich mit der softwareseitigen Umsetzung des Soundgenerators. Die Konzeption und hardwareseitige Umsetzung wird in der entsprechenden Bachelorarbeit (Müller L. , 2017) beschrieben. An entsprechenden Stellen wird auf diese verwiesen. Um ein volles Verständnis des entwickelten Systems, hardware- und softwareseitig, zu garantieren, ist es dringend erforderlich die oben genannte Bachelorarbeit im Voraus zu lesen.

1.1 Zielsetzung der Arbeit

In Kooperation mit der *GIGATRONIK Stuttgart GmbH* soll ein Soundgenerator für ein Elektrofahrzeug entwickelt werden. Ein von der Firma *GIGATRONIK* entwickeltes Quad mit elektronischem Antriebsstrang wird hierbei als Versuchsträger verwendet. Ein bereits bestehender Soundgenerator soll, basierend auf dessen Schwachstellen, überarbeitet und erneuert werden. Primär gilt es, durch die Entwicklung von neuen Programmstrukturen, die klangliche Dynamik eines Verbrennungsmotors so realitätsnah wie möglich zu reproduzieren. Zudem soll es durch die ausgewählte und verwendete Software möglich sein, neue Soundpakete von verschieden Fahrzeugtypen schnell und ressourcensparend zu implementieren. Die einfache Handhabung der verwendeten Software steht hierbei im Vordergrund, um einen störungsfreien Betrieb jederzeit zu gewährleisten. Alle Funktionen der zur softwareseitigen Umsetzung des Soundgenerators benötigten Programmstrukturen werden im Verlauf der Arbeit beschrieben. Die Erzeugung der zur klanglichen Dynamikentwicklung benötigten Kenngröße der Motorlast steht hierbei im Vordergrund.

Im Verlauf der Arbeit sollen Soundpakete verschiedener Fahrzeugtypen entstehen. Dies bedarf einer audiotechnischen Aufnahme verschiedener Schallquellen realer Fahrzeuge mit Verbrennungsmotor. Die aufgenommenen Soundsamples müssen im Nachbearbeitungsprozess dem realen Fahrzeugklang angepasst werden, damit über die Lautsprecher des Soundgenerators ein authentisches Klangbild wiedergegeben werden kann. Der anfallende Nachbearbeitungsprozess wird im Verlauf der Arbeit im Detail beschrieben. Hierbei wird primär auf die verwendeten Audio Plug-Ins und deren genaue Funktion eingegangen. Dabei soll verdeutlicht werden, welche Audio Plug-Ins für den speziellen Zweck der Synthese von Motorklängen geeignet sind um die emotionale Wirkung auf den Fahrer zu verstärken. Zudem sollen alle benötigten Arbeitsschritte beschrieben werden, die zum

Einfügen neuer Motorklänge in den Soundgenerator nötig sind. Die benötigte Projektstruktur des verwendeten Software-Tools wird hierbei genau analysiert und erklärt. Im Nachhinein soll es möglich sein, neue Soundpakete verschiedener Fahrzeugtypen jederzeit in den Soundgenerator zu integrieren oder bestehende Soundpakete weiter zu verbessern.

1.2 Vorgehensweise

In einem Grundlagenkapitel soll das für die Bachelorarbeit benötigte Grundwissen vermittelt werden. Zunächst soll der Begriff Sounddesign erläutert und anschließend auf die Besonderheiten des Sounddesigns in Videospielen eingegangen werden. Hierbei wird vor allem die Klanggestaltung in Rennsimulationsspielen genauer betrachtet. Der synthetisierte Motorenklang, der in den Spielen vorkommenden Fahrzeuge, spielt hierbei die entscheidende Rolle. Im Anschluss erfolgt eine Beschreibung der für die Synthese von Motorenklängen benötigten Klangsyntheseverfahren, welche in aktuellen Rennsimulationen zum Einsatz kommen. Zum Abschluss des Grundlagenkapitels soll der, in Fahrzeugen zur Vernetzung von Steuergeräten benötigte, CAN-Bus genauer betrachtet werden.

Der praktische Teil der Arbeit befasst sich zunächst mit der Auswahl der passenden Audio Engine, welche zur Synthese der Motorklänge für den Soundgenerator verwendet werden soll. Hierbei werden zunächst die grundlegenden Funktionen einer Audio Engine erklärt und die Vorteile gegenüber einem Software Sampler herausgearbeitet. Im Anschluss werden die zur Ansteuerung der Fahrzeugsounds benötigten Programmstrukturen beschrieben. Die Erklärung der Funktionen zur Erzeugung der klanglichen Lastendynamik steht hierbei im Vordergrund. Abschließend erfolgt eine Beschreibung der softwareinternen Projektstruktur der verwendeten Audio Engine, sowie ein detaillierter Einblick in die Implementierung der entsprechenden Audio Dateien in die Software.

Im praktischen Teil wird zudem der Nachbearbeitungsprozess der Soundsamples genauer betrachtet. Hierbei werden alle anfallenden Arbeitsschritte, die zur kompletten Aufbereitung der einzelnen Audiodateien nötig sind, beschrieben. Die zur Aufbereitung und Bearbeitung verwendeten Audio Plug-Ins werden vorgestellt und deren Funktionen beleuchtet.

Zum Abschluss der Arbeit wird zunächst ein persönliches Fazit gezogen, bevor mögliche zukünftige Erweiterungen des Soundgenerators aufgezeigt werden.

2. Grundlagen

2.1 Sounddesign

Der Begriff Sounddesign entstand Ende der 1970er Jahre im New Hollywood Kino nach dem Zusammenbruch des bis dahin herrschenden Studiosystems. Junge Filmproduzenten, wie George Lucas und Steven Spielberg, und deren Interesse an neuen Technologien hatten zur Folge, dass nicht mehr der akustische Realismus oberste Priorität hatte. Um einen gewissen dramaturgischen Effekt zu erzielen, entstanden kreative Möglichkeiten einen Klang oder ein Geräusch zu bearbeiten. Durch die neuen Erkenntnisse war es jetzt legitim, Geräusche im Film zu ersetzen, sie klanglich zu verfremden oder mit anderen zu unterlegen.

Das Sounddesign kann dazu dienen, die im Bild gezeigten Handlungen zu unterstützen und zu punktieren; es kann aber auch dazu benutzt werden nicht gezeigte Handlungen auditiv zu erzählen. Zudem kann mit Sounddesign Einfluss auf die emotionale Wirkung einer Handlung, einer Person oder eines gezeigten Objektes genommen werden und verstärkt somit die Immersion des jeweiligen Mediums. Dadurch ist es heutzutage in allen audiovisuellen Medien vertreten: im Film und Fernsehen, in der Werbung, im Rundfunk, in Videospielen aber auch in interaktiven Medieninstallationen. [1]

2.2 Sounddesign in Games

Das Sounddesign für Videospiele unterscheidet sich von dem für Filme oder für Hörspiele dahingehend, dass es sich hierbei nicht um ein lineares Medium handelt. Ein Videospiel ist interaktiv und somit kann das Sounddesign nicht an einer Zeitskala definiert werden, sondern muss von den Aktionen des Spielers und der Spielelogik ausgelöst werden. Es kann zeitlich nicht genau vorhergesagt werden, wann welche Aktion ausgeführt wird. Zudem besitzen viele Spiele keinen singularen Handlungsstrang. Das heißt, dass gewisse Handlungsketten den Spielverlauf maßgeblich beeinflussen und somit auch Auswirkungen auf das Sounddesign haben.[2]

Eine weitere Besonderheit ist, dass nicht wie bei linearen Medien alle Einzelteile des Sounddesigns, wie Musik, Dialog, Foleys, Ambience, Effekte und der Dialog in einer finalen Mischung zusammengefügt und aufeinander optimiert werden können. Die einzelnen

[1] Vgl. Busche, 2012
[2] Vgl. Horowitz & Looney, 2014

Elemente können sich dabei gezielt inhaltlich ergänzen. Bei einem Videospiel ist das so nicht möglich. Alle Soundpakete müssen alleinstehend funktionieren, da sich die anderen Komponenten des Sounddesigns komplett verändern können. Ziel hierbei sollte sein, das Sounddesign so interaktiv und vielseitig erscheinen zu lassen, wie das Spielgeschehen.

2.1.1 Motorklang in Rennsimulationen

Der Motorklang hat vor allem in Rennspielen und -simulationen einen sehr hohen Stellenwert. Dieser trägt, mit der Detailtreue des 3D Modells und der Fahrphysik, maßgeblich zur Authentizität des Spiels bei. In diesem Spielegenre hat vermeintlicher Realismus oberste Priorität. Dem Spieler soll das Gefühl vermittelt werden, er sitze in einem echten Rennauto auf einer echten Rennstrecke, welches sich in Realität genauso verhält. Der Großteil der Spieler ist das reale Fahrverhalten der Autos unbekannt, zweifellos aber wissen sie, wie die Autos aussehen und klingen. Somit sind diese zwei Faktoren hauptsächlich für das Realitätsempfinden und die Immersion beim Spieler verantwortlich.

ABBILDUNG 1 RENNSIMULATION VS. REALITÄT[3]

[3]Vgl. Digiprost, 2014

Deshalb wird bei den Entwicklern großen Wert auf den Motorklang gelegt. Zum einen muss die echte Klangcharakteristik des Motors detailgetreu wiedergegeben werden, zum anderen muss das Verhalten des Motors realitätsgetreu nachempfunden werden. Dabei spielen zwei Parameter eine wichtige Rolle: Das Drehzahlverhalten und das Lastverhalten des Motors (vgl. Müller, 2017). Nicht zu vernachlässigen sind dabei auch klangliche Nebenprodukte, wie Turboladergeräusche oder Fehlzündungen. All diese Klangkomponenten und deren realistisches Verhalten führen schlussendlich zu einem authentischen Motorklang, der so beim Konsumenten wahrgenommen wird.[4]

Da die meisten Entwickler keine öffentlichen Dokumentationen zu Verfügung stellen, wie letztendlich genau der Motorsound erzeugt wird und auch sonst recht wenig aussagekräftige Informationsquellen vorhanden sind, ist es zunächst einmal schwer die ganze Klangsynthese nachzuvollziehen.

Die Rennsimulation *Assetto Corsa* von *Kunos Simulazioni* ist aber so konzipiert, dass Kontent von Dritten hinzugefügt werden kann. Dafür bietet der Entwickler ein SDK (Software Development Kit) an, welches für den Ton eine ausführliche Dokumentation der Parameter und ein Beispielprojekt der Audio Engine beinhaltet. Kunos Simulazioni verwendet hierbei die Audio Engine *FMOD Studio* von *Firelight Technologies.*

Der grundlegende Motorsound in *Assetto Corsa* wird dabei wie folgt erzeugt: Die Basis bilden Aufnahmen des realen Fahrzeuges. Dabei wird in gewissen Abständen das gesamte Drehzahlband aufgenommen. Die einzelnen Samples beinhalten eine konstant gehaltene Drehzahl des Motors. Der Samplevorgang wird dabei zweimal durchgeführt, einmal im Leerlauf ohne anliegende Motorlast und ein weiteres Mal unter Volllast.

Die aufgenommenen Samples werden in die Middleware implementiert. Diese werden dort auf den Wertebereich eines vorher definierten „RPM“ Parameter angelegt und überblendet. Die Leerlauf- und Volllastsamples werden dort jeweils auf eigenen Spuren angelegt und verlaufen zu jedem Zeitpunkt parallel zueinander. Die fehlenden Zwischenwerte der Drehzahl werden über den internen Pitch Algorithmus der Audio Engine interpoliert. Die Motorlast wird mittels Lautstärkeautomation der einzelnen Spuren über den Parameter „throttle“ erzeugt.

[4] Vgl. Donnelly, 2014

Über diese zwei Parameter und deren Ansteuerung wird schlussendlich der Motorsound generiert.[5]

2.3 Klangsynthese

Im Folgenden werden Klangsyntheseverfahren beschrieben, die sich für die Erzeugung eines authentischen Motorsounds eignen und aktuell in der Praxis angewendet werden.

2.3.1 Sampling

Sampling hat in der Musikproduktion und in der Tontechnik verschiedene Bedeutungen. Daher muss zwischen diesen Bedeutungen unterschieden werden. Grundlegend ist damit die Aufzeichnung akustischer Ereignisse gemeint.

Zum einen wird bei der Konvertierung eines analogen in ein digitales Signal von Sampling gesprochen. Dabei wird ein zeitkontinuierliches analoges Signal durch zeitlich diskrete Abtastung und Zuweisung zu einem diskreten Wertebereich in ein digitales Signal umgewandelt. Die Abtastwerte entsprechen dabei den Amplitudenwerten des originalen analogen Signals.[6] Um die Genauigkeit der Analog-Digital-Wandlung zu vergrößern muss die Abtastfrequenz und der Wertebereich, die Bittiefe, erhöht werden.

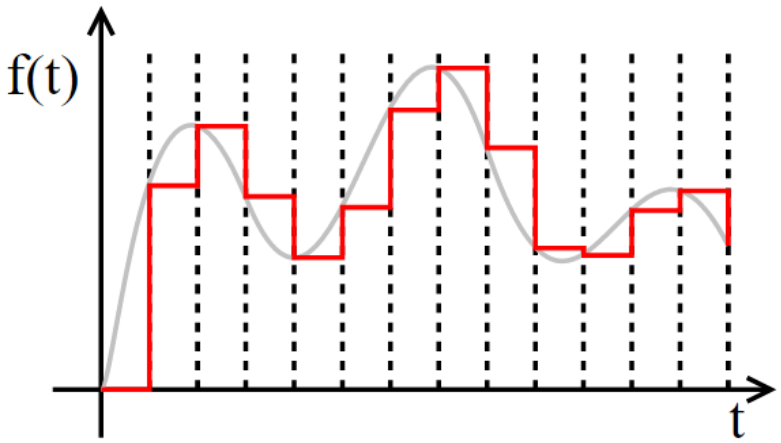

ABBILDUNG 2 ANALOGES SIGNAL UND DESSEN DIGITALES PENDANT[7]

[5] Vgl. Kunos Simulazioni, 2016
[6] Vgl. Raffaseder, 2010, S. 133
[7] Vgl. Wikipedia, 2015

In der Musikproduktion ist Sampling aber auch eine Methode zur Klangsynthese. Damit ist gemeint, dass beliebige aufgezeichnete akustische Ereignisse, wie zum Beispiel ein Ton eines Musikinstrumentes oder ein Geräusch, auf Knopf- oder Tastendruck, auf einem MIDI-Keyboard oder Controller, wiedergegeben werden können. Es besteht zudem die Möglichkeit, das Sample in unterschiedlichen Tonhöhen und mehrstimmig wiederzugeben. Mit dieser Technik lassen sich recht einfach reale Instrumente abbilden.

Problematik bei der Verwendung nur eines Samples und dessen Veränderung der Tonhöhe hat aber zu Folge, dass nicht nur der Grundton verändert, sondern das gesamte Spektrum verschoben wird. Somit wird die Obertonstruktur verfälscht wiedergegeben. Durch Verwendung mehrerer Samples für die jeweiligen Tonhöhen und Dynamikstufen, sogenanntes Multisampling, kann das Originalinstrument authentischer imitiert werden.[8] Dabei steigt der Aufnahme- und Datenaufwand mit dem Detailgrad an. Es wird jedoch nie exakt der Klang des Originalinstruments nachempfunden werden können, da verschiedene Zufallsfaktoren, wie das Mitschwingen anderer Saiten bei Saiteninstrumenten oder bei einem Flügel nicht abgebildet werden können.[9]

ABBILDUNG 3 *AKAI S900* HARDWARE SAMPLER[10]

Vorteile von Sampling sind die Klangbearbeitungsmöglichkeiten, die moderne Sampler bieten. Bei einem Sampler handelt es sich um ein elektronisches Musikinstrument oder ein Softwareinstrument, welches für die Verwaltung der Samples und deren Ansteuerung zuständig ist. Moderne Software Sampler, wie *Native Instrument Kontakt 5* oder *Steinberg*

[8] Vgl. Raffaseder, 2010, S. 224
[9] Vgl. Müller P. , 2014, S. 32
[10] Vgl. The Fourth Man, 2014

Halion bieten eine Vielzahl an Möglichkeiten den Klang eines Samples zu formen und zu modulieren. Dadurch können neue Klänge erzeugt werden, die das reale Instrument niemals wiedergeben könnte.

2.3.2 Granularsynthese

Bei der Granularsynthese wird ein beliebiges Audiosignal in wenige Millisekunden lange Stücke, sogenannte Grains, zerteilt. Die Grains werden danach in einer beliebigen Reihenfolge neu angeordnet und ineinander überblendet, um Störartefakte zu vermeiden. Der Vorteil gegenüber dem Sampling Verfahren liegt darin, dass die Tonhöhe und die Wiedergabegeschwindigkeit entkoppelt sind. Durch kontinuierliches Wiederholen eines einzelnen Grains kann dieser beliebig lang wiedergeben werden. Zudem ist es möglich, die Wiedergaberichtung umzukehren, wodurch neuartige Klänge erzeugt werden können. Die wichtigsten Steuerparameter der Granularsynthese sind die Startposition der einzelnen Grains, deren Länge und wie diese ineinander übergeblendet werden. Maßgeblich für den Klang ist zudem die Abspielgeschwindigkeit der Grains. Die Granularsynthese eignet sich primär für Timestretching- und Pitch-Shifting-Anwendungen, aber auch zur experimentellen Klangerzeugung.[11]

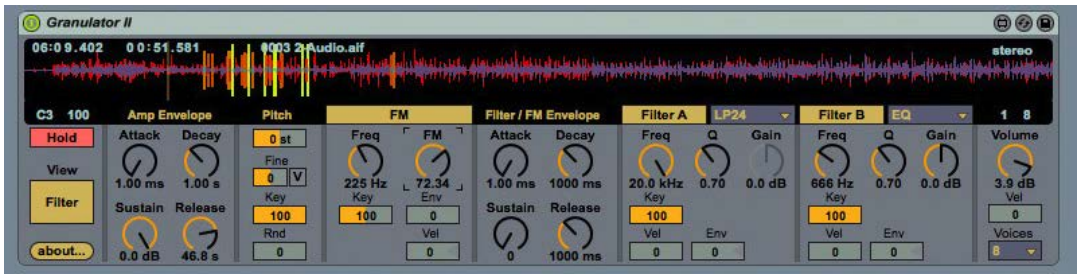

ABBILDUNG 4 *GRANULATOR II* IN *ABLETON LIVE 9*[12]

Die Granularsynthese kann und wird außerdem zur Erzeugung von Motorklängen verwendet. *LeSound AudioMotors Pro* und *Crankcase Audio REV* sind dabei die meistverbreiteten Softwarelösungen. Beide Lösungen bieten eine Integration für die Audio Engines *FMOD Studio* oder *Wwise*. Ihr grundlegendes Funktionsprinzip ähnelt sich sehr.

[11] Vgl. Raffaseder, 2010, S. 226
[12] Vgl. Müller P. , 2014, S. 30

Als Grundlage dafür dient ein aufgenommener Motorhochlauf, welcher von der Software in seinem Spektrum auf die Motorordnungen hin analysiert wird. Über diese lässt sich das Drehzahlband genau bestimmen. Mit Hilfe der Granularsynthese ist es nun möglich, eine beliebige Drehzahl innerhalb des Drehzahlbandes beliebig lange wiederzugeben.[13] [14] Somit lässt sich mit geringem Dateiaufwand ein authentischer Motorklang erzeugen.

ABBILDUNG 5 LESOUND AUDIOMOTORS V3 PRO[15]

2.4 CAN Bus

Das *Controller Area Network* ist ein Bussystem, welches in der Automobilindustrie zur Vernetzung von Steuergeräten dient. Bei einem Bussystem teilen sich alle Teilnehmer des Netzwerkes eine gemeinsame Datenleitung, um über diese zu kommunizieren. Der CAN-Bus wurde 1983 von Bosch und Intel entwickelt. Ziel war es, die zunehmende Länge der Kabelbäume in Fahrzeugen zu reduzieren und somit Gewicht einzusparen. CAN ist nach ISO 11898-2 (Highspeed-CAN) und ISO 11898-3 (Lowspeed-CAN) international standardisiert, jedoch sind beide Realisierungen nicht miteinander kompatibel.[16] [17]

[13] Vgl. AudioGaming , 2016
[14] Vgl. Crankcase Audio, 2016
[15] Vgl. AudioGaming , 2016
[16] Vgl. Wikipedia, 2016
[17] Vgl. ME-Meßsysteme GmbH, 2016

Physikalisch wird der CAN-Bus über einen Zweidrahtbus realisiert, bei dem wahlweise Kupferleitungen oder Lichtwellenleiter verwendet werden können. Je nach Treiber sind bis zu 127 Teilnehmer in einem Netzwerk möglich. Da der Bus ein Multi-Master-System ist, sind alle Teilnehmer gleichberechtigt und können gleichermaßen senden und empfangen. Um Datenkollisionen auf dem Datenbus zu vermeiden wird das CSMA/CR Verfahren eingesetzt.[18]

Der CAN Bus arbeitet objektorientiert. Das heißt, dass alle versendeten Nachrichten an alle Netzwerkteilnehmer gesendet werden. Anhand des Object-Identifier bestimmt jeder Teilnehmer, ob die empfangene Nachricht für ihn relevant ist. Ist das nicht der Fall, wird diese verworfen. Damit das funktioniert, darf der Object-Identifier nur von einem Sender im Netzwerk benutzt und versendet werden. Über die Object-ID können die Nachrichten zudem priorisiert werden. Je kleiner die ID, desto höher ist deren Priorität.[19] [20]

Eine CAN-Nachricht wird auch als Frame bezeichnet. Es gibt vier verschiedene Arten von Frames: Das Remote-Frame dient zur Anforderung von Daten, das Error-Frame wird zum Senden von Fehlermeldungen benutzt. Das Overload-Frame wird als Puffernachricht gesendet, wenn der Eingangspuffer des Empfängers voll ist. Das Daten-Frame wird zur Datenübermittlung versendet. Von diesem gibt es zwei Varianten, das Standard Daten-Frame und das Extended Daten-Frame. Diese unterscheiden sich darin, wie viel Bit für den Object-Identifier zur Verfügung stehen. Das Standard Frame besitzt 11 Bit, das Extended Frame 29 Bit. Dadurch ist die Adressierung von wesentlich mehr Objekten möglich.[21] [22]

[18] Vgl. elektro-archiv.de, 2016
[19] Vgl. ME-Meßsysteme GmbH, 2016
[20] Vgl. elektro-archiv.de, 2016
[21] Vgl. CAN in Automation (CiA), 2016
[22] Vgl. elektro-archiv.de, 2016

Ein CAN Daten-Frame ist wie folgt aufgebaut:

KOMPONENTE	LÄNGE	BESCHREIBUNG
Start of Frame Bit	1 Bit	Kennzeichnet eine neue Nachricht
Object-Identifier	11/29 Bit	Adressierung und Priorisierung der Nachricht
RTR-Bit	1 Bit	Remote Flag, Aufforderung, Daten zu senden
Control Field	6 Bit	Enthält Informationen über nachfolgenden Daten
Data Field	0…64 Bit	Eigentliche Nutzdaten
CRC Field	16 Bit	Prüfsumme zur Fehlerüberprüfung
Acknowledgement Field	2 Bit	Bestätigt korrekt empfangene Nachricht
End-of-Frame Field	7 Bit	Kennzeichnet Ende der Nachricht
Intermission Field	3 Bit	Erzeugt Pause zwischen zwei Nachrichten

TABELLE 1 AUFBAU EINES CAN DATEN FRAMES

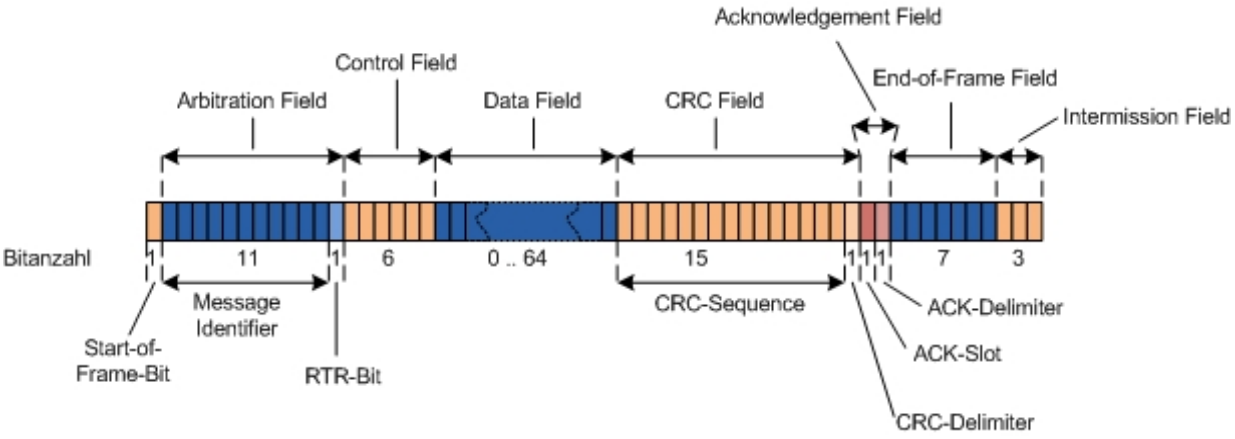

ABBILDUNG 6 AUFBAU EINES STANDARD DATEN FRAMES[23]

[23] Vgl. elektro-archiv.de, 2016

2.5 Programmierrelevante Begriffe

2.5.1 API

Bei einer API (Application-Programming-Interface) handelt es sich um eine Programmierschnittstelle. Diese dient zur Anbindung von Hardware oder Software in ein anderes Programm. Eine API wird meistens vom Hersteller angeboten und beinhaltet Bibliotheken mit Befehlssätzen. Das Programm kann die Software oder Hardware beeinflussen und steuern, ohne diese direkt ansprechen zu müssen.[24]

2.5.2 SDK

Ein Software Development Kit enthält alle Komponenten die nötig sind, um auf Basis dieser eigene Anwendungen zu entwickeln. Dazu zählen Programme, Werkzeuge, APIs und Dokumentationen der Anwendung.[25]

2.5.3 MQTT

MQTT (Message Queue Telemetry Transport) ist ein offenes Nachrichtenprotokoll. Es dient zur Übertragung von Daten zwischen Geräten, die als Nachrichten verschickt werden. Die Daten werden über eine normale Netzwerkverbindung übertragen. Das Protokoll ist sehr flexibel und eignet sich daher für die verschiedensten Anwendungen.[26]

[24] Vgl. Gründerszene, 2016
[25] Vgl. Wikipedia, 2016
[26] Vgl. Wikipedia, 2016

3. Audio Engines

Eine Videospielumgebung verändert sich ständig, wie zum Beispiel die Geräuschquelle relativ zum gespieltem Charakter, die Umgebung und viele weitere Details. Aufgrund dieser sich ständig ändernden Variablen muss sich das Sounddesign und das komplette Audio kontinuierlich anpassen. Dabei bieten Audio Engines Funktionen und Hilfsmittel zur Erzeugung dynamisch reagierender Audiosignale an.[27]

Das Arbeiten mit einer Audio Engine für die Entwicklung eines Videospiels oder einer interaktiven Audioanwendung bietet Programmierern und Sounddesignern gleichermaßen einen großen Mehrwert. Diese sind flexibel auf allen gängigen Videospielplattformen einsetzbar. Die Hersteller bieten für jede Plattform eigene SDKs und APIs an. Oftmals sind diese in den Game Engines, wie *Unity* oder *Unreal Engine* bereits implementiert, was den Arbeitsaufwand nochmals verringert. Bei Game Engines handelt es sich um das Pendant der Audio Engines, mit deren Hilfe die visuellen Inhalte erstellt werden und die Spielelogik programmiert wird.

Viele Audio Engines werden mit einer dazugehörigen Middleware angeboten, welche es dem Sounddesigner ermöglichen, das Sounddesign selbst zu implementieren. Die Middleware bietet dabei viele Funktionen, Tools und Effekte, die ihm dabei helfen, die einzelnen Sounds so natürlich und dynamisch wie möglich zu gestalten. Beispielsweise können für verschiedene Orte verschiedene Hallräume angelegt werden. Exemplarisch ändert sich der Nachhall beim Betreten einer Höhle. Der Sounddesigner kann in Rücksprache mit dem Programmierer eigene Parameter mit einem wählbaren Wertebereich definieren, die zur zusätzlichen Steuerung der Sounds dienen. Über zugewiesene Sounds und Automationen können natürliche Übergänge und Variationen der Sounds erzeugt werden. Eine Audio Engine ist in der Lage, Effekte in Echtzeit bei minimaler Ressourcennutzung zu berechnen. Das bedeutet, dass Effekte automatisiert werden können, um dynamisch auf das aktuelle Audiomaterial während des Spielens reagieren zu können.

Spieleprogrammierer müssen bei der Nutzung einer Audio Engine mit Middleware lediglich diese implementieren und die Soundevents mit der Spielelogik verknüpfen. Vom Sounddesigner werden ausschließlich die nötigen Dateien für die Audio Engine angeliefert.

[27] Vgl. Firelight Technologies., 2016, S. 17

Wird auf die Verwendung einer Audio Engine verzichtet, muss der Programmierer die Audiodateien selbst implementieren. Ein dynamisches Sounddesign ist mit dieser Methode sehr viel aufwendiger, da der Programmierer in Absprache mit dem Sounddesigner alles selbst programmieren muss. Ihm stehen nicht die Funktionen und Effekte zur Verfügung, die eine Audio Engine bietet.[28]

3.1 Auswahl der Audio Engine

Nachdem die Anforderungen für den neuen Soundgenerator definiert wurden (siehe Müller L. , 2017) und beschlossen wurde, dass dieser mit einer Audio Engine realisiert werden soll, muss nun die passende Softwarelösung gefunden werden. Gängige Audio Engines in der Gaming Branche sind *FMOD Studio* von *Firelight Technologies* und *Wwise* von *Audiokinetic*. Viele Entwicklerstudios arbeiten heute oft noch ohne Audio Engine oder benutzen Custom Lösungen in ihren Videospielen. Eine komplett selbstentwickelte „Audio Engine“ zur Erzeugung des Motorklangs schied von Anfang an aus. Der Entwicklungsaufwand wäre zu groß für den vorgegebenen Realisierungszeitraum des Projektes. Somit sollte auf eine schon vorhandene Softwarelösung zurückgegriffen werden.

Wwise und *FMOD Studio* sind die wohl am weitesten verbreiteten Audio Engines in der Gaming Branche. Beide zeichnen sich durch ihre umfangreichen Effekte und Werkzeuge aus. Zudem können Plug-Ins von Drittherstellern erworben werden, die die Engines nochmals erweitern. Beide Hersteller bieten für ihre Audio Engines APIs für alle gängigen Plattformen an, egal ob PC, Spielekonsole oder Smartphone. Ein weiterer Vorteil beider Engines ist die mitgelieferte Middleware, ein Programm, in dem die einzelnen Sounds implementiert werden können und welches die Arbeit mit den Audio Engines sehr komfortabel und einfach macht. Der Aufbau der *FMOD Studio* Middleware ähnelt dem Aufbau gängiger Digital Audio Workstations sehr, was die Einarbeitung für Sounddesigner und Tonschaffende extrem erleichtert und beschleunigt. Die Nutzung der beiden Engines ist für den geplanten Anwendungszweck, eine nichtkommerzielle Nutzung, kostenlos. Lediglich eine Projektregistrierung bei dem Hersteller ist nötig.

[28] Vgl. Dmoch, 2016

Vergleicht man beide Audio Engines im Hinblick auf den vorgesehenen Anwendungszweck, dem Erzeugen eines fahrdynamischen Motorsounds, fällt auf, dass *FMOD Studio* dafür die besseren Möglichkeiten bietet. Die Klangerzeugung ist einerseits über klassisches Sampling oder durch Kauf eines Dritthersteller Plug-Ins möglich, welches mit einem Granularsynthese-Verfahren arbeitet. In *Wwise* ist die authentische Motorklangerzeugung nur über ein Dritthersteller Plug-In möglich, welches ebenfalls mit einem Granularsynthese-Verfahren arbeitet. Generell bietet *FMOD Studio* für diesen Anwendungszweck die besseren Implementierungsmöglichkeiten als *Wwise*. Die Middleware bietet eine kleine Zusatzanwendung namens „Engine Designer“, mit welcher ein Gaspedal simuliert wird. Durch Änderung des Gaspedalreglers werden Drehzahl- und Motorlastparameter realistisch verändert. So kann schon in der Middleware das Klangverhalten des Motorsounds optimiert werden. Betrachtet man die in den letzten Jahren erschienenen großen Rennsimulationen und -spiele, wie *Assetto Corsa*, *Raceroom Racing Experience*, *Project Cars* und die *Forza Horizon* Reihe, fällt auf, dass, bis auf *Raceroom*, alle Spiele *FMOD Studio* als Audio Engine verwenden. Somit wurde letztendlich *FMOD Studio* als Audio Engine für den Soundgenerator ausgewählt, da diese sich für den vorgesehenen Einsatzzweck besser eignet.

3.2 Vergleich Audio Engine und Sampler

In nachfolgender Tabelle sollen zwei Klangerzeuger gegenübergestellt werden. Es sollen deren Funktionen und Möglichkeiten speziell zur Erzeugung von authentischen Motorensounds verglichen werden. Der Vergleich bezieht sich auf den geplanten, beziehungsweise aktuellen Einsatz am Elektro-Quad. Es soll zudem die Umsetzbarkeit der neuen Anforderungen an den Soundgenerator überprüft werden.

	FMOD Studio	Halion 5
Anwendung	Games	Musikinstrument
Ansteuerung	API, selbst programmierbar	Midi-fähige Hardware
Motorsound Erzeugung	Sampling und. Pitch Shifting, Granularsynthese (Plug-In)	Sampling
Steuerbare Parameter	frei definierbar	Notenskala und Parameter des Midi-Protokolls
Lastdynamik	überblenden von Leerlauf- und Volllastsamples	nicht möglich
Motor An/Aus Sound	möglich	möglich
Fehlzündungen	möglich	nicht möglich
Umsetzung Systemsounds (z.B. Blinker)	uneingeschränkt möglich	begrenzt möglich
Datenaufwand pro Motorsound	Ca. 2 MB	Ca. 300 MB (ohne Lastdynamik)
Plattform	Windows, Mac OS, Spielekonsolen und Smartphones	Windows und Mac OS
Preis	Indie-Lizenz kostenfrei	349€ UVP., schon vorhanden

TABELLE 2 VERGLEICH SAMPLER UND AUDIO ENGINE

4. Praktische Umsetzung

In folgendem Kapitel soll die komplette softwareseitige Umsetzung des Soundgenerators beschrieben werden. Dieses Kapitel dient zudem der Dokumentation des Funktionsumfangs und der prinzipiellen Funktionsweise. Die dazugehörige Konzeption des neuen Soundgenerators wird in (Müller L. , 2017) beschrieben.

4.1 Programmierung Soundgenerator

Der Soundgenerator besteht aus zwei Programmteilen, dem „CarPC“ und dem eigentlichen Programm „Soundgenerator“. Beide Programmteile wurden in der Entwicklungsumgebung Visual Studio 2015 von Microsoft programmiert. Als Programmiersprache wurde C# verwendet.

4.1.1 CarPC

Beim CarPc handelt es sich um firmeninterne entwickelte Software, welche für diese Anwendung angepasst wurde. Diese dient dazu, die ankommenden CAN Nachrichten aus dem CAN Interface auszulesen und diese auf Relevanz zu filtern. Wird eine relevante Nachricht registriert, so werden die darin befindlichen relevanten Daten gespeichert und an den Soundgenerator übertragen. Welche CAN Nachricht und welche Werte darin relevant sind, kann über eine Konfigurationsdatei bestimmt werden. Die Werte werden in Variablen zwischengespeichert und mittels MQTT Protokoll an den Soundgenerator übermittelt.

4.1.2 Soundgenerator

Das selbstentwickelte Programm Soundgenerator besteht primär aus fünf selbst programmierten Klassen, welche in drei logische Kategorien aufgeteilt sind. Die Klasse „Program“ ist für die Startinitialisierung und für die Weitergabe der vom CarPC empfangenen Daten zuständig. In den Klassen „ControlSound“, „SystemSound“ und „EngineSound“ werden die empfangenen Daten für die Ansteuerung von *FMOD Studio* ausgewertet und angepasst. Die Klasse „AudioEngine“ ist für die Ansprache und Steuerung von *FMOD Studio* zuständig.

Für die Einbindung der *FMOD Studio* API müssen zunächst einige Klassen und Bibliotheken in die Entwicklungsumgebung eingebunden werden. Wichtig ist, dass die Engine aus zwei Teilen besteht, zum einen das „Low Level“ System mit eigener API und dem „Studio“ System mit eigener API. Das „Low Level“ System stellt dabei einfache Funktionen zur Tonwiedergabe zur Verfügung und kann auch alleine genutzt werden. Beim Erstellen eines „Studio“ Systems wird

automatisch ein „Low Level“ System erzeugt. Damit ist der volle Funktionsumfang der Audio Engine nutzbar. Für die Implementierung von *FMOD Studio* bietet *Firelight Technologies* auf ihrer Webseite eine ausführliche Dokumentation an. Zudem werden regelmäßig Updates veröffentlicht, welche Fehler beheben und der Audio Engine neue Funktionen hinzufügen.

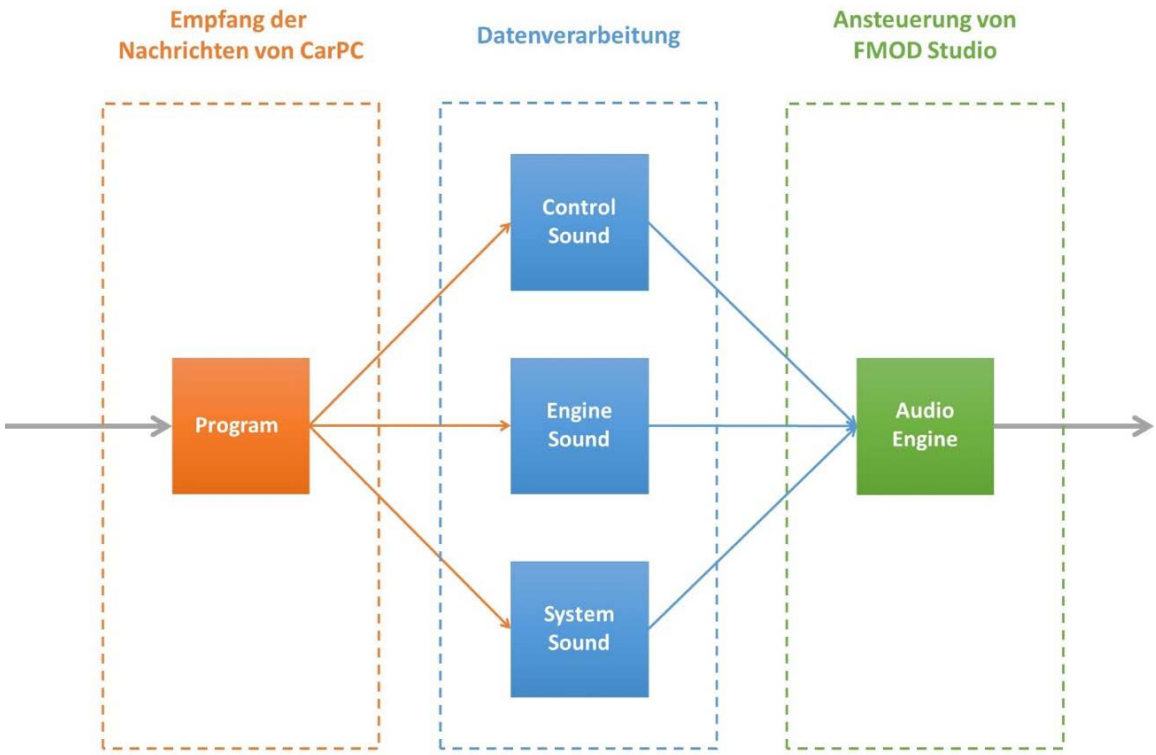

ABBILDUNG 7 KLASSENDIAGRAMM DES SOUNDGENERATORS

Bei der Klasse „Program“ handelt es sich um die Hauptklasse. In dieser wird beim Start des Konsolenprogramms der komplette Soundgenerator initialisiert. Bei der Initialisierung werden alle Objekte erzeugt, alle Motorsounds registriert und der erste davon, in diesem Fall der Motorsound des *Porsche Panamera GTS*, geladen und abgespielt.

Des Weiteren befindet sich in dieser Klasse folgendes Event:

```
PRIVATE STATIC VOID DATAVIEWMODEL_PROPERTYCHANGED(OBJECT SENDER, PROPERTYCHANGEDEVENTARGS EVENTARGS),
```

welches vom MQTT-Receiver jedes Mal aufgerufen wird, sobald sich ein empfangener Wert geändert hat. In diesem Event wird über eine Switch-Anweisung herausgefunden, welcher Wert sich geändert hat und dieser wird dann entsprechend weitergeben. Bei jedem Aufruf dieses Events wird am Ende die update-Methode der Audio Engine aufgerufen, damit alle Änderungen wirksam akustisch umgesetzt werden. Die Methode ist ursprünglich dazu

gedacht, in einer Spieleumgebung bei jedem neu angezeigten Frame aufgerufen zu werden, um kontinuierlich die geänderten Steuerdaten umzusetzen. In dieser Anwendung ist dies nicht möglich und muss wie beschrieben gelöst werden. Das Event wird aber zur Laufzeit in Intervallen von bis zu wenigen Millisekunden aufgerufen. Somit ergeben sich daraus keine Probleme, wie zum Beispiel Sprünge oder träges Reaktionsverhalten.

Name	Botschaft	Startbit	Läng...	Byteanordnung	Wertetyp
APP_sound_emerg_stop	APP_Request	21	1	Intel	Unsigned
APP_sound_mute	APP_Request	22	1	Intel	Unsigned
APP_sound_program	APP_Request	16	5	Intel	Unsigned
APP_sound_start_stop	APP_Request	23	1	Intel	Unsigned
APP_sound_volume	APP_Request	2	6	Intel	Unsigned
CANtronic_indicator	CANtronic_Set_LEDs	20	1	Intel	Unsigned
CANtronic_warning_lights	CANtronic_Set_LEDs	21	1	Intel	Unsigned
DISPLAY_button1_active_long	DISPLAY_Buttons	1	1	Intel	Unsigned
DISPLAY_button2_active_long	DISPLAY_Buttons	3	1	Intel	Unsigned
DISPLAY_button3_active_long	DISPLAY_Buttons	5	1	Intel	Unsigned
PS1FL_actual_motor_speed	PS1FL_DriveData_FL	0	16	Intel	Unsigned
PS1FL_actual_motor_torque	PS1FL_DriveData_FL	16	16	Intel	Signed
PS2FR_actual_motor_speed	PS2FR_DriveData_FR	0	16	Intel	Unsigned
PS2FR_actual_motor_torque	PS2FR_DriveData_FR	16	16	Intel	Signed
PS3RL_actual_motor_speed	PS3RL_DriveData_RL	0	16	Intel	Unsigned
PS3RL_actual_motor_torque	PS3RL_DriveData_RL	16	16	Intel	Signed
PS4RR_actual_motor_speed	PS4RR_DriveData_RR	0	16	Intel	Unsigned
PS4RR_actual_motor_torque	PS4RR_DriveData_RR	16	16	Intel	Signed
VECU_atv_driving_mode	VECU_AtvStates	0	8	Intel	Unsigned
VECU_ignition_active	VECU_SW_Diagn_1	6	1	Intel	Unsigned
VECU_set_motor_torque_FL	VECU_TorqInstr_FL	0	16	Intel	Signed
VECU_set_motor_torque_FR	VECU_TorqInstr_FR	0	16	Intel	Signed
VECU_set_motor_torque_RL	VECU_TorqInstr_RL	0	16	Intel	Signed
VECU_set_motor_torque_RR	VECU_TorqInstr_RR	0	16	Intel	Signed
VECU_throttle_position	VECU_SW_Diagn_2	0	8	Intel	Unsigned

TABELLE 3 LISTE ALLER VERWENDETEN CAN NACHRICHTEN UND SIGNALE

Die Klasse „AudioEngine“ ist für die Anbindung von *FMOD Studio* verantwortlich. In dieser Klasse werden die Systeme der Audio Engine generiert und initialisiert. Bei der Initialisierung wird das Lautsprecherausgabeformat definiert, in diesem Fall eine 5.0 Surround Anordnung. Zudem ist das Liveupdate und der Profiler aktiviert, um später, während der Laufzeit des Soundgenerators noch Änderungen an den Sounds in der Middleware vornehmen zu können. Der Profiler wird dazu genutzt, Testfahrten aufzuzeichnen. Dabei werden alle Aktivitäten der Audio Engine aufgezeichnet. Die Sounds können im Nachhinein angepasst und die Fahrt virtuell mit Hilfe der Daten nachgefahren werden. Somit müssen die Motorsounds nicht

zwangsweise am Quad optimiert werden. Dies vereinfacht die Implementierung und Optimierung der Sounds immens.

Neben der Initialisierung werden in dieser Klasse auch sämtliche Methoden zur Verfügung gestellt, welche die Engine steuern. Neben den Methoden zum Laden und Entladen der Bänke und Events werden auch Methoden zur Ansteuerung der Parameter und der Lautstärke bereitgestellt, die von den Datenverarbeitungsklassen genutzt werden können.

Die empfangenen Daten werden an die jeweiligen Datenverarbeitungsklassen weitergegeben. Zu erwähnen ist, dass alle Objekte als Singleton generiert werden. Das bedeutet, zur Laufzeit gibt es von jedem Objekt genau eines. Die Datenverarbeitung ist aus Gründen der Übersichtlichkeit auf drei Klassen aufgeteilt. Die Klasse „SystemSound“ ist für die Systemklänge zuständig. Dazu gehören Blinker- und Warnblinker-Sound sowie das Piepsen bei der Rückwärtsfahrt. Bei der Initialisierung werden die notwendigen Events der Audio Engine dafür geladen. Bei dem Blinker und dem Warnblinker handelt es sich um das gleiche Event in *FMOD Studio*. Wenn der Blinker oder die Warnleuchten aktiviert wurden, kommt als CAN Nachricht ein entsprechendes Signal an, welches zwischen 1 und 0 hin- und herwechselt. Dieses Signal wird verwendet um einen Parameter im Event zu steuern, so dass, synchron zur Blinkerleuchte, das klassische Relaisschalten akustisch wiedergegeben wird. Wenn der Blinker inaktiv ist, ist das Signal durchgehend null und kein Sound wird abgespielt. Das Rückfahrpiepsen wird nur abgespielt, wenn der entsprechende Fahrmodus gewählt ist.

Die Klasse „ControlSound“ ist primär für sämtliche Steuerung der Audio Engine zuständig, die nichts mit der Wiedergabe einzelner Sounds zu tun hat. Darunter zählt die Lautstärkesteuerung und das Beenden des Programms beim Abschalten des Quads.

Der Soundgenerator kann während der Nutzung des Quads entweder mit einer firmenintern entwickelten App oder rudimentär über drei Knöpfe an der Seite des Informationsdisplays am Quad gesteuert werden. Über die App kann die Lautstärke eingestellt und beliebig zwischen einzelnen Motorensounds gewählt werden. Am Quad selbst kann lediglich der Motorsound gestartet, gestoppt oder stummgeschalten und nach einer festen Reihenfolge in einer Endlosschleife zwischen die Motorsounds gewechselt werden.

Die Klasse „EngineSound“ ist das Herzstück des Soundgenerators. Diese ist für die komplette Steuerung der Motorensounds verantwortlich. Sie ist für die Berechnung und Steuerung der Drehzahl- und Lastparameter zuständig. Diese werden je nach gewähltem Fahrmodus unterschiedlich berechnet. Sie verwaltet zudem sämtliche Motorsounds und ist für deren Wechsel über die App oder per entsprechenden Knopfdruck zuständig.

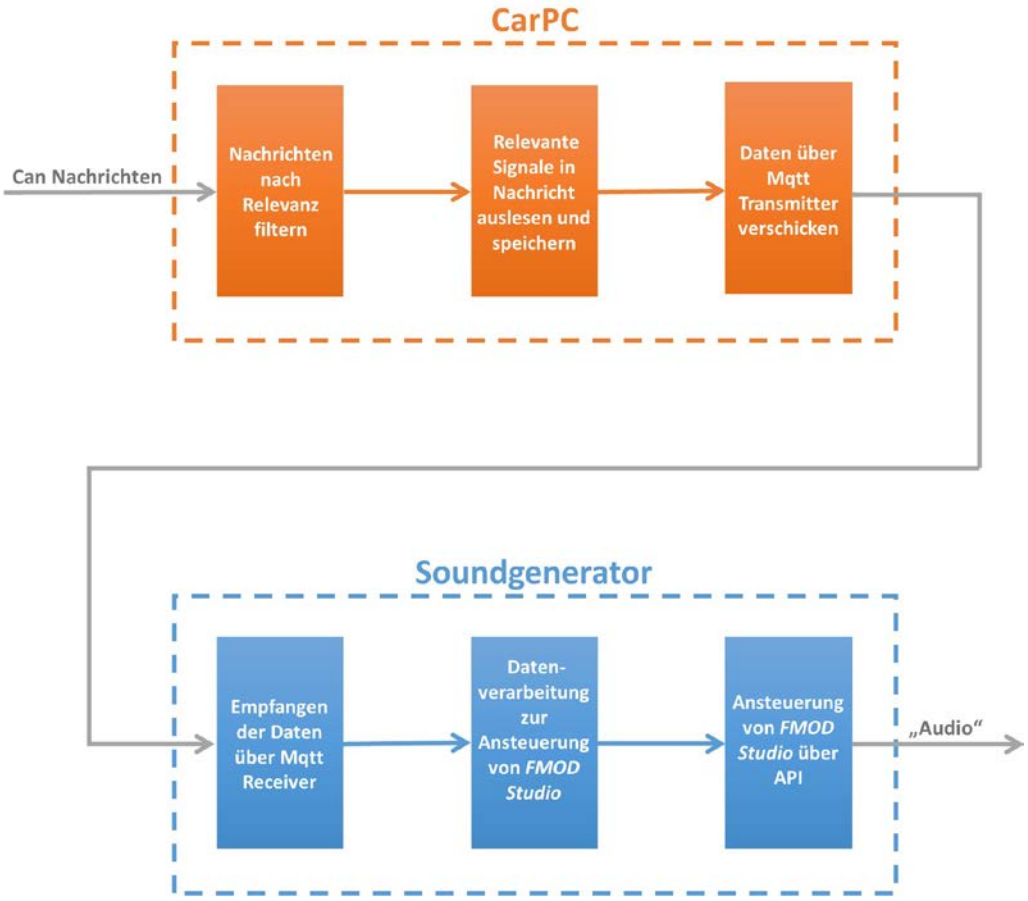

ABBILDUNG 8 SIGNALFLUSSDIAGRAM DES GESAMTEN SOUNDGENERATORS

4.1.3 Berechnung des Drehzahl- und Lastparameters

Die Berechnung der Parameter ist essentiell für die Ansteuerung der *FMOD Studio* Events. Die Authentizität des Soundgenerators ist von deren realitätsgetreuen Ansteuerung maßgeblich abhängig.

Die Drehzahl wird im Stand und in der Fahrt unterschiedlich errechnet. Bei der Fahrt muss der Drehzahlbereich der aufgenommenen Motorsounds dem Drehzahlbereich der Elektromotoren angepasst werden. Der nötige Faktor wird anhand folgender Formel berechnet:

$$RPM = RPM_{min} + \frac{RPM_{diff} * QuadRPM_{mean}}{QuadRPM_{max}},$$

wobei $QuadRPM_{mean} = \sum_{i=1}^{m} \frac{QuadRPM_i}{m}$ und $RPM_{diff} = RPM_{max} - RPM_{min}$.

RPM_{min} und RPM_{max} sind dabei der Minimal- und Maximalwert des Drehzahlparameters in *FMOD Studio*. $QuadRPM_{mean}$ ist der Mittelwert aller aktuellen Drehzahlen der vier Elektromotoren am Quad. Dieser Wert wird schon direkt beim Empfangen der Daten vom CarPc errechnet und übergeben. $QuadRPM_{max}$ ist die maximale Drehzahl der Elektromotoren. Diese beträgt maximal 2880 RPM. Der somit errechnete Faktor passt die beiden Drehzahlbänder zueinander an. Wird ein anderer Motorsound gewählt wird dieser Faktor neu berechnet.

Im Stand berechnet sich die Drehzahl wie folgt:

$$RPM = RPM_{min} + \frac{RPM_{diff} * QuadThrottle}{QuadThrottle_{max}},$$

wobei $RPM_{diff} = RPM_{max} - RPM_{min}$.

Der Unterschied zur Berechnung in der Fahrt liegt darin, dass nicht die Drehzahlen der Elektromotoren verwendet werden können, da diese im Stand immer Null betragen. Anstatt dieser wird die Drehzahl des Motorsounds mit der Handgasstellung gesteuert. Die Auflösung des Signals beträgt 8 Bit, welches einem Wertebereich von 0 bis 254 entspricht. Mit Errechnung des entsprechenden Faktors kann damit ebenfalls der jeweilige Motorsound authentisch wiedergegeben werden.

Die Berechnung des Lastparameters ist nur während der Fahrt notwendig. Im Stand wird dieser fest auf Null gesetzt, da im Leerlauf keine Last vorhanden ist. In der Fahrt wird die Last nach folgender Formel berechnet:

$$Load = \frac{\vec{M}_{target_{mean}} + \vec{M}_{actual_{mean}}}{2} + Offset,$$

wobei $\vec{M}_{target_{mean}} = \left| \sum_{i=1}^{m} \frac{\vec{M}_{target_i}}{m} \right|$, $\vec{M}_{actual_{mean}} = \left| \sum_{i=1}^{m} \frac{\vec{M}_{actual_i}}{m} \right|$

und $Offset = \frac{MaxOffset}{RPM_{diff}} * (RPM - RPM_{min})$.

$\vec{M}_{target_{mean}}$ und $\vec{M}_{actual_{mean}}$ sind die Mittelwerte der Soll- und Ist-Drehmomente. Diese werden, wie die Drehzahl, schon beim Empfangen der Daten vom CarPC gemittelt. Das Ist-

Drehmoment ist das aktuell anliegende Drehmoment an den Elektromotoren. Das Soll-Drehmoment ist das vom Steuergerät vorgegebene Drehmoment an die Motoren.

Der Mittelwert der beiden Mittelwerte der Drehmomente stellt vorerst ein recht authentisches Lastverhalten dar. Wichtig ist zudem, dass mit den Absolutwerten der Mittelwerte von Soll- und Ist-Drehmoment gerechnet wird, da definitionsbedingt diese negativ sind, wenn rückwärts gefahren wird. Aus Dokumentationen konnte entnommen werden, dass die Drehmomente bei der Fahrt maximal 14 Nm pro Elektromotor, beziehungsweise -14 Nm bei Rückwärtsfahrt, betragen. Dadurch ergibt sich durch die Mittelung ein Wertebereich für den Lastparameter in *FMOD Studio* von 0 bis 7.

Der Offset ist ein dynamischer Offset und ist abhängig vom errechneten Drehzahlparameter. Mit $MaxOffset$ wird der maximale Last-Offset bei der maximalen Drehzahl angegeben. In der Praxis erwies sich 1 als $MaxOffset$ für ausreichend. Der Offset verhält sich linear zur Drehzahl, das bedeutet, im Leerlauf bleibt die Last unverändert, der Offset hat keinen Einfluss. Je höher die Drehzahl steigt, desto größer wird der Offset, bis er schließlich seinen Maximalwert erreicht. Der Offset wird aus rein klangästethtischen Gründen benötigt. Er soll verhindern, dass der Lastparameter bei kontunuierlicher Fahrt der Höchstgeschwindigkeit zu sehr sinkt. Der Motorsound soll dadurch kraftvoller und „angestrengter“ klingen.

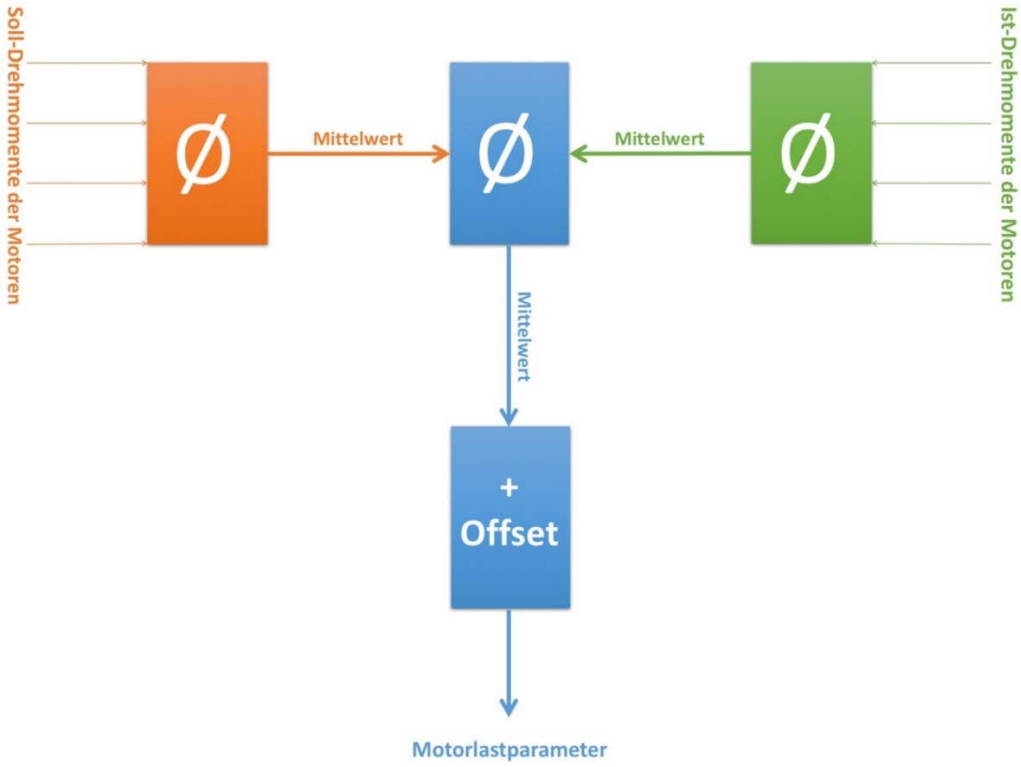

ABBILDUNG 9 BERECHNUNG DES LASTPARAMETERS

4.2 Post Produktion der Samples

In diesem Abschnitt soll ein Überblick über alle nötigen Arbeitsschritte gegeben werden, um die Aufnahmen für die Implementierung in *FMOD Studio* vorzubereiten.

4.2.1 Sichten

Nach der durchgeführten Aufnahme müssen alle Dateien gesichtet und strukturiert werden. Bei der Aufnahme wurden alle Dateien schon nach Fahrzeugen sortiert. Außerdem wurden die Samples grob benannt. Dabei wurde die Scene und Take Funktion des genutzten *Sounddevice 788t* genutzt. Unterschieden wurde in *IDLE*, *LOAD*, *ON*, *OFF*, *REF* und *BACKFIRE*. Sobald die richtige Scene geladen wurde, wird die Takenummer automatisch hochgezählt. Während der Aufnahme wurden eine Tabelle geführt, in welcher Drehzahl und spätere Funktion der aufgenommenen Takes dokumentiert wurde.[29] Bei den Aufnahmedateien handelt es sich um Multi-Wave Dateien, in denen alle aufgenommen Spuren zusammen in einer Datei gespeichert werden. Dies bietet Vorteile im Handling, da alle Signale eines Takes immer zusammenbleiben.

Für die weitere Bearbeitung ist es erforderlich alle Dateien in eine DAW (Digital Audio Workstation) zu laden, in diesem Fall *Avid Pro Tools*. Für jedes Fahrzeug wurde ein separates Projekt angelegt. Danach wurden der Anzahl der aufgenommenen Signale entsprechende temporäre Spuren angelegt, in welche die Audiodateien gezogen werden, so dass zum Schluss alle gleichen Signale auf derselben Spur liegen. Es ergibt Sinn bei den Aufnahmen des *Porsche Panameras* und beim *Audi A6* die Signale beider Abgasmündungsgeräusche auf einer Stereospur zusammenzufassen, da somit das weitere Bearbeiten vereinfacht wird. Weiterhin ist es sinnvoll die Spuren zu gruppieren, da diese immer parallel geschnitten werden müssen und in sich nicht verschoben werden sollen. Beim *Porsche Panamera* wurden zusätzlich die charakteristischen Fehlzündungen, beziehungsweise das „Blubbern“ aufgenommen, welche beim Schalten oder beim Wegnehmen des Gases entstehen. Bei diesen Aufnahmetakes sind lediglich die Signale der Abgasanlage relevant, da diese nur dort zu hören sind.

Nachdem die Spuren im Projekt angelegt wurden, können die einzelnen Spuren grob vorgeschnitten und unbrauchbare Takes gelöscht werden. Beim Schneiden soll ausschließlich der unbrauchbare Vor- und Nachlauf der Aufnahme weggeschnitten werden. Danach können

[29] Vgl. *Tabelle 1* S. 33

mit Hilfe der bei der Aufnahme erstellten Tabelle die Clips in den Spuren ihrer Funktion nach umbenannt werden. Mit der Funktion ist, wie auch schon bei der Aufnahme, der spätere Verwendungszweck gemeint. Zum Beispiel, ob es sich um ein Last- oder Leerlaufsample, Start- oder Abschaltsample oder ein Fehlzündungssample handelt. Der Syntax sieht dabei wie folgt aus:

[FUNKTION]_[NAME FAHRZEUG]_[DREHZAHL]_[POSITION]_[TAKE Nr.]

Nach der richtigen Benennung werden die einzelnen Takes ihrer Funktion entsprechend sortiert und die einzelnen Gruppen mit Markern ersichtlich gemacht. Da es bei den Leerlauf- und Lastsampleaufnahmen für jede aufgenommene Drehzahl mehrere Takes gibt, erleichtert es die weitere Arbeit, wenn die späteren Samples eines Drehzahlhochlaufes zusammengestellt werden. Die Klangfarbe der Aufnahmetakes kann merklich variieren. Es ist darauf zu achten, dass die aufeinanderfolgenden Samples klanglich zueinanderpassen, damit später ein homogen klingender Motorsound entsteht. Des Weiteren muss bei der Auswahl darauf geachtet werden, dass so wenig Störschall wie möglich, wie zum Beispiel Windgeräusche oder vorbeifahrende Autos, auf der jeweiligen Aufnahme zu hören sind und dass die Drehzahl möglichst konstant über einen längeren Zeitraum gehalten wird. Nach diesen Vorbereitungen kann mit der Bearbeitung der Aufnahmen gestartet werden.

Porsche Panamera GTS

	1st	2nd	3rd	4th
ON	ONPT02/04/05			
OFF	ONPT01/03/06			
Fehlzündungen	LOADPT2/45			
Sonstiges	LOADPT39/40HL	LOADPT42 HL	LOADPT43/44	
Begrenzer				
Fahrreferenz	REFPT01			
Offload			**Onload**	
Leerlauf				
750	IDLEPT01		LOADPT01	
1000	IDLEPT02		LOADPT02	LOADPT23
1250			LOADPT12	
1500	IDLEPT03		LOADPT03/09	LOADPT22
1750				LOADPT28
2000	IDLEPT04/05		LOADPT04/13/18	LOADPT25
2250				
2500	IDLEPT06		LOADPT05/14	LOADPT27
2750				
3000	IDLEPT07		LOADPT07	LOADPT29
3250				
3500	IDLEPT08		LOADPT08	LOADPT30
3750				
4000	IDLEPT09		LOADPT10	LOADPT31
4250				
4500	IDLEPT10		LOADPT11	LOADPT33
4750				
5000	IDLEPT11		LOADPT16HL	LOADPT34
5250				
5500	IDLEPT12		LOADPT17	LOADPT35
5750				
6000	IDLEPT13/14		LOADPT19	LOADPT35
6250				
6500	IDLEPT15		LOADPT20	LOADPT37
6750				
7000	IDLEPT16		LOADPT21	LOADPT38

TABELLE 4 DATEIZUORDNUNG *PORSCHE PANAMERA GTS*

4.2.2 Entfernen von Störschall und technische Bearbeitung

Als nächster Arbeitsschritt muss der Störschall, der sich auf den Aufnahmen befindet, minimiert werden. Dazu zählen primär Wind- und Abrollgeräusche der Reifen. Generell sind davon auch nur die Aufnahmen während des Fahrens betroffen. Auf den Signalen der sich im Motorraum angebrachten Mikrofone befindet sich sehr wenig Störschall. Sie müssen in dieser Hinsicht kaum nachbearbeitet werden. Die umgebende Karosserie und das windgeschützte Anbringen der Mikrofone hatten eine sehr saubere Aufnahme zur Folge. Der meiste Störschall befindet sich an den für die Aufnahme des Abgasmündungsgeräuschs am Heck des Fahrzeuges angebrachten Mikrofonen. Speziell bei den Aufnahmen des *Audi A6* waren durch die nicht ideale tiefe Mikrofonpositionierung teils sehr laute Windgeräusche zu hören. Zudem sind bei allen Fahrzeugen die Abrollgeräusche der Reifen auf den hinteren Mikrofonen zu hören.

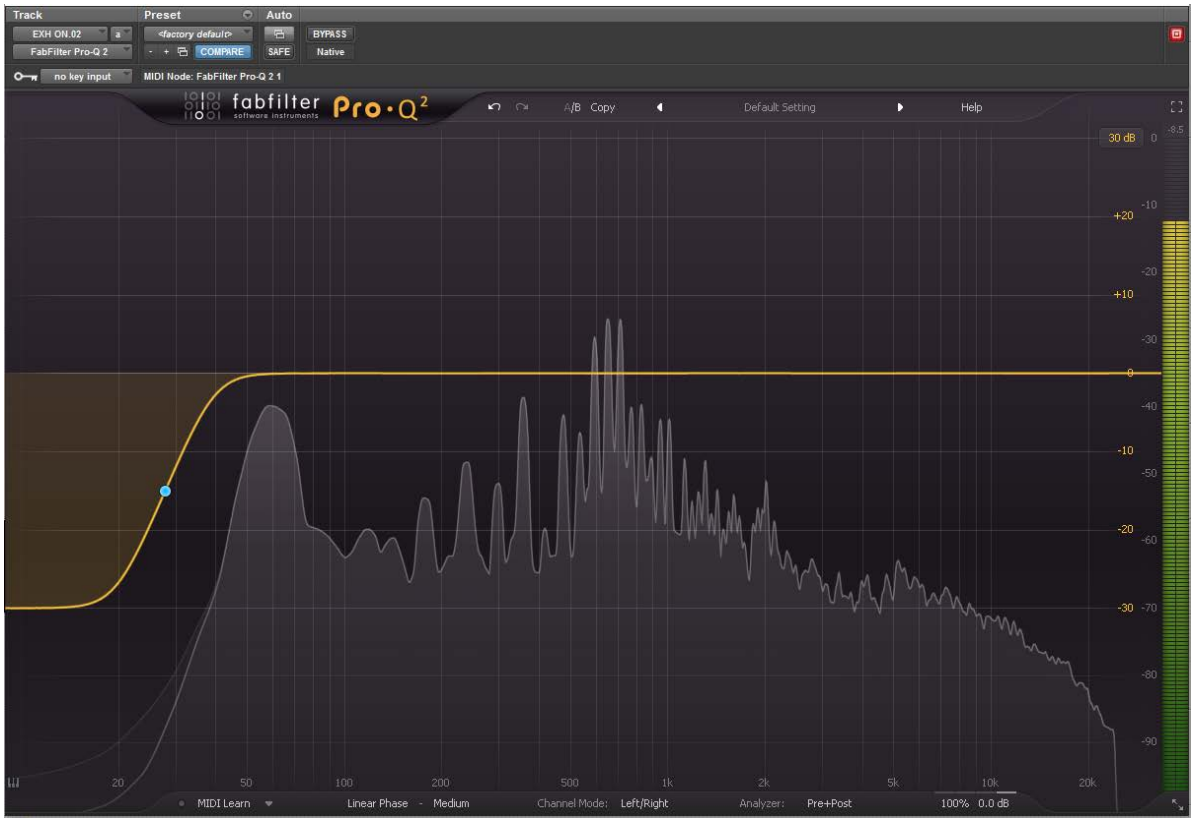

ABBILDUNG 10 *FABFILTER PRO-Q 2*

Es empfiehlt sich, auf allen Spuren die Tiefen zu beschneiden. Dabei ist darauf zu achten, dass dies unterhalb der ersten Motorordnung geschieht. Die Einstellungen des EQs sollten für jeden Aufnahmetake separat angepasst werden. In ProTools bietet es sich an, den EQ als Audio Suite Plug-In direkt auf den jeweiligen Clip anzupassen und anschließend zu rendern. Als EQ wurde der *Pro-Q 2* von *Fabfilter* verwendet. Er bietet neben vielen verschiedenen

Filtern und Einstellungen auch einen integrierten Analyser, der das Finden der richtigen Einstellung vereinfacht. Der EQ wurde im linearen Modus verwendet. Als Filter kann ein Low Cut mit recht hoher Ordnung verwendet werden oder alternativ ein Low Shelf Filter. Die Einstellungen können innerhalb eines Takes auf allen Spuren angewendet werden.

Die weiteren Bearbeitungsschritte werden in *RX 5 Advanced* von *iZotope* durchgeführt. Dies ist ein eigenständiges Programm, welches zur Audio Restauration gedacht ist. Für den Audio Transfer aus *ProTools* und zurück gibt es ein Audio Suite Plug-In. RX 5 wird ausschließlich auf den Signalen der Mikrofone angewendet, die am Heck des Fahrzeugs angebracht waren. Als erstes Modul wird mit dem *Corrective EQ das* Abrollgeräusch der Reifen abgesenkt. Dazu wird der Frequenzbereich zwischen 800 Hz und 2,5 kHz mit einem Bell Filter abgesenkt. Die Absenkung variiert je nach Sample. Bei Samples, die bei niedriger Geschwindigkeit aufgenommen wurden, ist diese kleiner als bei denen, die bei hoher Geschwindigkeit aufgenommen wurden. Zudem werden mit einem High Shelf Filter die Höhen abgesenkt, um hochfrequenten Störschall zu minimieren. Als nächstes werden die Windgeräusche mittels des *Dialog DeNoiser* weiter abgesenkt. Bei den Einstellungen ist darauf zu achten, dass das Signal nach der Bearbeitung nicht zu viele Klangartefakte enthält. Diese können vermindert werden, indem die Renderqualität im Modul auf die Bestmögliche eingestellt wird.

ABBILDUNG 11 BEDIENOBERFLÄCHE *IZOTOPE RX5 ADVANCED*[30]

[30] Vgl. iZotope, 2016

Als letzten Schritt in RX 5 wird das Modul *Deconstuct* angewandt. Dieses kann das Verhältnis von tonalem Anteil und Geräuschanteil beeinflusst werden. In diesem Fall wird der Geräuschanteil um ca. 6 dB abgesenkt.

Nach der Entfernung des Störschalls können die Aufnahmen auf die spätere Lautsprecherkonfiguration angepasst werden. Da an der Rückseite des Quads nur ein Lautsprecher verbaut ist und später die Signale am Quad der Wiedergabeposition ihrer Aufnahmeposition entsprechen sollen, können die Signale der Abgasanlage zu einem Monosignal zusammengefasst werden.

4.2.3 Klangästhetische Bearbeitung

In diesem Bearbeitungsschritt sollen die Aufnahmen in ihrem Klang optimiert werden und charakteristische Klangstrukturen der einzelnen Fahrzeuge hervorgehoben werden. Als Referenz dienen dabei die subjektiven Eindrücke, die bei der Aufnahme und dem Fahren der Fahrzeuge gesammelt wurden. Die Mikrofonpositionen entsprechen zudem nicht den Hörpositionen, in denen sonst ein Fahrzeug wahrgenommen wird, weshalb das Frequenzspektrum der Signale angepasst werden muss. Generell muss bei der klangästhetischen Nachbearbeitung das spätere Panning der einzelnen Signale beachtet werden. Ziel ist bei der späteren Wiedergabe ein homogenes und authentisches Klangbild rund um das Quad zu erreichen. Die Schallquellen sollen am Quad ihren aufgenommenen Positionen nachempfunden werden. Alle im Motorraum aufgenommenen Signale werden später nur vorne am Quad wiedergegeben und die Aufnahmen der Abgasanlage am Heck.

Viele Bearbeitungsschritte an den einzelnen Fahrzeugaufnahmen ähneln sich und verfolgen dasselbe Ziel. Lediglich die genauen Einstellungen der Plug-Ins unterscheiden sich. Diese Einstellungen müssen für die einzelnen Fahrzeugaufnahmen individuell nach Gehör und persönlichem Geschmack gewählt werden. Es soll im Weiteren ein Überblick über die verwendeten Tools gegeben werden und mit welcher Intention diese mit den gewählten Einstellungen angewandt wurden.

Die Aufnahmen der Mikrofone im Motorraum klingen alle sehr hell, da die Motorhaube und Karosserie die hohen Frequenzanteile nach außen hin sehr stark dämpfen. Mit Hilfe eines Equalizers kann dieses Dämpfungsverhalten nachempfunden werden. Prinzipiell kann jeder beliebige EQ verwendet werden, in diesem Fall wurde wieder auf den *Pro-Q 2* von *Fabfilter* zurückgegriffen. Mit einem High-Shelf Filter lassen sich die Höhen ab 2 kHz um 10 bis 20 dB

absenken. Die genauen Einstellungen sind vom jeweiligen Signal abhängig und müssen individuell gewählt werden. Des Weiteren sorgt eine leichte Absenkung bei circa 500 Hz und eine Anhebung bei 150 Hz für mehr Klarheit und Volumen. Unangenehme Störresonanzen können mit Hilfe des EQs, durch entsprechende Absenkung der betroffenen Frequenzen, entschärft werden.

Das Ansauggeräusch des Motors oder das Turboladerpfeifen werden am besten hervorgehoben, wenn auf der jeweiligen Aufnahme die Höhenabsenkung nicht ganz so stark vorgenommen wird und zudem die Tiefen mittels Low-Shelf Filter abgesenkt werden. Somit ist später das Ansaug- oder Turbogeräusch separat im Verhältnis steuerbar. Die Mikrofonsignale im Motor benötigen neben der Anpassung des Frequenzgangs keine weitere Bearbeitung. Diese Bearbeitung verändert sich für die einzelnen Samplearten (Leerlauf-, Last-, Startsamples, etc.) nicht, kleinere individuelle Anpassungen einzelner Aufnahmetakes wurden mittels Automation des EQs umgesetzt.

Bei den Auspuffsignalen unterscheidet sich die Bearbeitung zwischen den Leerlaufsamples, zu denen in diesem Fall auch die Start- und Abschaltsamples gehören, und den Lastsignalen. Grundlegend werden alle Auspuffsamples erst einmal mit einem Equalizer bearbeitet. Mit diesem sollen die charakteristischen Motorordnungen und Resonanzen der Abgasanlage betont werden. Zudem sollen Störresonanzen abgesenkt werden. Grundsätzlich werden dafür die Tiefen und Tiefmitten angehoben. Der Equalizer muss dabei für jedes einzelne Sample individuell angepasst werden. Dafür werden der Frequenzregler und der Gainregler der betroffenen Filter automatisiert. Im Bereich zwischen 500 Hz bis 1,5 kHz wird breitbandig abgesenkt, da dort, bedingt durch den Frequenzgang der Mikrofone, eine Überbetonung vorliegt, welche korrigiert werden muss. Diese ist für alle Samples gleich und muss nicht individuell angepasst werden.

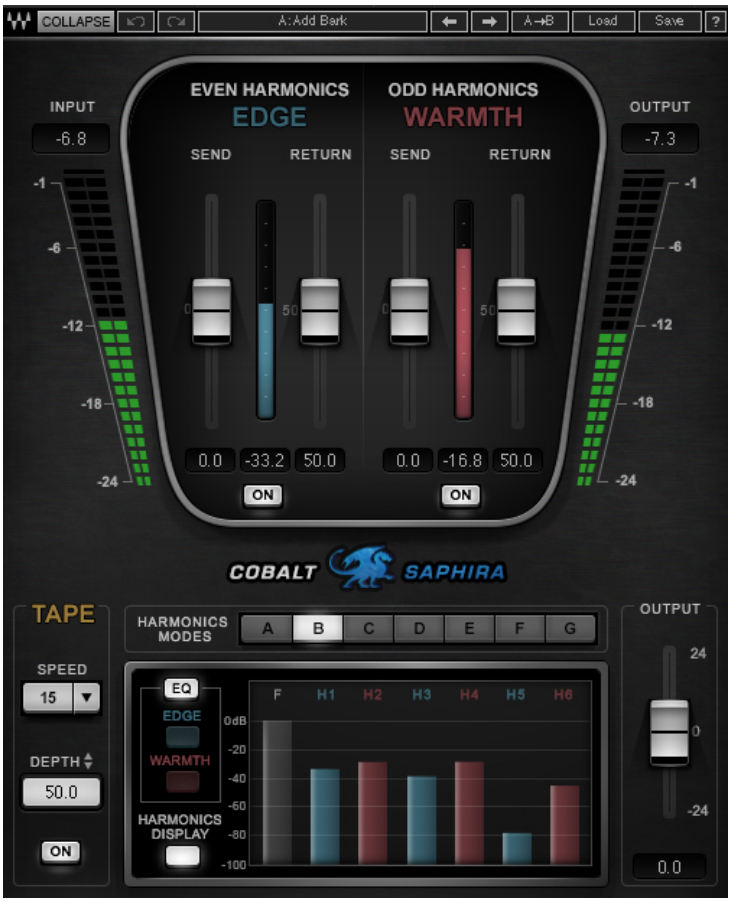

ABBILDUNG 12 *WAVES COBALD SAPHIRA*

Die Lastsamples benötigen zudem noch einen weiteren Bearbeitungsschritt. Die Aufnahmen unter Last variieren klanglich sehr stark. Bei dem verwendeten Aufnahmeverfahren konnte nicht gewährleistet werden, dass alle Samples bei annähernder Volllast aufgenommen wurden. Sie klangen, wider Erwarten, nicht kräftig und druckvoll genug. Durch Verzerren können die aufgenommene Signale den bei der Aufnahme gewonnenen Eindrücken angepasst werden. Hierfür hat sich das Plug-In *Cobald Saphira* von *Waves* als am besten geeignet herausgestellt. Die Besonderheit dieses Verzerrers ist, dass nicht nur die Obertonstruktur veränderbar ist, sondern die geraden und ungeraden Obertöne getrennt voneinander einstellbar sind. Dabei kann der Verzerrungsgrad und dessen Mischungsverhältnis individuell eingestellt werden. Wie viel Verzerrung nötig ist, hängt vom jeweiligen Fahrzeug ab. Dabei soll das Ergebnis natürlich klingen. Die Aufnahmen des *Porsche Panamera GTS* konnten dabei mehr verzerrt werden, wie die Aufnahmen des *Audi A6*. Als Ausgangspunkt bieten sich die „Harmonics Modes“ C oder D an. In der Praxis stellte sich heraus, dass durch Hinzufügen von geraden Obertönen der Auspuffklang aggressiver und sportlicher wurde. Die ungeraden Obertöne machten den Klang voller und voluminöser, jedoch auch im Gesamten undefinierter

und matschiger. Die genauen Einstellungen mussten für jedes Sample individuell angepasst werden, da die Signale sich im Pegel und der Klangfarbe unterscheiden. Der Grad der Verzerrung und die Verhältnisse der Obertöne müssen jedem Sample angepasst werden. Ziel dieses Bearbeitungsschritts ist es, nicht nur den Klang voller und sportlich wirken zu lassen, sondern auch die Unterschiede in der Klangfarbe anzupassen, damit der spätere Motorhochlauf unter Volllast in *FMOD Studio* homogen klingt.

Als letzter Schritt wird auf allen Spuren das Plug-In *Pi* von *Sound Radix* insertiert. Dabei handelt es sich um einen interaktiven Phasenmischer, welcher die einzelnen Signale dynamisch zueinander in der Phase dreht, so dass nur minimale Phasenauslöschungen entstehen. Damit wird garantiert, dass sich später keine tieffrequenten Signalanteile auslöschen und die Motorensounds drucklos und dünn klingen. Zudem wird der nachfolgende Schnitt der Samples vereinfacht. Für diese Anwendung sind die Standardeinstellungen des Plug-Ins passend und müssen nicht geändert werden. Nachdem die Bearbeitung der Samples abgeschlossen ist, müssen alle Effekte auf die Samples gerechnet werden, damit mit dem Feinschnitt begonnen werden kann.

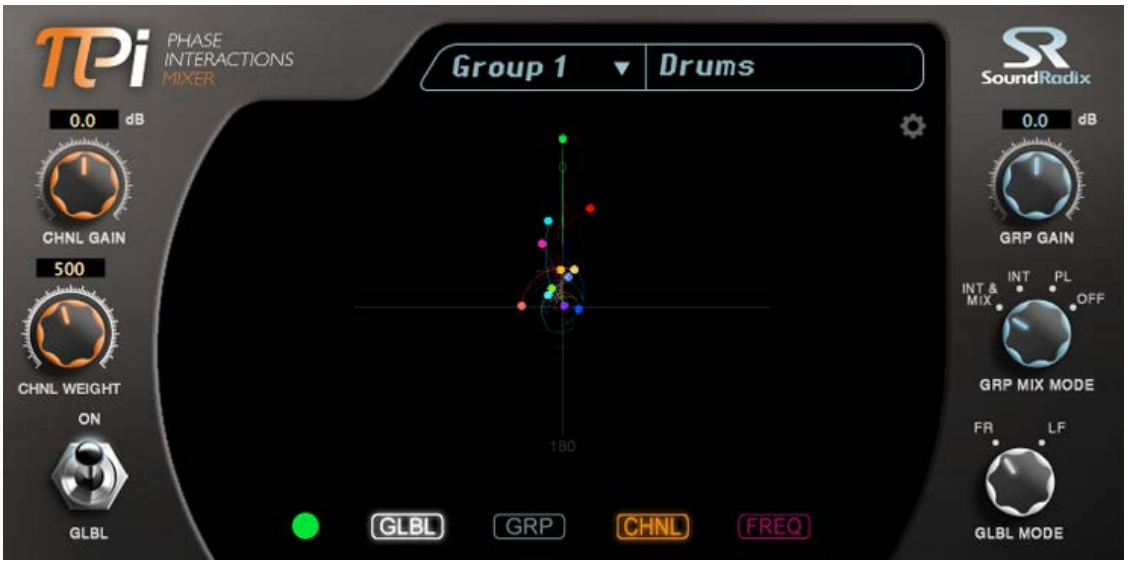

ABBILDUNG 13 *SOUNDRADIX PI*[31]

[31] Vgl. Sound Radix, 2016

4.2.4 Schnitt

Bevor die Samples in die Audio Engine implementiert werden können, müssen diese geschnitten werden. Als Ausgangsmaterial dienen der Audio Engine Samples, die kontinuierlich in einer Schleife wiedergegeben werden können, ohne das Störartefakte, wie Knacken oder Sprünge entstehen. Als Basis dafür würde theoretisch nur eine komplette Zündabfolge ausreichen. Praktisch wird aber mit Samplelängen von circa einer Sekunde gearbeitet. Dabei ist darauf zu achten, dass diese im Loop kontinuierlich weiterläuft und nicht unterbrochen wird, da sonst merkliche inhaltliche Sprünge entstehen.

Wichtig für den Schnitt ist, dass alle Aufnahmen eines Takes parallel geschnitten werden, damit diese später in *FMOD Studio* einfach angelegt werden können. Dafür bietet sich wieder die Gruppierfunktion der DAW an. Alle betroffenen Spuren werden zu einer Gruppe zusammengefasst. Somit werden alle Aktionen auf allen gruppierten Spuren ausgeführt.

Die Schwierigkeit im Schnitt besteht darin, alle Spuren so zu schneiden, das bei der Loop-Wiedergabe einerseits die inhaltliche Kontinuität gewährleistet ist, andererseits beim Übergang in keiner der einzelnen Spuren ein Knacken zu hören sind. Es empfiehlt sich anfangs durch die Nulldurchgänge der Abgasanlagenspur zu schneiden, da diese am meisten tieffrequente Signalanteile besitzt. Von diesem Ausgangspunkt an müssen die Enden der Samples so lange angepasst werden, bis kein Knacken mehr zu hören ist.

Bei dem Schnitt der Start- beziehungsweise der Abschaltsamples des Motors, muss für ein optimales Überblenden der Sounds, Vor- und Nachläufe einkalkuliert werden. Die Vorläufe der Startsamples und ebenso die Nachläufe der Abschaltsamples müssen die gleiche Länge haben. Nachdem der Schnitt abgeschlossen ist, können alle Samples exportiert und in FMOD Studio angelegt werden.

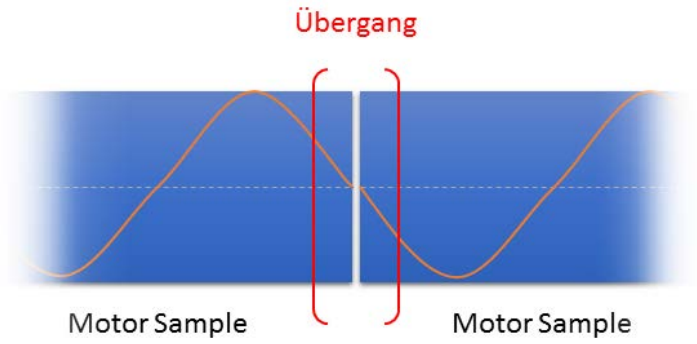

ABBILDUNG 14 OPTIMALER LOOPÜBERGANG

4.3 Projektstruktur und Implementierung in *FMOD Studio*

Nachfolgend werden alle Funktionen erklärt, die zur Motorsounderzeugung benötigt werden. Der Projektaufbau in der Middleware und der Workflow zur Implementierung neuer Sounds werden erläutert. Für das Einarbeiten in die Middleware wird vom Entwickler ein ausführliches Benutzerhandbuch bereitgestellt. Zudem werden kostenlose Videotrainings und Tutorials zur Verfügung gestellt, welche laufend erweitert werden.

4.3.1 Grundlegende Projektstruktur

Um mit der Audio Engine arbeiten zu können, muss zuerst das Grundkonzept verstanden werden. Wie Sounds implementiert und wie diese aus der Middleware an die eigentliche Anwendung übergeben werden.

FMOD Studio arbeitet eventbasiert. Die Events können in der Middleware erzeugt und verwaltet werden. In diese werden die jeweiligen Sounds implementiert. Zusätzlich können Mechanismen zur zusätzlichen Steuerung der Events integriert werden, die von der programmierten Anwendung genutzt werden können, so dass ein dynamisch agierender Klang entsteht. Um die Events an die Anwendung zu übergeben, müssen diese „Banks“ zugewiesen werden. Dabei handelt es sich um die Exportdateien, die an die Anwendung übergeben werden. Diese enthalten alle benötigten Audiodateien und die zur Steuerung nötigen Metadaten. Sie werden beim „Built“-Vorgang erstellt. Jedes Event kann nur einer Bank zugewiesen werden, jedoch können einer Bank mehrere Events zugewiesen werden. Für den Export muss es eine Master Bank geben, in welcher zusätzliche Metadaten gespeichert werden. Diese ist standardmäßig angelegt. In den Einstellungen können die Exportoptionen verändert werden. Es ist möglich, Einstellungen am Audio Codec vorzunehmen oder einen benutzerdefinierten Dateipfad für den Export anzulegen. Des Weiteren werden Export Presets für die gängigen Plattformen angeboten. Für diesen Anwendungszweck wurde das Exportpreset „Desktop“ gewählt und dessen Einstellungen beibehalten. Lediglich ein benutzerdefinierter Dateipfad wurde angelegt, so dass die Dateien direkt in das Verzeichnis des Soundgenerators gespeichert werden.

ABBILDUNG 15 BEDIENOBERFLÄCHE VON *FMOD STUDIO*

Um neue Sounds dem Soundgenerator hinzuzufügen oder vorhandene zu überarbeiten, kann das bereits vorhandene Projekt „*FMOD_Master*“ verwendet werden. In diesem sind alle zum jetzigen Zeitpunkt implementierten Sounds angelegt. Öffnet man das Projekt, wird die Bedienoberfläche wie in *Abbildung 10* angezeigt. Auf der linken Seite befindet sich der Browser, indem alle Events, Banks und Assets verwaltet werden können. Unter dem Reiter Assets werden alle Audiodateien verwaltet. Diese können per „Drag & Drop“ hinzugefügt werden. Es wird eine Kopie der Dateien im Projektverzeichnis angelegt. Auf der rechten Seite befinden sich die Properties, welche diverse Einstellmöglichkeiten bieten, jedoch bei dieser Anwendung irrelevant sind und ignoriert werden können. In der Mitte befindet sich der Event Editor. In diesem kann das jeweils ausgewählte Event bearbeitet und diesem neue Samples hinzugefügt werden. Im unteren Bereich befindet sich das sogenannte „Deck“. Dort werden zusätzliche Optionen zu den ausgewählten Objekten im Event Editor angezeigt. Wählt man zum Beispiel eine Spur aus, kann im Deck die Lautstärke oder das Panorama verändert oder Effekte insertiert werden. Wird ein Sample auf einer Spur ausgewählt, wird dort der Sampleeditor angezeigt.

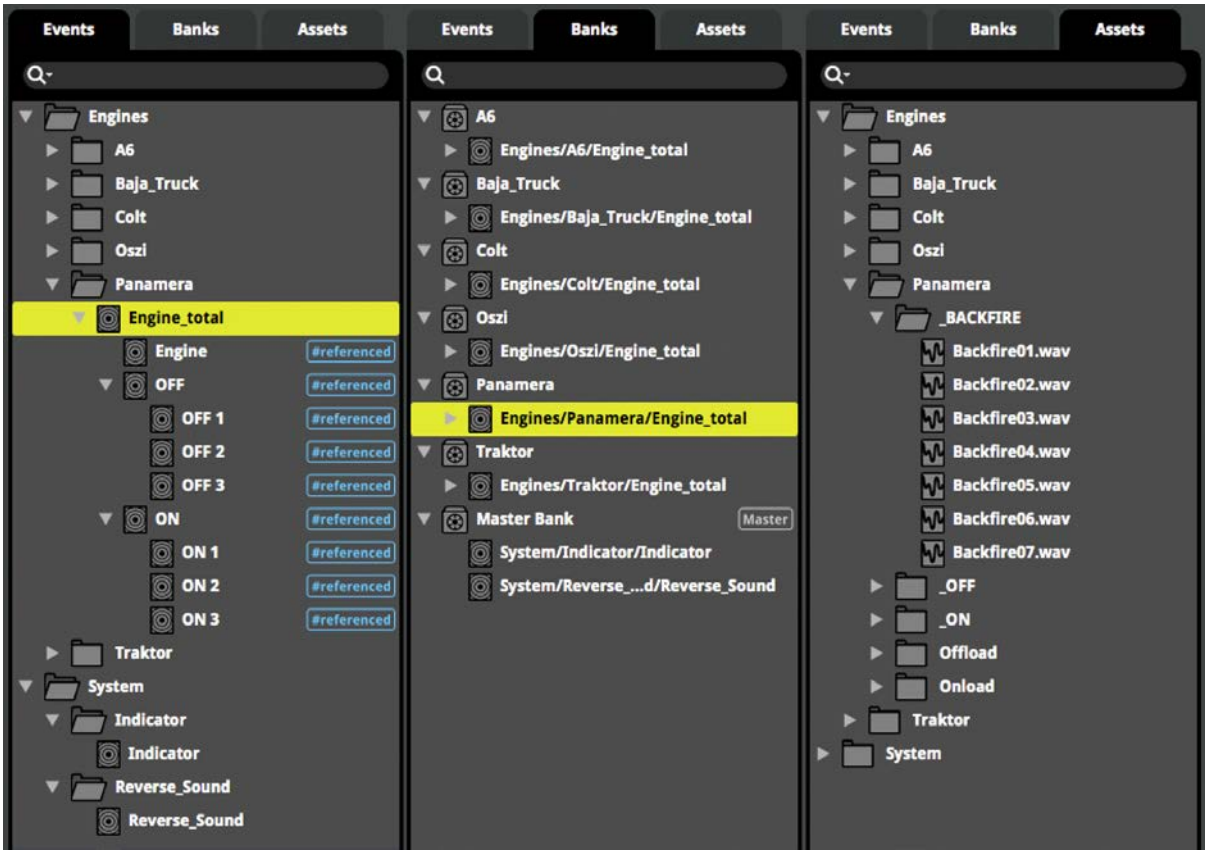

ABBILDUNG 16 ANGELEGTE STRUKTUR IM BROWSER

Beim Anlegen der Events wurde zwischen zwei Soundgruppen unterschieden. Im Ordner *Engines* werden die verschiedenen Motorsounds angelegt. Für die einzelnen Motorsounds werden wiederum Ordner mit deren Namen erstellt. In diesen werden schlussendlich die nötigen Events erzeugt. Für jeden Motorsound wird eigens eine Bank mit dem gleichen Namen des entsprechenden Ordners angelegt. Dies vereinfacht die spätere Ansteuerung der Events und die Erweiterung des Soundgenerators. Während der Benutzung des Soundgenerators wird nur der aktuell abgespielte Motorsound geladen. Wird der Sound gewechselt, werden die aktuell geladenen Daten verworfen und die entsprechende Bank des neuen Sounds geladen.

Die zweite Soundgruppe sind die Systemsounds, zu denen der Blinker und das Piepsen bei Rückwärtsfahrt gehören. Deren Events werden im Ordner „System“ angelegt, welcher wiederum Unterordner für die jeweiligen Events besitzt. Die Systemsounds werden der *Master Bank* zugewiesen. Diese ändern sich während der Laufzeit nicht und werden beim Start des Soundgenerators geladen.

Bei der Dateiverwaltung wird die Ordnerstruktur auf gleiche Weise umgesetzt. Bei den Samples für die einzelnen Motorsounds ergibt es jedoch Sinn, diese nach ihrer späteren Funktion nochmals zu sortieren. So werden im Unterordner *_ON* alle Startsounds und im Unterordner *_OFF* die Abstellsounds eines Fahrzeugs abgelegt. In den Ordnern *Onload* und *Offload* befinden sich die Last-, beziehungsweise die Leerlaufsamples. Beim *Porsche Panamera GTS* wurden zudem die Fehlzündungen gesampelt. Diese werden im Ordner *_BACKFIRE* abgespeichert.

Für die Verteilung der Sounds im Panorama wird der standardmäßige Surround Panner der Middleware genutzt. Das am Quad verwendete Lautsprecherrouting unterscheidet sich von der normalen Nutzung des Panners. Aus *Abbildung 12* können das Routing der Lautsprecher und die vorgesehenen Signale dafür entnommen werden. Näheres zur Hardware und zum Wiedergabesystem kann in (Müller L. , 2017) nachgeschlagen werden.

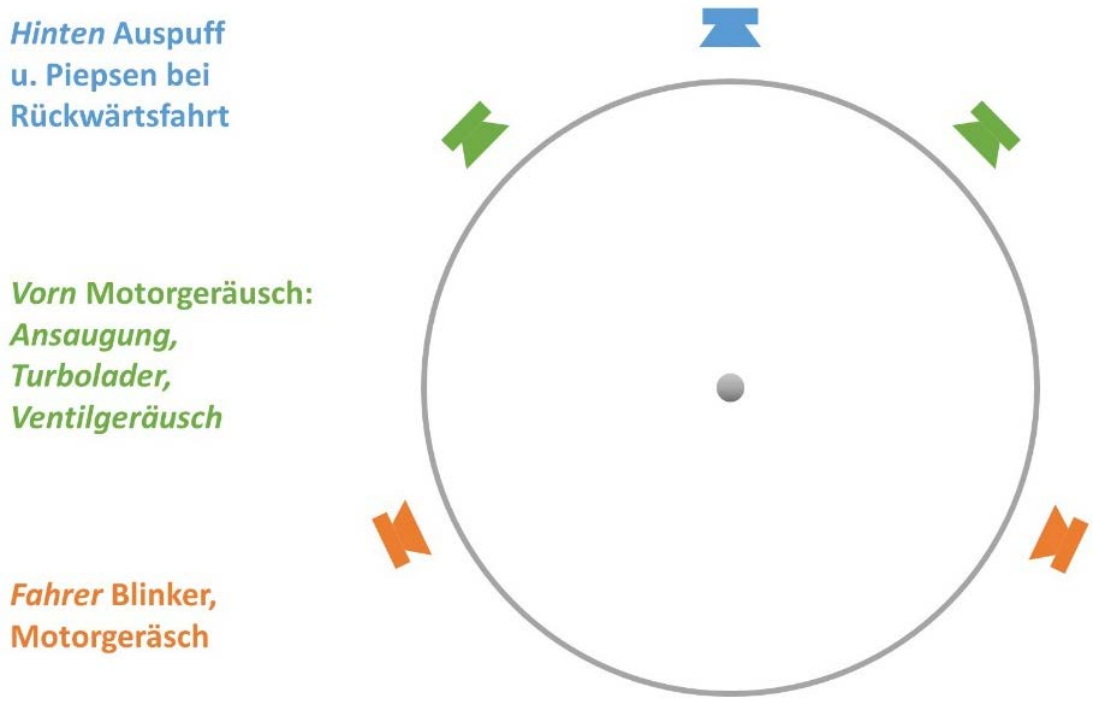

ABBILDUNG 17 VERTEILUNG DER LAUTSPRECHER IM PANNER

4.3.2 Implementierung der Systemsounds

Zunächst wird die Klangerzeugung der Systemsounds erklärt. Blinker und Warnblinker teilen sich ein Event, da derselbe Sound verwendet wird. Als Erstes wird ein neues Event mit dem Namen *Indicator* in entsprechender Ordnerstruktur erstellt. Standardmäßig enthält ein Event eine leere Audiospur und eine Masterspur. Die Standardskala ist die Timeline, welche zeitbasiert arbeitet. Auf dem Master befindet sich in einem Insertslot ein Plug-In namens *3D Panner*, welches über einen Rechtsklick auf das Plug-In unter entsprechendem Menüpunkt gelöscht werden kann. Dieses wird in dieser Anwendung nicht benötigt.

ABBILDUNG 18 BLINKER EVENT

Der klassische Blinker-Sound, ein schaltendes Relais, soll sich später synchron zum Blinken der Blinkerleuchte verhalten. Um dies zu realisieren, muss im Event ein neuer Parameter angelegt werden. Dieser wird erstellt, indem auf das Plussymbol neben dem Timeline-Reiter geklickt wird. Es öffnet sich ein Menü, in welchem Name und Wertebereich definiert werden können. Der benötigte Parameter wird mit *„indicator“* bezeichnet und bekommt den Wertebereich 0 bis 1 zugewiesen. Nach dem Erstellen wird der Parameter als Reiter neben der Standard Timeline angezeigt und im oberen Bereich des Event Editors erscheint ein Regler mit selbigem Namen, über den der Parameter gesteuert werden kann. Dieser kann später vom Soundgenerator angesteuert werden. Dafür wird im diesen Fall das ankommende CAN Signal des Blinkers oder des Warnblinkers genutzt, welches, wenn der Blinker aktiv ist, zwischen dem Wert 0 und 1 oszilliert. Die benötigten zwei Samples werden an den jeweiligen Enden des Parameters angelegt, wie in *Abbildung 13* ersichtlich. Bei den Samples handelt es sich einmal um das Öffnen und einmal um das Schließen des Relais. Das Schließen muss im Wertebereich 0.9 bis 1 angelegt werden. Die Länge der Regions auf der Parameterskala stellt nicht deren Länge dar, wie man es von der üblichen Timeline in DAWs gewohnt ist, sondern lediglich den Bereich, in welchem das Sample getriggert und wiedergegeben wird. Als letzter Schritt muss das Panorama angepasst werden. Die Ein- und Ausgangskanäle der einzelnen Spuren passen sich standardmäßig der Kanalanzahl des Audiomaterials an. Da die Blinkersamples

Monosignale sind, muss die entsprechende Spur ausgewählt werden. Im Deck klickt man mit der rechten Maustaste auf das Ausgangsmeter und stellt das Ausgangsformat auf 5.1. Automatisch stellt sich jetzt auch die Masterspur auf 5.1 Surround um. Nun kann das eigentliche Panning vorgenommen werden. Die Masterspur wird ausgewählt und im Deck wird ein Surround-Panner angezeigt. Dort werden durch Klicken alle Lautsprecher, bis auf die beiden Surroundkanäle, deaktiviert. So wird später der Blinkersound nur auf den Lautsprechern beim Fahrer zu hören sein. Das spätere Lautstärkeverhältnis muss am Quad eingestellt werden.

Für das Piepsen bei Rückwärtsfahrt wurde ebenfalls ein Event namens *Reverse_Sound* angelegt. Auf dessen Timeline wird das selbe Sample eines Sinustons in gleichen Abständen angelegt. Um diese wird durch Rechtsklick auf die schwarze Leiste unter der Zeitskala ein Loopbereich erzeugt, welcher in *Abbildung 14* als blauer Balken erkenntlich ist. Wird dieses Event gestartet, entsteht ein gleichmäßiges Piepsen, solange das Event abgespielt wird. Das Panorama wird so eingestellt, dass nur der Centerlautsprecher im Panner aktiv ist. Später wird der Sound nur aus dem hinteren Lautsprecher am Quad wiedergegeben, welcher bei Rückwärtsfahrt der Fahrtrichtung des Quads entspricht. Somit sollen Passanten vorgewarnt werden. Das Piepsen wird selbst wiedergegeben, wenn kein Motorsound gewählt oder dieser stummgeschalten ist.

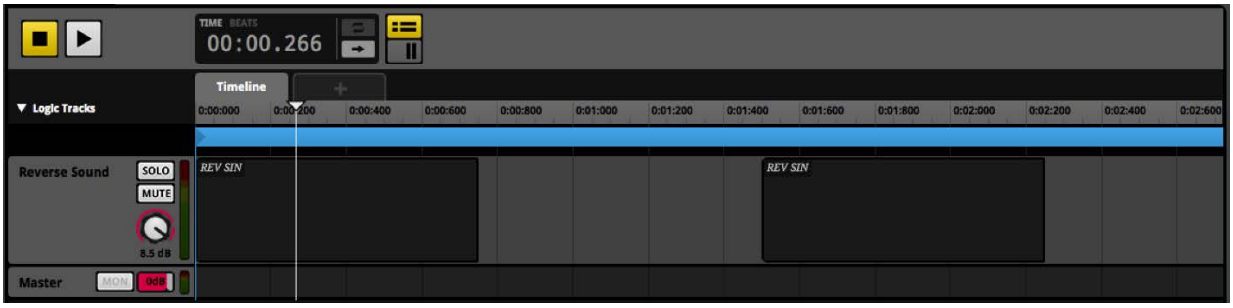

ABBILDUNG 19 RÜCKWÄRTSPIEPSEN EVENT

4.3.3 Implementierung der Motorensounds

Maßgeblichen Einfluss auf den entwickelten Implementierungsworkflow und die eigentliche Klangerzeugung hatte das SDK von der Rennsimulation *Assetto Corsa*, welches von Entwicklerstudio *Kunos Simulazioni* zur Verfügung gestellt wird. Am Beispiel der

Implementierung der Aufnahmen des *Porsche Panamera GTS* wird die Motorklangerzeugung und die genutzten Funktionen in *FMOD Studio* erläutert.

FMOD Studio bietet die Möglichkeit Events ineinander zu verschachteln. Diese *Nested Events* werden erzeugt, indem in eine Spur im Event mit der rechten Maustaste geklickt wird und im Menü das *Event Sound* Modul ausgewählt wird. Dadurch wird eine neue Region und das dazugehörige Event erzeugt. Dieses kann durch Doppelklick darauf oder über den Event Browser geöffnet werden. Enthält das verschachtelte Event Parameter, müssen diese auch im übergeordneten Event angelegt werden. Somit können diese angesteuert werden.

ABBILDUNG 20 ENGINE_TOTAL EVENT DES *PORSCHE PANAMERA GTS*

Im angelegten Ordner des jeweiligen Fahrzeugs im Event Browser wird ein Event namens *Engine_total* angelegt. Darin wird *Nested Events* erzeugt, welches mit *Engine, ON* und *OFF* benannt wird. Das *Engine* Event enthält dabei die eigentliche Motorklangerzeugung bei laufendem Motor. Mit den ON und OFF Events wird das Starten beziehungsweise Abschalten des Motors simuliert. Zunächst soll die Implementierung der Motorsamples in das *Engine* Event erklärt werden, bevor auf die restliche Implementierung eingegangen wird.

ABBILDUNG 21 ENGINE EVENT, RPM PARAMETER

Im Engine Event werden, im Fall des Panameras, insgesamt acht Spuren benötigt. Die Benennung der Spuren kann aus *Abbildung 21* entnommen werden und entspricht der Funktion und der Mikrofonposition. Die beiden verbleibenden Spuren dienen als Gruppenspuren für die On- beziehungsweise Offload Samples und werden entsprechend benannt. Um die jeweiligen Spuren in die Gruppenspuren zu routen, müssen diese markiert werden. Durch Rechtsklick auf eine ausgewählte Spur kann im Unterpunkt *Output Master* die gewünschte Gruppenspur gewählt werden. Um die Unterscheidung zu vereinfachen, können diese eingefärbt werden. Neben den Spuren werden zudem zwei Parameter benötigt, der *rpm* Parameter mit einem Wertebereich von 0 bis 7500 und der *load* Parameter mit einem Wertebereich von 0 bis 7. Auf die genaue Benennung der Parameter ist zu achten, da diese sonst später nicht vom Soundgenerator angesprochen werden können. Der Wertebereich unterscheidet sich von der Drehzahl des aufgenommenen Fahrzeuges. Dies ist funktionsbedingt nötig. Das Parameterminimum muss 0 sein, da sonst das Pitchen der Samples nicht einwandfrei funktioniert. Das Parametermaximum ist zudem größer als das gesampelte Drehzahlmaximum, da sonst das letzte Sample nicht angelegt werden kann.

Die Samples werden anhand der Benennung der Spuren an den Drehzahlparameter angelegt, so dass die Drehzahl des jeweiligen Samples der linken Kante seiner Region entspricht. Nur die Leerlaufsamples werden bei 0 angelegt. Die Regions der Samples werden bis zum Anfang des nächsten Drehzahlsamples aufgezogen. Spielt man jetzt das Event ab und bewegt den RPM Parameter, wird zwischen den Samples hin- und hergesprungen und die einzelnen Samples werden nicht im Loop wiedergegeben. Es fehlen noch die Übergänge zwischen den Samples. Dazu muss im nächsten Schritt das Pitch Shifting der einzelnen Samples eingerichtet werden, bevor diese mit Fades überblendet werden können.

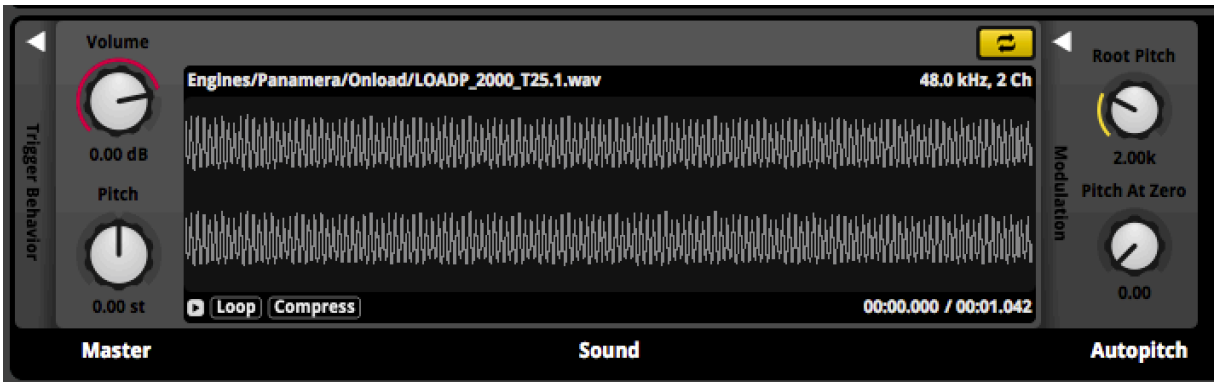

ABBILDUNG 22 SAMPLEEDITOR MIT EINGESTELLTEM ROOT PITCH

Als Nächstes muss für jedes Sample einer Drehzahl der sogenannte *Root Pitch* des *Autopitch* Modulators bestimmt und der Loopmodus im Sampleeditor aller Samples aktiviert werden. Dieser wird im Sampleeditor im oberen rechten Eck aktiviert. Die entsprechende Schaltfläche muss gelb sein, wie in *Abbildung 22* zu sehen ist.

Um den *Autopitch* Modulator nutzen zu können, muss dieser im Sampleeditor jedes Samples erstellt werden. Mit Rechtsklick auf den *Pitch-Regler* wird dieser im Untermenü „Add Modulation“ ausgewählt. An der rechten Seite des Sampleeditors werden zwei weitere Regler angezeigt, wobei nur der *Root Pitch* relevant ist. Da die Samples auf dem Drehzahlparameter angelegt sind, wird mit dem *Root Pitch* die genaue Drehzahl des Samples angegeben und mit dem korrespondierendem Wert der Parameterskala verknüpft. Das Sample wird nun oberhalb dieses Wertes hochgepitcht und unterhalb runtergepitcht. So entsteht später ein kontinuierlicher und homogener Drehzahlhochlauf.

Da die im Namen des jeweiligen Samples angegebenen Drehzahlen nur grobe Richtwerte sind, jedoch für den *Root Pitch* eine möglichst präzise Angabe der Drehzahl nötig ist, muss diese anhand folgender Formel errechnet werden:

$$Root\ Pitch = \frac{f_{1.MO} * 120}{Zylinderanzahl}$$

Bei $f_{1.MO}$ handelt es sich um die Frequenz der ersten Motorordnung. Um diese herauszufinden, muss das Sample erneut in die DAW geladen werden, um mit einem Analyzer Plug-In die entsprechende Frequenz herauszufinden. Da der *Root Pitch* zwischen allen Signalen eines Aufnahmetakes übernommen werden kann, muss nur ein Sample eines Takes analysiert werden. Für die Frequenzanalyse bietet sich das Sample, welches das Auspuffsignal enthält an, da auf diesem die tiefen Signalanteile am ausgeprägtesten sind. Als Analyzer erwies sich das *SPAN* Plug-In von *Voxengo* am zuverlässigsten. Da die Motorordnungen sehr tief liegen, muss die Auflösungen des Analyzers sehr hoch eingestellt werden, um diese gut ablesen zu können. Nachdem der *Root Pitch* mit Hilfe der Formel bestimmt wurde, kann dieser in der Middleware eingestellt werden.

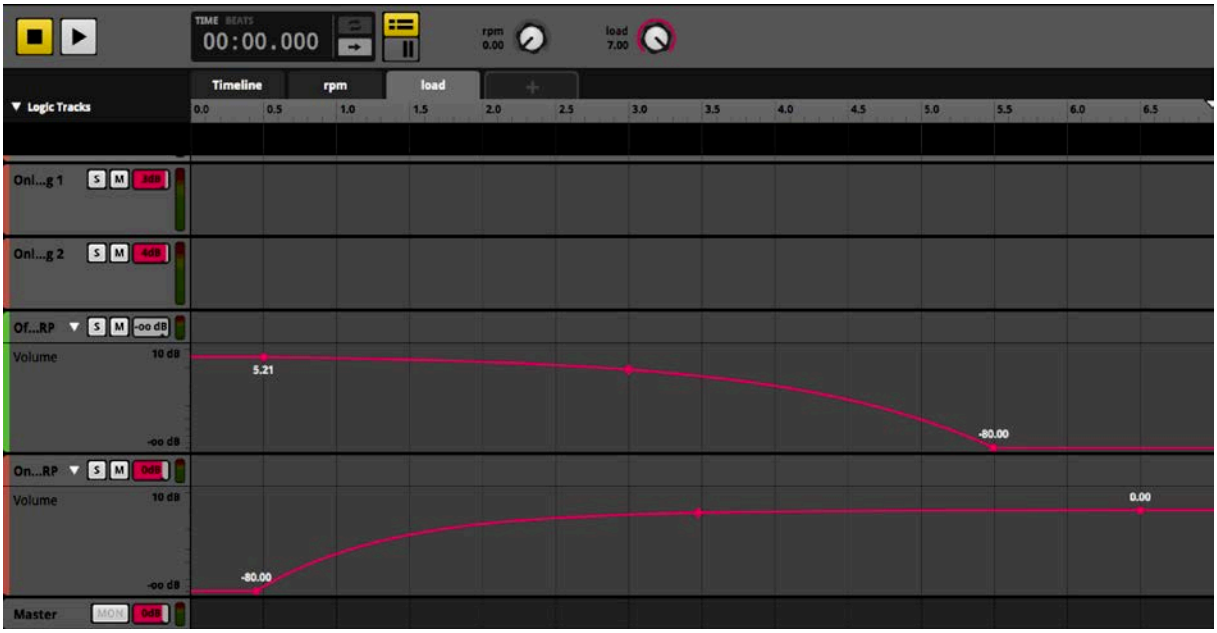

ABBILDUNG 23 ENGINE EVENT, LOAD PARAMETER

Im Reiter des Lastparameters werden die Lautstärkeautomationen erstellt, welche Leerlauf- und Lastsamples über den gesamten Regelbereich des Parameters überblenden. Um dies nicht für jede Spur einzeln vorzunehmen, wurden die Gruppenspuren angelegt. Durch Rechtsklick auf den Lautstärkeregler der Spuren kann diesem eine Automation zugeordnet

werden. Unter der jeweiligen Spur kann nun die Automation geschrieben werden. Die gewählten Automationskurven werden in *Abbildung 23* gezeigt. Bei Erstellung der Automation wurde darauf geachtet, dass sich Lautstärke und Klangfarbe gleichmäßig verändern. Es musste verhindert werden, dass bei mittlerer Regelposition die Lautstärke abfällt.

Nachdem die Lautstärkeautomationen im Lastparameter definiert und der *Root Pitch* berechnet und eingestellt wurde, können die Samples mit Hilfe von Crossfades ineinander übergeblendet werden. Dazu werden die Regions aufgezogen, so dass diese überlappen. Die Fadelänge und dessen Form muss nach Gehör angepasst werden. Ziel ist es, dass ein homogener Übergang zwischen den Samples entsteht. Sprünge in Klangfarbe und Tonhöhe sollen entfernt werden. Dies muss für Last- und Leerlaufsamples getrennt vorgenommen werden. Jedoch können die Fades innerhalb der Gruppen kopiert werden. Wichtig bei der Feinjustierung ist zudem, dass sich bei dem Verändern des Lastparameters lediglich die Klangfarbe und die Lautstärke verändert, nicht aber die Tonhöhe. Ist dies nicht der Fall, müssen manuell die korrespondierenden Leerlauf- und Lastsamples angepasst werden. Das wird am besten durch minimale Veränderung des *Root Pitch* realisiert. Dies ist auch nötig, wenn im Übergang zweier aufeinanderfolgenden Samples starke Phasenauslöschungen entstehen. Als letzter Schritt werden die Spuren mittels des Surround Panners auf die entsprechenden Lautsprecher verteilt, so dass am und um das Quad ein authentisches und homogenes Klangbild entsteht. Die genaue Abstimmung des Panoramas und der Lautstärkeverhältnisse muss am Quad durchgeführt werden. Der Motorklang kann nach Bedarf mit den internen Effekten der Middleware weiter optimiert werden.

Nachdem der Motorklangerzeugung im *Engine* Event abgeschlossen ist, müssen im *Engine_total* Event noch die Parameter zur Steuerung angelegt werden. Der Lastparameter ist identisch zu dem im *Engine* Event. Beim Drehzahlparameter unterscheidet sich der Wertebereich. Der Minimumwert entspricht dem kleinsten *Root Pitch.* Der Maximalwert dem Drehzahlbegrenzer. Dies wird gemacht, um bei der Programmierung Parameteroffsets zu vermeiden, die zur Berechnung des Drehzahlfaktors sonst nötig wären. Wichtig ist dabei, das die Schreibweise aller Parameter einheitlich ist.

ABBILDUNG 24 SEEK SPEED

Als letzter Schritt muss der *Seek Speed* des Drehzahlparameters eingestellt werden. Dazu muss auf den entsprechenden Drehregler des Parameters im *Engine_toal* Event geklickt werden. Im *Deck* werden nun weitere Optionen für den Parameter angezeigt. Der *Seek Speed* sorgt für eine Glättung der Parameteränderung. Bei größeren Sprüngen wird zwischen Start- und Endwert kontinuierlich interpoliert. Der *Seek Speed* gibt dabei die maximale Änderungsrate pro Sekunde an. Dieser spielt für den Leerlauf eine Rolle, so dass ein geschmeidiger Hochlauf entsteht.

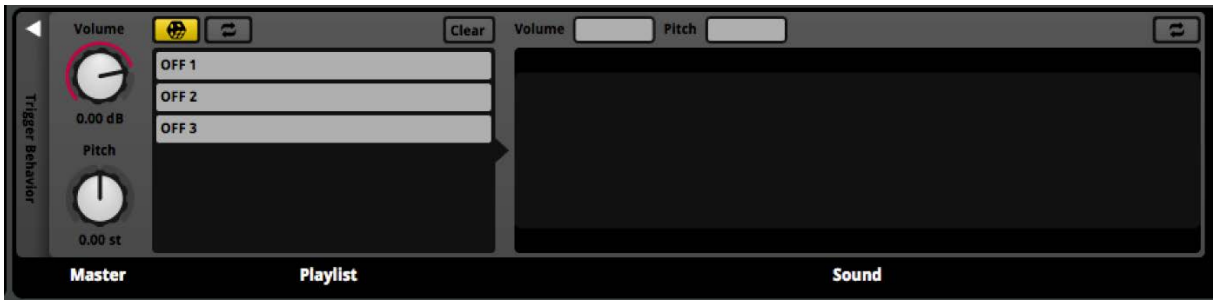

ABBILDUNG 25 MULTI SOUND MODUL

Schließlich können die Start- und Abstellsounds implementiert werden, die nötigen Events wurden bereits im *Engine_total* Event erzeugt. Die Implementierung der Start- und Abstellsounds ist dabei identisch. Da mehrere Start- und Abstellvorgänge gesampelt wurden, wird im jeweiligen ein Event *Multi Sound* Modul erstellt. In dieses können mehrere Sounds geladen werden und zufällig wiedergegeben. Es ist zudem möglich im Modul Events zu erstellen, welche sich wie *Nested Events* verhalten. Es werden entsprechend der Anzahl der gesampelten Vorgänge Events erstellt. In diesen werden anschließend die einzelnen Takes angelegt. Wichtig ist, dass das Sound Modul und das dazugehörige *Nested Event* im *Engine_total* Event dieselbe Länge, wie die Samples haben.

ABBILDUNG 26 OFF SOUND EVENT

Die Events können schließlich im *Engine_total* Event überblendet werden, wie in *Abbildung 20* zu sehen ist. Bevor das *Engine* Event in das *OFF* Event übergeblendet wird, muss ein *Sustain Point* gesetzt werden. Dieser wird erstellt, indem an entsprechender Stelle in schwarzen Bereich unter der Skala mit rechter Maustaste geklickt wird. Dieser dient beim Wiedergeben des Events dazu, dass das Event bis zu diesem Punkt abgespielt und dann gehalten wird. Die Steuerung über die Parameter ist weiterhin möglich. Neben den Drehreglern der Parameter wird nun eine weitere Schaltfläche anzeigt, welche mit *Cue* bezeichnet ist. Wird dieser *Cue* ausgelöst, wird der *Sustain Point* übersprungen und das Event läuft regulär weiter. Der *Cue* wird später vom Programmcode ausgelöst, wenn der Motor abgestellt werden soll.

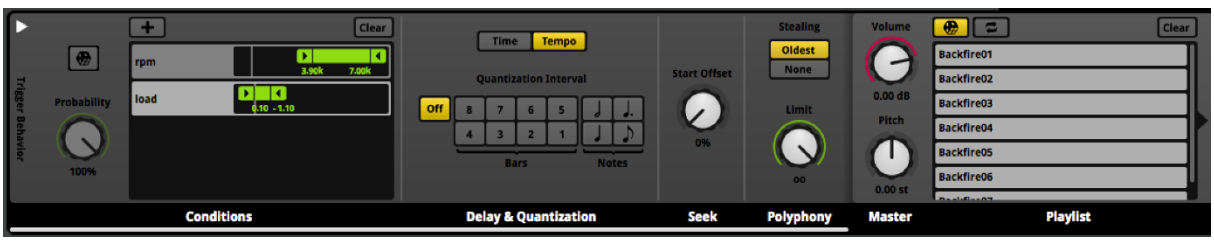

ABBILDUNG 27 PROGRAMMER SOUND MODUL

Beim *Porsche Panamera* wurden neben dem Motorsound auch die Fehlzündungen gesampelt. Diese werden über das *Programmer Sound* Modul angesteuert, welches auf einer separaten Spur im *Engine_total* Event erstellt wird. Dessen Länge entsprecht der des *Engine* Events. Beim

*Programmer Soun*d Modul handelt es sich um ein erweiterter *Multi Sound* Modul. Es bietet die Möglichkeit die Auslösebedingungen selbst zu wählen. Unter *Conditions* werden beide Parameter ausgewählt. Beim realen Fahrzeug entstehen die Fehlzündungen, wenn im oberen Drehzahlbereich vom Gaspedal gegangen wird. Für die Anwendung bedeutet das, dass die entsprechenden Samples nur im hohem Drehzahlbereich bei kleiner Last abgespielt werden sollen. Dies wird im *Programmer Sound* Modul eingestellt. Wichtig ist, dass beim Lastparameter nicht 0 als untere Auslösegrenze gesetzt wird, da im Stand keine Fehlzündungen abgespielt werden sollen. Die genauen Einstellungen können der *Abbildung 27* entnommen werden.

ABBILDUNG 28 MIXER OBERFLÄCHE

Um alle Sounds zu finalisieren, muss dies am Quad geschehen. Dabei ist das *Live Update* eine hilfreiche Funktion. In der Middleware muss diese Funktion im unteren rechten Bereich aktiviert werden. Dazu muss der Soundgeneratoranwendung gestartet sein. Öffnet man das Mixer Fenster müssen jedoch zuerst VCA Gruppen erstellt werden und diesen Events zugewiesen werden. Dies wird im linken Bereich des Fensters umgesetzt. Das genaue Routing kann der *Abbildung 28* entnommen werden. Neben dem Einstellen der Lautstärke und dem Panorama können jedem VCA Effekte zugewiesen werden. Bei der Finalisierung wurden alle Sounds in ihrer Lautstärke angepasst. Alle Motorsounds wurden gleich laut eingestellt. Zudem wurden die Sounds mittels EQ auf das Wiedergabemedium optimiert.

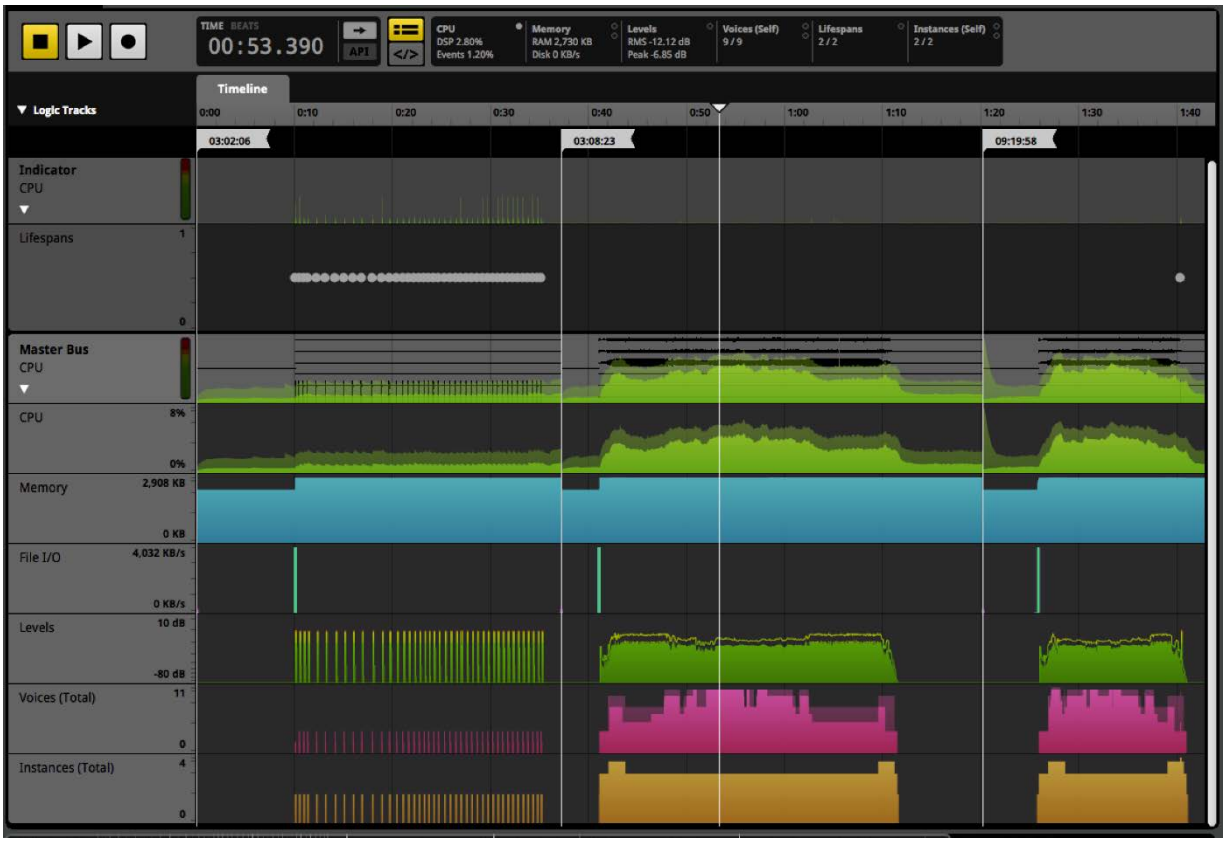

ABBILDUNG 29 PROFILER OBERFLÄCHE

Mit der *Profiler* Funktion von *FMOD Studio* kann zum Schluss die Funktion und Ansprache der Events überprüft werden. Dazu wird in der Middleware das *Profiler* Fenster geöffnet. Für die Nutzung muss dieser bei der Initialisierung des Studio Systems im Programmiercode aktiviert werden. In diesem können alle Aktivitäten der Audio Engine während der Ausführung des Soundgenerators aufgezeichnet und kontrolliert werden. Es ist außerdem möglich, anhand der aufgenommenen Steuersignale, Änderungen zu testen.

5. Fazit

Aufgabe dieser Abschlussarbeit war die softwareseitige Weiter- und Neuentwicklung eines bereits vorhandenen Soundgenerators für Elektrofahrzeuge. Zielsetzung war es, diesen um die Klangvarianz bei unterschiedlichen Motorlastmomenten zu erweitern.

Basis dafür war ein von der *GIGATRONIK Stuttgart GmbH* umgebauter Quad, bei dem der Antriebstrang auf Basis eines Verbrennungsmotors durch einen selbstentwickelten Elektroantrieb ersetzt wurde. Der darauf befindliche Soundgenerator basierte auf dem Softwaresampler Halion. Eine eigens entwickelte Hardware, welche die CAN Nachrichten in das Midi Protokoll transferierte, diente zur Ansteuerung des Samplers. Nach Analyse des Systems wurde festgestellt, dass sich diese Art der Motorklangerzeugung nicht zur Umsetzung der neuen Anforderungen eignet. Der neue Soundgenerator wurde schließlich mit der Audio Engine *FMOD Studio* von *Firelight Technologies* umgesetzt, einer Anwendung zur dynamischen Klangerzeugung aus der Gaming Branche. Die Midi Ansteuerung wurde durch eine direkte Ansteuerung über CAN Nachrichten ersetzt.

Für die softwareseitige Neuentwicklung konnte zwar auf firmenintern entwickelte Software zurückgegriffen werden, jedoch sollte ein großer Teil in Eigenregie programmiert werden. Daher war es erforderlich sich die dazu nötigen Programmierkenntnisse anzueignen. Die Audio Engine musste mittels der vom Entwickler bereitgestellten API eingebunden und mit dem firmenintern entwickelten Programm „CarPC“ verbunden werden. Um mit den in den CAN Nachrichten enthaltenen Signalen die Audio Engine anzusteuern, mussten diese angepasst und verarbeitet werden. Es wurde eine Logik entwickelt, welche das Drehzahlband der Elektromotoren am Quad zu den Drehzahlbändern der synthetisierten Motorenklänge anpasst. Des Weiteren musste der Motorlastparameter definiert und aus den Drehmomentsignalen generiert werden.

Parallel zur Programmierung wurde in der Middleware von *FMOD Studio* eine Projektstruktur erarbeitet, die eine einfache Implementierung und Erweiterung von Sounds möglich macht und kompatibel zum programmierten Anwendung ist. Es sollten mit Hilfe der internen Werkzeuge und Effekte und mit einer möglichst simplen Ansteuerung möglichst authentische Motorsounds erzeugt werden. Da die Audio Engine jedoch noch weitere Anwendungsmöglichkeiten bietet, sollte dies mit dem akustischen Feedback des Blinkers und Warnblinkers demonstriert werden.

Als letztes wurden die aufgenommenen Fahrzeugsounds technisch und klanglich für den Einsatz im Soundgenerator vorbereitet. Störschall musste minimiert und die aufgenommenen Signale den Hörgewohnheiten klanglich angepasst werden. Nach der Bearbeitung wurden die Samples für die Implementierung in *FMOD Studio* vorbereitet. Dazu mussten diese extrem akkurat geschnitten werden, so dass später kein Knacken oder inhaltliche Sprünge entstehen. Nachfolgend wurden alle Sounds in die Audio Engine implementiert. Dabei wurde sehr lang an den optimalen Übergängen zwischen den Samples gearbeitet, so dass ein möglichst authentisches Motorklangbild entstand. Als letzter Schritt wurden die Samples auf das Wiedergabesystem angepasst und untereinander in ihrer Lautstärke angeglichen.

Der entwickelte Soundgenerator ist in der Lage über das erweiterte Wiedergabesystem (siehe Müller L. , 2017) einen glaubhaft authentischen Motorsound zu generieren, der sich dynamisch der aktuellen Fahrsituation anpasst. Zudem konnte der benötigte Datenaufwand und die benötige Rechenleistung stark reduziert werden. Durch die realistische Verteilung der Schallquellen konnte rund um das Quad ein homogenes Klangbild geschaffen werden, wie man es von herkömmlichen Fahrzeugen mit Verbrennungsmotor gewohnt ist.

5.1 Mögliche zukünftige Erweiterungen des Soundgenerators

Die neue Umsetzung des Soundgenerators mittels der Audio Engine *FMOD Studio* und der direkten Ansteuerung über den CAN Bus bietet weiterhin Erweiterungspotential.

Ein naheliegendes Erweiterungsziel ist, den Soundgenerator um weitere Motorensounds zu erweitern. Es kann zudem mit der Klangerzeugung selbst experimentiert werden. Neben dem klassischen Sampling, wie es bei dieser Umsetzung der Fall war, kann auch mit einem Dritthersteller Plug-In für die Audio Engine experimentiert werden, welche die Motorsounds auf Basis einer Granularsynthese generiert. Ein Vorteil wäre die einfachere und schnellere Implementierung neuer Sounds und der noch geringere Datenaufwand. Nachteil können Klangartefakte und eine höhere Prozessorauslastung sein. Um diese beiden Verfahren objektiv zu vergleichen, müssten diese praktisch gegenübergestellt und evaluiert werden.

Es bietet sich zudem an, den Soundgenerator um weitere separat angesteuerte Soundkomponenten zu erweitern. Eine Komponente wäre, das Turboladergeräusch oder ein virtuelles Automatikgetriebe zu erweitern und Schaltmomente einzubauen. Dies kann dem Fahrer beim akustischen Einschätzen der gefahrenen Geschwindigkeit unterstützen und die allgemeine Authentizität erhöhen.

Die jetzige Umsetzung ermöglicht es, jedes vorhandene CAN Signal akustisch wiederzugeben. Es wäre zum Beispiel möglich, dem Fahrer akustisch den Status der Akkus mitzuteilen. Es könnte ein Warnton bei niedrigem Akkustand wiedergegeben werden oder eine akustische Information beim Starten des Quads wie voll die Akkus momentan geladen sind.

Das langfristige Entwicklungsziel sollte jedoch sein, auf Basis dieses Soundgenerators ein seriennahes Steuergerät zu entwickeln, welches einfach und kompakt in einem Elektrofahrzeug untergebracht werden kann. Die Umsetzung mit einem *Windows PC* und einem Audio Interface als Soundkarte ist für ein Technologieträger- und Showfahrzeug ausreichend, jedoch weit von einem Serieneinsatz entfernt. Der Fokus der weiteren Entwicklung sollte auf der software- und hardwareseitigen Performanceoptimierung liegen, um dieses Konzept langfristig zu einem seriennahen Prototyp weiterzuentwickeln.

Danksagung

Ich bedanke mich herzlichst bei der Firma *GIGATRONIK Stuttgart Gm*bH, welche die Realisierung dieser Abschlussarbeit möglich gemacht hat. Besonderer Dank gilt meinem Zweitprüfer, Herrn Dipl.-Ing. Jochen Lüling, für die Betreuung und Unterstützung. Hervorzuheben sind zudem Simon Kunz, Raphael Wank, Florian Klingenstein, Lukas Herrmann und Tobias Holl, die uns mit Rat und Tat bei der Umsetzung des Projektes unterstützt haben.

Besonders herzlicher Dank gilt auch meinem Erstprüfer, Herrn Prof. Oliver Curdt. Herr Prof. Curdt hat mich während des Studiums und der Abschlussarbeit immer engagiert betreut, unterstützt und gefördert.

Des Weiteren danke ich Lukas Müller, mit dessen hervorragender Zusammenarbeit dieses Projekt innerhalb von sechs Monaten realisiert wurde.

Zuletzt möchte ich mich bei meiner Familie bedanken, welche mich mein ganzes Studium aktiv unterstützt und gefördert hat.

Literaturverzeichnis

AudioGaming. (2016). *AudioMotors*. Abgerufen am 12. Januar 2017 von LeSound: http://lesound.io/product/audiomotors-pro/

Busche, A. (24. Januar 2012). *Sound Design – Ton im Film*. Abgerufen am 12. Januar 2017 von kinofenster.de: http://www.kinofenster.de/film-des-monats/archiv-film-des-monats/kf1202/sounddesign-ton-im-film/

CAN in Automation (CiA). (2016). *CAN knowledge*. Abgerufen am 12. Januar 2017 von CiA: https://www.can-cia.org/en/can-knowledge/

Crankcase Audio. (2016). *Crankcase Audio*. Abgerufen am 12. Januar 2017 von Crankcase Audio: http://www.crankcaseaudio.com/

Digiprost. (2014). *Project CARS Vs Real Life.* Abgerufen am 12. Januar 2017 von YouTube: https://i.ytimg.com/vi/fTrwsokoNXg/maxresdefault.jpg

Dmoch, M. (2016). *Game Audio - Vertonung von Computerspielen.* Stuttgart.

Donnelly, C. (11. August 2014). *Vehicle Engine design – Project CARS, Forza Motorsport 5 and REV*. Abgerufen am 12. Januar 2017 von Designing Sound: http://designingsound.org/2014/08/vehicle-engine-design-project-cars-forza-motorsport-5-and-rev/

elektro-archiv.de. (2016). *CAN*. Abgerufen am 12. Januar 2017 von elektro-archiv.de: http://www.elektro-archiv.de/?artikel=CAN

Firelight Technologies. (2016). *Documentation*. Abgerufen am 12. Januar 2017 von FMOD: http://www.fmod.org/documentation

Firelight Technologies. (2016). *FMOD Studio User Manual.* Abgerufen am 12. Januar 2017 von FMOD.org: http://www.fmod.org/

Gründerszene. (2016). *Application-Programming-Interface (API)*. Abgerufen am 12. Januar 2017 von Gründerszene LEXIKON: http://www.gruenderszene.de/lexikon/begriffe/application-programming-interface-api

Horowitz, S., & Looney, S. (11. August 2014). *Masterclass: Using Game Audio Middleware*. Abgerufen am 12. Januar 2017 von Electronic Musician: http://www.emusician.com/how-to/1334/masterclass-game-audio-middleware/48158

iZotope. (2016). *RX 5 Audio Editor*. Abgerufen am 12. Januar 2017 von iZotope: https://www.izotope.com/en/products/repair-and-edit/rx.html

Kunos Simulazioni. (2016). Audio pipeline v1.9.

ME-Meßsysteme GmbH. (18. August 2016). *Grundlagen.* Von ME-Meßsysteme: https://www.me-systeme.de/de/support/grundlagen abgerufen

Müller, L. (2017). *Weiterentwicklung eines Soundgenerators für Elektrofahrzeuge - Konzeption und hardwarespezifische Umsetzung eines neuen Systems.* Stuttgart: Hochschule der Medien, Bachelorarbeit.

Müller, P. (2014). *Klangsynthese - Musik/Sounddesign mit Synthesizern.* Stuttgart: Hochschule der Medien.

Raffaseder, H. (2010). *Audiodesign.* München: Carl Hanser Verlag.

Segeberg, H., & Schätzlein, F. (2005). *Sound Zur Technologie und Ästhetik des Akustischen in den Medien.* Marburg: Schüren Verlag GmbH.

Sound Radix. (2016). *PHASE INTERACTIONS MIXER*. Abgerufen am 12. Januar 2017 von SoundRadix.com: https://www.soundradix.com/products/pi/

The Fourth Man. (2014). *Akai S900 sampler 1991-1994*. Abgerufen am 12. Januar 2017 von The Fourth Man: http://thefourthman.com/akai-s900-sampler-1991-1994.html

Wikipedia. (2016). *Abtastung (Signalverarbeitung)*. Abgerufen am 12. Januar 2017 von Wikipedia: https://de.wikipedia.org/wiki/Abtastung_(Signalverarbeitung)

Wikipedia. (2016). *Controller Area Network*. Abgerufen am 12. Januar 2017 von Wikipedia: https://de.wikipedia.org/wiki/Controller_Area_Network

Wikipedia. (2016). *MQTT*. Abgerufen am 12. Januar 2017 von Wikipedia: https://de.wikipedia.org/wiki/MQTT

Wikipedia. (2016). *Software Development Kit*. Abgerufen am 12. Januar 2017 von Wikipedia: https://de.wikipedia.org/wiki/Software_Development_Kit