Bacheloroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Det humanistiske fakultet
Institutt for musikk

Håkon F. Fjellanger

Utfordringer med bruken av flere Asio-lydkort i samme programvare

En undersøkelse av utfordringene som hindrer oss i å bryte med dagens standard

Bacheloroppgave i Musikkteknologi
Veileder: Øyvind Brandtsegg
Mai 2023

NTNU
Kunnskap for en bedre verden

Håkon F. Fjellanger

Utfordringer med bruken av flere Asio-lydkort i samme programvare

En undersøkelse av utfordringene som hindrer oss i å bryte med dagens standard

Bacheloroppgave i Musikkteknologi
Veileder: Øyvind Brandtsegg
Mai 2023

Norges teknisk-naturvitenskapelige universitet
Det humanistiske fakultet
Institutt for musikk

NTNU
Kunnskap for en bedre verden

Sammendrag

Målet for dette prosjektet er å kartlegge de teknologiske utfordringene som en kombinert lyddriver for Asio-lydkort må ta høyde for. Hensikten med dette arbeidet er å bidra til målet om å skape et pålitelig Asio-aggregat for Windows-brukere. Informasjonen er bygget på resultatene fra maskinvaretester som er laget i sammenheng med prosjektet.

Resultatene viser at det er mulig å kontrollere to lydkort i samme programvare. De viser også at lydkortene kan kjøre hver sin isolerte lydprosessering, men at det oppstår utfordringer når signaler skal rutes mellom dem. Arbeidet produserer pålitelige kilder til informasjon som øker kunnskapen til et av musikkteknologiens sentrale verktøy.

Innhold

Terminologiliste

DAW	-	Forkortelse for Digital Audio Workstation, og henviser til musikkredigeringsprogrammer som Logic, Ableton, Pro Tools, og Reaper.
Lydkort	-	USB-enhet som konverterer fra analogt til digitalt signal, og som brukes i en DAW.
Asio	-	Protokollen som brukes mellom et lydkort og en DAW. Man kan se på dette som den offisielle malen for måten disse to elementene kommuniserer.
Asio-lydkort	-	Et lydkort som kan brukes i en DAW gjennom Asio.
Driver	-	Et lag mellom en maskinvare og programvare som oversetter programvare-instruksjoner til maskinvare-handlinger.
Aggregat	-	En virtuell driver som kombinerer flere lydkort til en felles enhet.
Funksjon	-	En prosedyre inni en programvare. En rekkefølge av programvareinstruksjoner.
Funksjonskall	-	Leses funksjons-kall. Refererer til programvare-operasjonen som starter en funksjon.
Metode	-	En funksjon som er en del av et digitalt objekt. En metode er en funksjon, men en funksjon er ikke en metode.
Programinstans	-	Man kan kjøre flere instanser av samme programvare, eksempelvis kan man ha flere Word-filer åpne samtidig. Dette begrepet peker til én av disse.

Mål

Målet for prosjektet er å bidra i prosessen til å skape en mer pålitelig kombinert lyddriver for Windows-brukere. Dette bidraget skal være en kartlegging av de teknologiske muligheter og utfordringer en slik løsning må ta høyde for. Informasjonen skal være basert på resultater fra programvare som tester lydkort via Asio-protokollen.

Problematikken som står i senter av dette prosjektet, er utfordringen med å bruke flere lydkort i samme programvare på Windows. De fleste vil fortelle deg at Asio4All løser dette problemet. De vil samtidig anerkjenne at denne løsningen ikke alltid fungerer som forventet. Samtidig finnes det en fungerende løsning for iOS-brukere som ikke er tilgjengelig for Windows-brukere. Dette prosjektet skal bidra til en løsning som kan bryte dette skillet.

Avgrensninger

Jeg setter flere avgrensninger for dette prosjektets målområde.

Det er ikke en del av prosjektets resultatmål å lage en kombinert lyddriver. Dette målet ligger utenfor prosjektets rekkevidde, men brukes likevel til å skape retning og fremdrift.

Prosjektet skal kun forholde seg til spesifikke Asio-drivere. En kombinert lyddriver skal fungere for alle Asio-lydkort. Det er dermed lite hensiktsmessig å studere spesifikke driver-implementasjoner. Når det er sagt, er det fortsatt interessant å ta høyde for oppførselen som resulterer i at vi bruker driveren. Slik oppførsel bør tas høyde for av det resulterende produktet, og vil derfor være et relevant resultat for dette prosjektet.

Man skal etter beste evne unngå å trekke konklusjoner eller argumentere utfra det som kommer frem i diskusjonsfora og avisartikler. Det skal tilstrebes å basere informasjonen på resultater fra maskinvaretester.

Jeg ønsker kun å observere hvilke utfordringer som hindrer samarbeidet mellom to lydkort. Flere lydkort enn dette antallet vil antagelig introdusere flere utfordringer. Det er hensiktsmessig å begrense kompleksiteten i arbeidet for å sikre et mer kvalitativt resultat.

Alle programvaretester skal kjøres på Windows. Dermed vil de dataene vi samler være preget av det målsatte operativsystemet. Videre skal ikke forskjeller mellom iOS og Windows problematiseres. Vi kan ikke forandre på de premissene som settes av operativsystemet, og det er dermed lite hensiktsmessig å bruke tid og ressurser på slikt.

Motivasjon

I dette kapittelet vil jeg legge frem motivasjonen som inspirerte dette prosjektet.

USB

En vanlig datamaskin tillater at brukeren kobler til flere USB-objekter av samme type. Eksempelvis kan man koble til flere minnepinner samtidig, og få tilgang til dataene som ligger inni hver av dem. Man kan også overføre data fra en minnepinne til en annen, og dermed få dem til å samarbeide om ett felles mål.

Asio-lydkort er et unntak til denne egenskapen, og jeg er interessert i å finne de teknologiske hindringene som ligger bak. Min forventning er at hindringene er såpass komplekse at de ikke kan løses. Likevel vet jeg at det finnes en løsning på problemet for Mac-brukere, og jeg håper å finne ut at hindringene ikke er uløselige.

Motvirke pseudovitenskap

I startfasen av prosjektet har jeg oppdaget at det er vanskelig å finne informasjon om Asio-protokollen. Det er også lite hjelp å få om man vil vite mer om protokollens indre verker, utenom det å lese koden selv. I et slikt miljø kan det lett oppstå pseudovitenskapelige teorier om hvordan dette verktøyet opererer. Derfor ønsker jeg å samle håndfaste bevis, og på denne måten bidra til å forbedre vår felles forståelse av hvordan lydkort fungerer.

Kombinere lydkort for å få flere inputs

En typisk historie for en som starter å produsere musikk i dag likner ofte på følgende. Man anskaffer seg et lite og billig lydkort med to inputs. Det er jo tross alt bare en hobby, og man vil ikke investere for mye penger i det med en gang. Etter hvert som prosjektene blir større, øker behovet for antall inputs og man kjøper seg et større lydkort. Dagen kommer der dette behovet overstiger det store lydkortet med én input. Hvis man er så uheldig å bruke en Windows-datamaskin, vil man møte på uløselige problemer når man forsøker å få dette til.

Jeg ønsker å rette mitt arbeid mot å løse dette problemet. Det er ingen rollemessig begrunnelse som hindrer Windows i å tilby en slik løsning. Målet mitt er ikke å lage en ferdig og fungerende løsning. Jeg ønsker heller å kartlegge problematikken som gjør dette målet til en utfordring.

Metode

I dette kapittelet ønsker jeg å gi leseren et godt perspektiv på dybden av – og dermed troverdigheten til – den informasjonen som kommer frem av dette prosjektet.

Metoden min er forsøksbasert. Jeg har skrevet egen kode som tester hypoteser og løsninger. Begrunnelsen for å velge en forsøksbasert metode ligger i ønsket om å anskaffe håndfaste bevis. Jeg mener at mange musikkprodusenters kunnskap om lydkort består mest av teorier basert på løs spekulasjon. For å bryte med dette mønsteret, ønsker jeg at de konklusjonene som trekkes ut av dette arbeidet skal være basert på håndfaste bevismateriale. Med denne metoden håper jeg å lette på tåken som omfavner et av våre viktigste verktøy i musikkindustrien.

Innbakt i denne metoden er oppgaven om å lese både kode og dokumentasjon tilknyttet Asio. Informasjon jeg oppdager som et resultat av disse aktivitetene, ansees som resultater på lik linje med utdataene fra programvaretester.

For å gi meg selv en retning og fremdrift, har jeg satt meg målet om å lage et fungerende Asio-aggregat. Dette aggregatet skal fungere selv om lydkortene ikke har synkroniserte klokker. Det skal ha lav nok forsinkelse i lydsignalet til at det kan brukes både i studio og på scenen. Et slikt produkt vil gi en komplett løsning på det jeg problematiserer, og vil sende meg i retning mot de hindringene som er kilden til problemet. Jeg har ikke satt meg dette målet for å skape et ferdig produkt, men heller for å bruke prosessen som en katalysator for kunnskapsdannelse.

All kode er skrevet i C++. Verktøyene jeg har brukt til å skrive, teste og kjøre koden er disse:

- **Kompilator:** Visual Studio Community 2019. Denne kompilatoren er laget og distribuert av Microsoft, og kompilerer kode som kan kjøre på operativsystemet Windows.
- **Utviklingsmiljø:** Visual Studio Code, også kjent som VSCode.
- **Byggesystem:** CMake. Relevante CMakeLists-filer ligger innbakt i kodeprosjektets mappestruktur.
- **Enhetstesting:** CTest.

Maskinvarene jeg har brukt for å gjennomføre testene er som følger:

- **Lydkort 1:** Behringer u-phoria umc1820
- **Lydkort 2:** Motu M2
- **Mikrofon:** Shure SM58B
- **Datamaskin:** Acer Nitro 5 laptop med Windows operativsystem.
    - **CPU:** AMD Ryzen 5 5600H.
    - **Grafikkprosessor:** Nvidia GeForce RTX 3060 Laptop
    - **USB:** AMD USB 3.10 eXtensible Host Controller – 1.10 (Microsoft)
    - **USB versjon:** 3.0

Resultat

I dette kapittelet presenteres resultatene av arbeidet som er gjennomført. Vi begynner med en test av Asio4All, før det presenteres interessante oppdagelser om Asio-biblioteket. Til slutt legges det frem resultatene av programvaretester utført på to Asio-lydkort.

Asio4All

Asio4All er en programvare for Windows som skaper en virtuell driver for lydenheter som ikke støtter Asio. Målet til Asio4All er å gjøre disse driverne tilgjengelige for programvare som kun støtter Asio-enheter. Dette gjøres ved skape en bro mellom enhetens WDM-driver og Asio-protokollen.

Asio4All må kunne støtte flerkanals-enheter som velgere å fordele kanalene sine utover flere drivere. En bi-effekt av denne funksjonaliteten er at man kan sette sammen drivere fra forskjellige enheter. Dermed er det mulig å skape en kombinert lyddriver via Asio4All.

Testing av kombinert lyddriver med Asio4All

I dette delkapittelet dokumenteres ytelsen til en kombinert lyddriver laget med Asio4All.

I denne kombinerte driveren inkluderer vi kun én input og én output fra to lydkort. Programmet Reaper brukes for å kryss-koble lydkortene. Det vil si at hver input fra begge lydkort sendes til alle outputs i den kombinerte lyddriveren. Her er variablene for forsøket:

- Felles samplingsfrekvens: 48000
- Felles bufferstørrelse: 128
- Lydkort 1: Motu m2
- Lydkort 2: Behringer u-phoria umc1820

Det bør nevnes at de to opplistede lydkortene fungerer hver for seg uten problemer, både gjennom egen Asio-driver og via Asio4All.

Her er den observerte statusen til hver av signalrutene. ‘Umc’ referer til Behringer u-phoria umc1820, ‘Motu’ refererer til Motu m2:

Input		**Output**	**Status**
Umc	->	Umc	Ok. Liten forsinkelse.
Umc	->	Motu	Ok. Liten forsinkelse.
Motu	->	Umc	Starter ok. Går over til digital støy. Avslutter med rent signal forsinket med omtrent ett sekund.
Motu	->	Motu	Samme som Motu -> Umc.

Her ser vi at Asio4All klarer å rute lyd mellom Umc sine egne porter og fra Umc til Motu.

Motu sin input er det som viser problemer. Den fungerer utmerket i ca. ett minutt. Etter det hører man mye digitalt støy. Til slutt forsvinner støyen, men da har lydsignalet omtrent ett sekund forsinkelse. Et opptak av inputen til Motu ligger vedlagt: les Appendiks B - Testing av kombinert lyddriver med Asio4All.

Asio-biblioteket støtter ikke bruken av flere lydkort

I dette kapittelet presenteres egenskaper ved Asio-biblioteket som gjør at det kun tillater ett lydkort per program, mer spesifikt per programinstans.

Grunnen til at dette er interessant, er at Asio-biblioteket er en avhengighet hos andre programvare. Programvarebiblioteket PortAudio er et eksempel på dette[1]. De reglene som settes i Asio-biblioteket må følges av de som bruker det. Ved å kjenne til disse reglene, kan vi si noe om hvilke rammer som Steinberg har satt for Asio-protokollen.

Når man laster inn et lydkort, lagres en referanse til driveren i den globale variabelen *theAsioDriver*. Hver gang man bytter fra ett lydkort til et annet, overskrives denne variabelen med det nye lydkortet. De offisielle Asio-funksjonene – som DAW-utviklere forventes å bruke[2] – viderefører sitt funksjonskall til denne variabelen. Metoden de påkaller har samme navn som dem selv, med unntak av at funksjonene starter med 'ASIO'.

Dette betyr at alle disse funksjonene er avhengige av verdien til *theAsioDriver*. Når man laster inn et nytt lydkort, forandrer man i tillegg hvilket lydkort disse funksjonene agerer på. Dette er strukturen som håndhever prinsippet om ett lydkort per programinstans. Det er ikke laget noen flere variabler som gjør at man kan bryte med dette prinsippet.

```
110  ASIOError ASIOStart(void)
111  {
112      if(!theAsioDriver)
113          return ASE_NotPresent;
114      return theAsioDriver->start();
115  }
116
117  ASIOError ASIOStop(void)
118  {
119      if(!theAsioDriver)
120          return ASE_NotPresent;
121      return theAsioDriver->stop();
122  }
123
124  ASIOError ASIOGetChannels(long *numInputChannels, long *numOutputChannels)
125  {
126      if(!theAsioDriver)
127      {
128          *numInputChannels = *numOutputChannels = 0;
129          return ASE_NotPresent;
130      }
131      return theAsioDriver->getChannels(numInputChannels, numOutputChannels);
132  }
133
```

Figur 1 Utsnitt av Asio-funksjonene

[1] PortAudio, u.å.
[2] ASIO 2.3, u.å., s. 10

Utdypelse rundt *theAsioDriver*

Variabelen *theAsioDriver* er definert på linje 36 i kildefilen asio.cpp, og er av typen IASIO. Datatypen IASIO er protokollens abstrakte definisjon av en lydkort-driver. Den inneholder funksjoner for å eksempelvis bytte samplingsfrekvens, åpne driverens kontrollpanel, lese av antallet kanaler, etc. Definisjonen til IASIO kan man finne i filen iasiodrv.h.
Legg merke til at datatypens funksjoner mangler funksjonskropp. Dette er fordi Asio-protokollen kun definerer malen for kommunikasjon mellom lydkort og annen programvare. Denne malen benyttes slik: lydkort-utviklere lager funksjonskroppene, mens programvare-utviklere bruker disse funksjonene.

Bemanne flere lydkort samtidig

Her presenteres to enkle programmer som bruker to lydkort samtidig. Hensikten med det første programmet er å finne ut om det er mulig å kommunisere med to lydkort-drivere i samme programvare. I det andre programmet undersøker vi om det er mulig å kjøre maskinvare-funksjonalitet på to lydkort samtidig.

Det første programmet lar brukeren åpne konfigurasjons-panelet til to forskjellige lydkort-drivere. Funksjonene for å åpne konfigurasjonspanelene påkalles rett etter hverandre, som betyr at begge instansene er aktive når hvert vindu åpnes.

Det andre programmet spiller av en sinus-tone på to lydkort samtidig, spesifikt gjennom output 1 og 2. Man starter og stopper sinustonen på hver av dem via kommandolinjen.

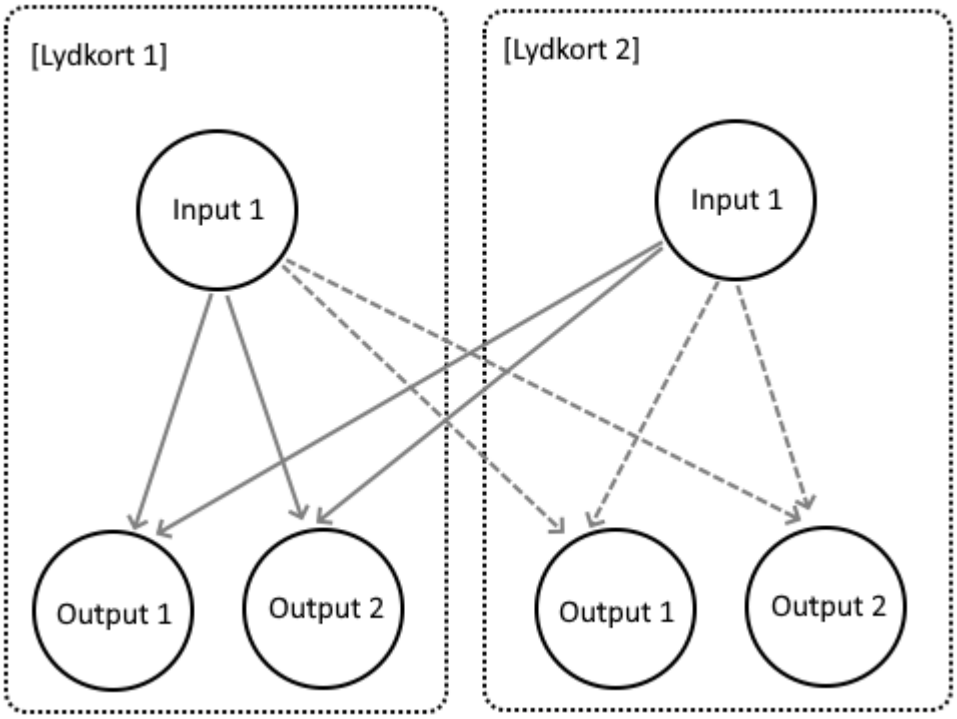
For å kunne realisere disse to programmene har jeg måttet omgå Asio-biblioteket. Istedenfor å benytte meg av funksjonene asio.cpp, lager jeg egne instanser av IASIO og sender funksjonskallene direkte til driverne.

Disse testene kan man kjøre ved å lese Appendiks A - Testing med flere lydkort-instanser. Merk at testen begynner først når brukeren har valgt begge driverne. Sørg for at begge lydkortene er tilkoblet og påskrudd før du oppgir den siste driveren. Det har ikke noe å si om man kobler inn et lydkort etter at man har startet programmet, men de må være tilkoblet før testen begynner.

Testing av et enkelt aggregat

Hensikten med dette forsøket er å se hva som skjer når man ønsker å sende lyd fra ett lydkort til et annet.

Dette programmet krysskobler to lydkort slik som på illustrasjonen under. Programmet sender input 1 fra hvert lydkort til output 1 og 2 på begge lydkortene. Merk at hver enhet også setter opp en intern ruting. Dette er for å verifisere at program og maskinvare er i orden og kan utføre grunnleggende funksjonalitet slik som forventet.



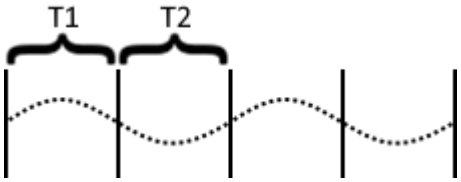
Figur 2 Signalgangen i programmet "Testing av et enkelt aggregat"

Det er tilstrebet å lage denne prosessen så enkel som mulig. Det benyttes et mellomlager til å holde på inndataene. Når mottaker-lydkortet kjører sin prosessering, leser lydkortet av dataene i dette mellomlageret og sender det ut i sine outputs. Mellomlageret er beskyttet av en lås (mutex) som hindrer det ene lydkortet i å overskrive dataene samtidig som det andre lydkortet leser av disse dataene. Hensikten er å forhindre race-conditions som kan forårsake forsinkelser i lydprosesseringen.

For å kjøre denne testen, les Appendiks A - Testing av et enkelt aggregat.

Buffernes varierende tidsintervaller

I dette forsøket ønsker vi å måle variasjonen i tidsintervallet mellom hvert kall til *bufferSwitch*-funksjonen. Det refereres til illustrasjonen under for å tydeliggjøre hva vi ønsker å måle i dette forsøket. Denne testen gjennomføres på kun ett lydkort om gangen.



Vi ser fire sekvensielle buffere som vi mottar fra lydkortet, avgrenset av de vertikale strekene. De vertikale strekene viser også tidspunktet for hvert kall til bufferSwitch. BufferSwitch er funksjonen der lydprosesseringen skjer. Den påkalles av lydkortets driver hver gang lydkortet har fylt opp input-bufferne sine og er klar til å ta imot nye data i output-bufferne sine.

T1 er tidsmengden mellom den første bufferens funksjonskall og den neste bufferens funksjonskall. T2 representerer det samme, bare med utgangspunkt i buffer nummer to. Vi ønsker å måle forskjellen mellom T1 og T2 for å se om tidsintervallene varierer.

Programmet måler minimum, gjennomsnittlig, og maksimal forskjell mellom to tidsintervaller. Denne målingen gjennomføres over et halvt sekund, og skrives til terminalen rett etter hver måling. Tidspunktene hentes fra standardbiblioteket Chrono ved å kalle på funksjonen ‘std::chrono::system_clock::now()’.

Under testen spilles det av en sinustone fra output 1 og 2 for å verifisere at lydprosesseringen kjører uten feil.

For å kjøre denne testen, les Appendiks A - Buffernes varierende tidsintervaller.

Test 1 – Motu

Inngangsvariabler:

- Maskinvare: Motu M2
- Samplingsfrekvens: 44100 samples per sekund
- Bufferstørrelse: 64 samples

Resultat:

Tidspunkt (s)	Minimum (µs)	Gjennomsnitt (µs)	Maksimum (µs)
0.5	0.1	106.2	739.9
1.0	0.0	61.9	970.8
1.5	0.0	114.1	887.9
2.0	0.0	114.0	944.9
2.5	0.0	116.4	955.0
3.0	0.3	60.6	967.3
3.5	0.1	77.9	1122.5
4.0	0.0	30.9	756.1
4.5	0.0	90.6	634.0
5.0	0.1	193.5	802.9
5.5	0.0	155.7	993.2
6.0	0.1	59.3	887.6
6.5	0.0	539.1	728.0
7.0	0.3	88.6	848.3
7.5	0.3	109.9	868.3
8.0	0.0	96.3	895.9
8.5	0.1	128.9	871.7
9.0	0.0	43.2	853.4
9.5	0.0	92.9	981.7
10.0	0.0	115.6	622.2
10.5	0.1	65.5	843.6
11.0	0.0	114.4	244.7
11.5	0.1	125.8	227.1
12.0	0.2	115.3	366.8
12.5	0.2	129.6	787.9
13.0	0.1	120.8	789.8
13.5	0.0	206.9	937.0
14.0	0.0	192.6	986.6
14.5	0.0	172.3	741.9
15.0	0.1	114.2	1183.9
15.5	0.0	114.1	1268.3
16.0	0.6	129.7	247.3
16.5	0.0	117.3	882.0
17.0	0.2	110.2	760.5
17.5	0.0	120.2	749.9
18.0	0.0	121.9	978.0
18.5	0.0	103.1	959.8
19.0	0.1	105.3	1193.1
19.5	0.0	194.4	821.7
20.0	0.5	130.8	760.4

Test 2 – Behringer u-phoria umc1820

Inngangsvariabler:

- Maskinvare: Behringer u-phoria umc1820
- Samplingsfrekvens: 44100 samples per sekund
- Bufferstørrelse: 64 samples

Resultat:

Tidspunkt (s)	Minimum (µs)	Gjennomsnitt (µs)	Maksimum (µs)
0.5	0.1	985.6	1468.9
1.0	0.1	983.5	1452.4
1.5	0.0	1001.9	1660.1
2.0	0.0	997.3	1091.2
2.5	0.1	879.1	1460.8
3.0	0.1	511.3	1874.6
3.5	0.2	971.0	1944.4
4.0	0.5	873.1	1100.3
4.5	0.0	985.3	1450.5
5.0	0.0	994.9	1495.6
5.5	0.1	876.5	1448.8
6.0	0.1	508.4	1654.3
6.5	0.1	990.2	1569.7
7.0	0.0	869.8	1464.6
7.5	0.1	998.6	1051.5
8.0	0.0	983.2	1059.6
8.5	0.3	873.2	1453.8
9.0	0.0	1009.3	1454.7
9.5	0.0	987.5	1557.1
10.0	0.0	867.9	1469.9
10.5	0.2	996.8	1825.7
11.0	0.1	985.3	1542.2
11.5	0.0	752.1	1480.1
12.0	0.0	1004.6	2004.8
12.5	0.2	967.6	1116.3
13.0	0.1	942.0	1459.6
13.5	0.0	992.9	1463.4
14.0	0.4	992.7	1485.7
14.5	0.1	500.4	1069.1
15.0	0.2	1006.0	1490.3
15.5	0.0	964.5	1567.6
16.0	0.0	504.3	1546.3
16.5	0.2	989.3	1798.4
17.0	0.0	966.9	1755.4
17.5	0.0	872.2	1463.9
18.0	0.0	994.7	1991.5
18.5	0.2	925.8	1462.5
19.0	0.0	505.6	1648.0
19.5	0.0	998.1	1459.5
20.0	0.3	970.0	1546.6

Diskusjon

Vurdering av eget arbeid

I dette delkapittelet vurderer jeg kvaliteten av det arbeidet som er gjort.

Testene inneholder verifisering av grunnleggende funksjonalitet

En positiv egenskap ved programvaretestene, er at de også verifiserer at program- og maskinvare kjører som forventet.

Testene spiller av en sinustone samtidig som testen pågår. Ved å lytte til sinustonen kan man raskt oppdage uforutsette feil i lydprosesseringen. Programvaren i *Testing av et enkelt aggregat* spiller ikke av en sinustone. Isteden er det satt opp en ekstra lydruting internt i lydkortet, slik at man kan teste denne rutingen for å verifisere at alt fungerer som det skal.

Det lurt å ha disse kontrollpostene fordi vi tester bruksområder maskinvarene ikke er laget for. Altså forventer vi å finne oppførsel som vi i utgangspunktet ikke kan forklare. Problemet er at slik oppførsel også kan være forårsaket av oss selv. Derfor lager vi disse verifiserings-tiltakene, nemlig for å gjøre det lettere å skille mellom uforklarlig oppførsel som skapes av oss selv, og oppførsel som viser svakheter ved maskinvaren.

Burde gjennomført tester på flere enheter

En svakhet ved forsøkene i dette prosjektet er at de kun er utført på to forskjellige maskinvarer. Denne svakheten gjør at diskusjonen rundt resultatene vil være mindre produktiv, ettersom at observasjonene ikke nødvendigvis er universelle for all maskinvare.

Diskusjon av resultater

I dette delkapittelet legger jeg frem min egen tolkning av – og diskuterer følgene av – resultatene i dette prosjektet. Delkapitlene diskuterer resultatene fra resultat-kapittelet med samme navn som dem selv.

Asio4All

I tiden til utformingen av denne rapporten, finnes det ingen offisiell metode for å lage kombinerte lydenheter til profesjonelle lydenheter på Windows.

Når man søker etter en løsning på internett, blir man anbefalt til å bruke Asio4All. Problemet med Asio4All er det faktum at aggregat-funksjonaliteten er implisitt. Dette betyr at den ikke støttes eksplisitt av utviklerne. Slik implisitt funksjonalitet opptrer ofte feilaktig, og dette kan vi se resultatet av i testene som er utført på Asio4All i dette prosjektet.

Asio-biblioteket støtter ikke bruken av flere lydkort

Praksisen av å bruke en Asio-driver om gangen er nokså utbredt. DAW-programmer som Logic, ProTools, Ableton, Reaper – som kan betegnes som de mest kjente studio-programvarene – benytter denne praksisen. Når alle de mest kjente og suksessfulle programmene gjør det samme, kan det virke som at utviklerne i hvert selskap har aktivt vurdert dette til å være den beste praksisen.

Det finnes to gode argumenter for at avgjørelsen heller ligger hos protokollens skapere, nemlig Steinberg som er skaperen og forvalteren av Asio-protokollen[3].Det ene argumentet baserer seg på det som står i Asio-bibliotekets kildekode. Det andre baserer seg på en standard praksis i programvareutvikling.

Det holder ikke å bare vise til egenskaper ved Asio-biblioteket for å argumentere for slikt. Jeg har vist at det er mulig å bruke en Asio-driver i egen programvare uten Asio-biblioteket. Biblioteket er kun én implementasjon av Asio-protokollen, og det er ingenting som hindrer utviklere i å lage sin egen. Likevel har Steinberg autoritet innenfor fagfeltet som forvalteren av protokollen, hvilket gjør at andre utviklere vil anse deres implementasjon som den offisielle og korrekte.

theAsioDriver er en singleton

Kildekoden gir et veldig tydelig signal om at man kun får lov til å bruke ett Asio-lydkort per programinstans.

På linje 36 i filen asio.cpp, defineres variabelen *theAsioDriver* som er av typen IASIO. Datatypen IASIO er protokollens digitale representasjon av et lydkort.

Denne variabelen er definert som en 'singleton'. En singleton-variabel forventes å være programmets eneste instans av en gitt datatype. Siden datatypen i dette tilfellet er *protokollens digitale representasjon av et lydkort*, betyr det at utviklerne av Asio gir oss et tydelig signal om at man ikke skal bruke flere lydkort samtidig.

[3] Steinberg (u.å.)

Begrunnelse for at theAsioDriver er definert som en singleton

Singleton er et begrep fra objekt-orientert programmering. Asio-biblioteket er skrevet i programmeringsspråket C, som ikke er et objekt-orientert språk. Likevel ser vi at variabelen brukes på samme måte som en singleton i andre programmeringsspråk.

For det første, variabelen er definert i programmets globale kontekst. Det betyr at variabelen er felles for hele programinstansen. Det er ikke lagt opp noen måte å lage flere instanser av denne variabeltypen gjennom de funksjonene som er oppgitt i dokumentasjonen.

For det andre, alle funksjonene som er oppgitt i dokumentasjonen ligger også i den globale konteksten av programmet. I tillegg, hver av dem viderefører sitt funksjonskall til en metode i *theAsioDriver* med samme navn som dem selv, med unntak av at funksjonene starter med «ASIO». Denne egenskapen er en av prinsippene til en singleton, nemlig at man gjemmer vekk instansen bak globale funksjoner.

Konseptet om abstrahering

Det andre argumentet baserer seg på en kjent praksis i programvareutvikling, nemlig konseptet om abstrahering. Leseren bør bemerke seg at argumentasjonen i dette delkapittelet avviker fra dette prosjektets erklærte metode. Argumentet er bygget på spekulasjon utfra en standard praksis i programvareutvikling, og bør sees i lys av denne svakheten.

Konseptet om abstrahering refererer til praksisen av å omdefinere en sekvens av operasjoner om til én jobb. Et praktisk eksempel på dette, er det å dra på en handletur. En typisk handletur innebærer å komme seg til butikken, finne det ene, finne det andre, betale, og deretter dra hjem for å legge fra seg varene. Istedenfor å si at du skal gjøre alle disse handlingene, er det lettere å definere denne sekvensen av handlinger i ett enkelt begrep; nemlig en «handletur». Asio-biblioteket er en ansamling av slike begreper. For eksempel, funksjonen for å bytte samlingsfrekvens krever både at driveren advarer andre programmer om forandringen, og at driveren forandrer samplingsfrekvens på den fysiske enheten.

Følgene av å anvende dette konseptet er at man blender seg selv fra å se ned i detaljene. Det er ikke vanlig å lese kildekoden til de bibliotekene man bruker i sin egen programvare. Dette gjør at reglene som blir satt i slike biblioteker gjerne følges blindt av de som bruker disse bibliotekene.

Et eksempel på at dette også gjelder for Asio-protokollen er PortAudio. PortAudio krever at utviklere som ønsker støtte for Asio-drivere, må laste ned Asio-biblioteket og inkludere det i PortAudio-biblioteket på egenhånd. Her legger PortAudio all sin lit til Steinbergs kildekode istedenfor å lage en egen løsning som inkluderer Asio blant PortAudio sine støttede protokoller.

Anvendelsen av denne praksisen, gjør det fornuftig å spekulere i at utviklerne for de mest kjente DAW-programmene ikke har gjort et *aktivt* valg om å begrense antall Asio-enheter per programinstans. Dette valget ligger hos Steinberg, og har spredt seg til de mange programmer som benytter seg av Asio.

Bemanne flere lydkort samtidig

De to programmene i dette forsøket har en enkel men veldig viktig betydning. I korte trekk beviser de at det er mulig å bemanne to lydkort samtidig i samme programvare. De viser også at det er mulig å kjøre lydprosessering fra to lydkort i samme programvare.

Det bør poengteres at disse testene ikke viser er hvorvidt to lydkort kan jobbe sammen. Det at disse to lydkortene styres fra samme brukergrensesnitt i det siste programmet, kan gi en falsk oppfatning om at lydkortene jobber sammen i samme prosess. Slik er det ikke. De kjører to helt separate prosesser underlagt samme program, bedre kjent som tråder.

Likevel har dette beviset en viktig betydning. Det beviser at Windows ikke hindrer oss i å bruke flere Asio-drivere i samme program. Betydningen av dette er at det ikke er operativsystemet som har feil og mangler. Utfordringen vår ligger på programvarenivå, nemlig det å synkronisere dataene fra to lydkort.

Videre viser det siste programmet en annen ting. Hvis man ønsker å koble sammen to lydkort og ikke er avhengig av å kunne sende lyd mellom dem, trenger man ingen spesiell synkronisering av hverken klokker eller lydstrømmer for å gjøre dette. Det eneste som kreves er at programvaren støtter en slik funksjonalitet.

Testing av et enkelt aggregat

Det vi oppdager fra denne testen er en forskjell mellom intern signalruting, og signalruting mellom lydkort. Den interne rutingen i hvert lydkort fungerer uten problemer. Signalrutingen mellom lydkortene fungerer derimot veldig dårlig. Det dukker opp mye digitalt støy ved faste intervaller, men det er perioder der signalet er rent og uten denne støyen. En interessant egenskap ved denne støyen, er at intensiteten virker som at den fader inn og ut. En annen egenskap ved denne støyen er at man kan forlenge perioden med rent signal ved å forstørre buffer-størrelsen.

Videre skal vi se på hvorfor denne støyen oppstår, og hva som forårsaker disse forskjellige egenskapene.

Buffernes varierende tidsintervaller

Dette forsøket gir oss et godt svar på hvorfor det oppstår mye støy når man forsøker å transportere lyd mellom to lydkort.

Hypotesen som utprøves i denne testen er som følger. Programmet fra den forrige testen forventer at lydkortene påkaller sin bufferSwitch-funksjon kun én gang. Hvis ett av lydkortene påkaller sin bufferSwitch-funksjon to ganger på rad, vil dette skape et hørbart hakk i lydstrømmen mellom lydkortene.
En slik situasjon kan oppstå på bakgrunn av køsystemet i en datamaskin. Siden en datamaskin har færre utregningskjerner enn kjørende programmer, er de nødt til å lage et køsystem for alle programmenes operasjoner. Lydprosesseringen vår påvirkes av et slikt køsystem fordi hvert lydkort kjører sin egen delprosess.
Hvis køsystemet bytter om på hvilket lydkort som kjører sin lydprosessering først, vil det oppstå et

hakk i lydstrømmen. Skjer dette ofte nok vil hakkene oppfattes som kontinuerlig støy, akkurat slik gestaltprinsippene forteller oss.

Resultatene bryter med det jeg forventet. Jeg forventet at denne testen ville falsifisere hypotesen, slik at vi ville se nokså lave verdier i resultatet. Derimot viser tallene at det er store variasjoner i tidsintervallet mellom hver prosesseringssyklus.

Ugyldiggjør en synkroniseringsmetode

Dette resultatet ugyldiggjør en spesifikk synkroniseringsmetode.

For å finne ut lydstrømmenes forskyvning i forhold til hverandre, kan man sammenlikne de tidspunktene der lydkortene påkalte bufferSwitch. Dette gjøres da ved å registrere tiden i starten av funksjonen. Hvis målingenes error-verdi er mindre enn tidsintervallet mellom hvert sample, kan man finne lydstrømmenes forskyvning i antall samples. Dermed kan man rette opp i forskyvningen og forhindre hakk i lydstrømmen.

Problemet er at dataene vi har samlet i denne testen viser oss at nøyaktigheten er for dårlig. Vi ser maksimale error-verdier på opp mot 2 millisekunder. I en lydstrøm på 44.1kHz, trenger en maksimal error-verdi på 22.7 mikrosekunder for å kunne nå en nøyaktighet som ligger på sample-nivå. Dette betyr at vi ikke kan basere oss på denne typen tidsmålinger når vi skal synkronisere lydstrømmer fra to forskjellige lydkort.

Konklusjon

Målet for dette prosjektet har vært å kartlegge de teknologiske utfordringene en kombinert lyddriver for Asio-lydkort må ta høyde for, med den hensikt å bidra i målet om å skape en pålitelig kombinert lyddriver for Windows-brukere.

Vi ser at utfordringen ikke ligger i hvorvidt operativsystemet støtter en slik funksjonalitet, men heller i måten man synkroniserer de forskjellige lydstrømmene. Denne utfordringen kompliseres av at det er vanskelig å basere løsningen på tidsmålinger.

Det som gir håp, er at man kan jobbe på programvarenivå for å løse problemet. På dette nivået finnes det utallige ressurser for digital lydbehandling, hvilket utvider horisonten av mulige løsninger. Selv om utfordringen er komplisert, viser dette arbeidet at det absolutt er mulig å skape en pålitelig kombinert lyddriver for Windows-brukere.

Kilder

- PortAudio. (u.å.). *Building Portaudio for Windows with ASIO support using MSVC*. portaudio.com. Lastet ned 08.04.2023 fra: http://portaudio.com/docs/v19-doxydocs/compile_windows_asio_msvc.html
- Steinberg. (u.å.). *ASIO Driver*. steinberg.help. Lastet ned 08.04.2023 fra: https://steinberg.help/wavelab_elements/v11/en/wavelab/topics/setting_up_your_system/asio_driver_c.html
- Steinberg. (u.å.) ASIO 2.3 (#3).
  Dokumentet ligger i zippet mappe fra https://www.steinberg.net/asiosdk under filnavn «ASIO SDK 2.3.pdf».

Appendiks A – Programvaretester

Alle programvaretestene ligger som kjørbare filer i mappen *Programvaretester*.

Merk at antivirusprogrammer muligens vil stoppe programmene fra å kjøre. Dette er fordi filene ikke er signerte med et sertifikat, og alle antivirusprogrammer skal reagere på dette. Om man ønsker å bygge prosjektet selv, er kildekoden vedlagt med nødvendige makefiles. Jeg anbefaler å bruke CMake til å bygge prosjektet.

Testing med flere lydkort-instanser

Begge programmene ligger i mappen *Programvaretester*.

Programmet som åpner konfigurasjonspanelene til to lydkort, heter *Open_configuration_panel.exe*.

Programmet som spiller av en sinustone på to lydkort, heter *Parallel_sine.exe*.

Testing av et enkelt aggregat

Programmet ligger i mappen *Programvaretester* og heter *Aggregate_Demonstration.exe*.

Buffernes varierende tidsintervaller

Programmet ligger i mappen *Programvaretester* og heter *Time_Variation_Test.exe*.

Appendiks B – Vedlegg

Testing av kombinert lyddriver med Asio4All

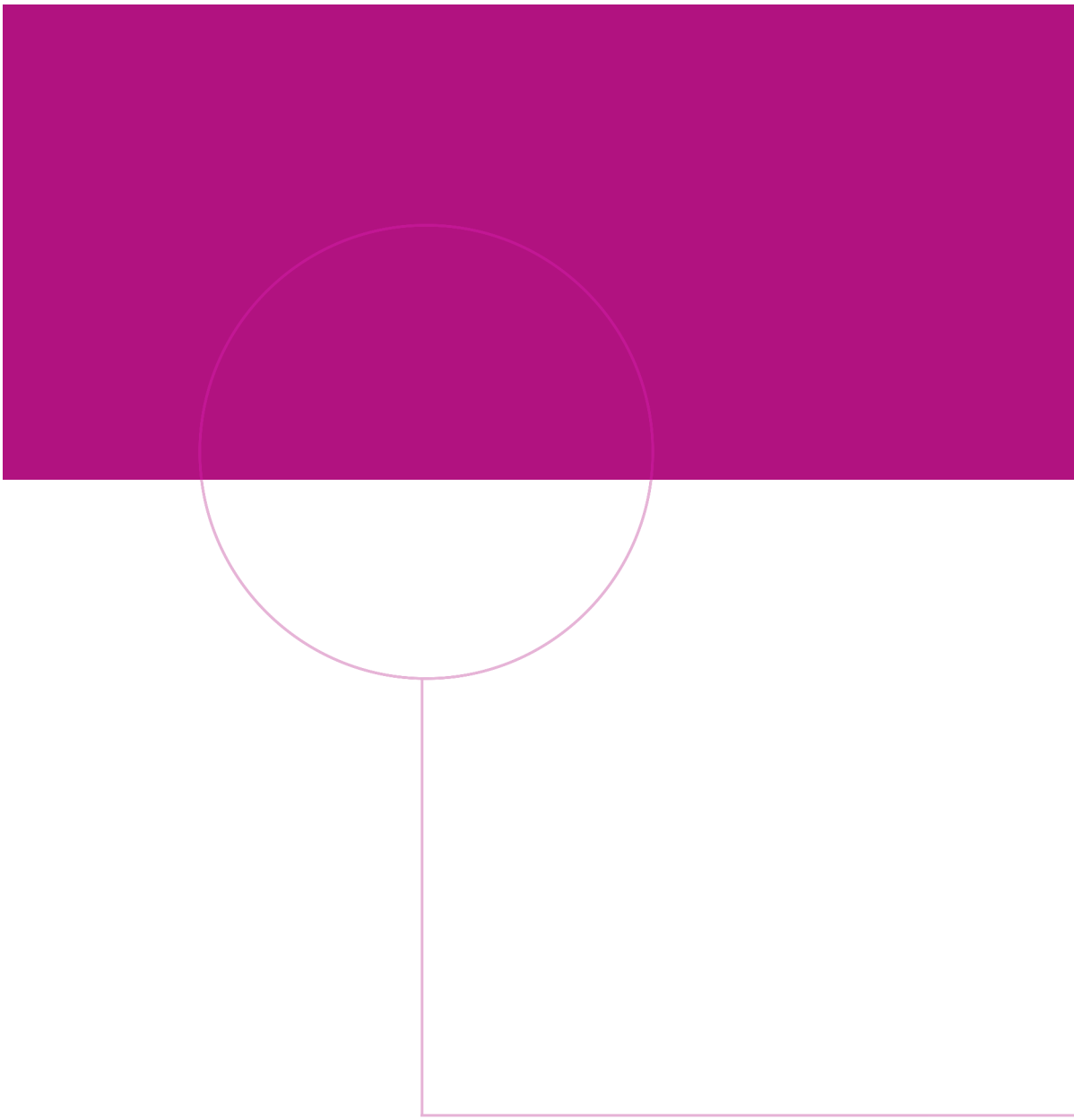
Denne filen ligger i mappen *Lydeksempler* og heter *Aio4All_Test.wav*.

Kildekoden

Kildekoden ligger i en komprimert mappe som heter *Kildekode.zip*.

ASIO_SDK

Asio-biblioteket er vedlagt som en komprimert mappe som heter *ASIO_SDK.zip*.



NTNU

Kunnskap for en bedre verden