Laboration 6

I denna laboration ska ni fortsätta med mer hårdvarunära programmering och nu använda Arduino för lite mer spännande signalbehandling. Kommer ni inte ihåg hur Arduinot är konfigurerat eller hur shieldet är kopplat, läs i handledningen till laboration 2.

Inför laborationen

Ni ska med hjälp av Arduinot skapa en realtidseffekt, först en overdrive/distorsion och sedan väljer ni att göra antingen en flanger- eller en ringmodulatoreffekt. Därefter ska ni göra ett enkelt reverb. Blir uppgiften för enkel gör ni ljudeffekten bättre och/eller smartare, eller så gör ni mer än en typ av ljudeffekt.

Förberedelse – 1. Granska befintlig kod och kommentarer

Granska befintlig kod med kommentarer innan laborationen börjar. Använd antingen en vanlig texteditor eller Arduino IDE. Bekanta er med den befintliga koden och strukturen. Börja med de olika globala variablerna, kolla sedan på `Timer2`-interrupten (som ligger längst ned i koden). Kolla sedan på den korta och torftiga `void loop`-funktionen.

All (egentlig) kod för denna laboration kommer att skrivas i `void loop`.

Förberedelse – 2. Läs på om effekttyperna

Läs på om hur en overdrive fungerar och hur man kan skapa overdriven i kod. Testa gärna en version i Matlab inför laborationen. Läs också på hur en flanger respektive en ringmodulator fungerar, samt olika sätt att digitalt skapa ett reverb.

Uppgiften

För att testa ljudeffekterna behöver ni koppla ihop ett Arduino med en av de analoga småsyntarna eller er mobiltelefon eller ljudutgången från er dator och en av högtalarna. Det är lite ont om utrustning tyvärr, så samsas med de övriga är ni snälla.

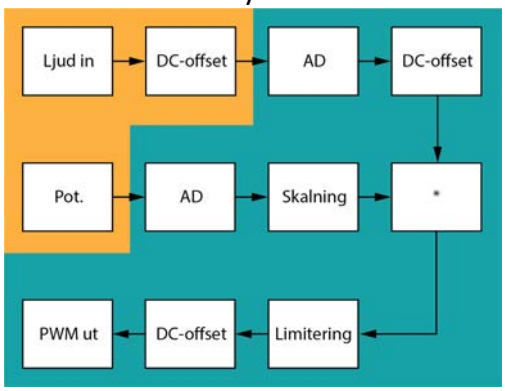
Testa loopbackfunktionen

Börja med att skicka det samplade ljudet direkt till utgången. Minns ni inte hur ni gjorde det, kolla i er kod från laboration 4.

Om allt funkar som det ska och ni har kopplat det hela rätt ska ni höra ljudet, samplat genom Arduinot i högtalaren.

Skapa en overdrive

Även om det kan tyckas vara lite onödigt att göra en overdrive av en 8-bitars samplad signal med ett Arduino är det trots allt en ganska trevlig inkörsport till denna laboration. Vi kommer att använda några globala variabler, fixa DC-offset och hårdlimitera ljudet, samt skriva det bearbetade ljudet till utgången igen.



Genomgående i laborationen har jag försökt att använda `soundSampleFromADC` som en samplecontainer för inkommande ljudsample och `sramBufferSampleValue` som en samplecontainer för utgående ljudsample. Det innebär dock att det

blir lite krystat med dessa variabelnamn ibland. Döp variablerna smartare än vad jag har gjort om ni vill.

Stoppa först inkommande ljudsample i `soundSampleFromADC` och ta bort DC-offseten. Tänk efter var DC-offseten borde ligga om vi har samplat med 8 bitar och ingångsnivån var mellan 0 och +5V.

Värdet på potentiometern stopps i variabeln `badc1`, kolla i `Timer2`-interrupten ifall ni inte minns sedan de tidigare Arduinolaborationerna. Värdet på `badc1` är ett 8-bitarsvärde. Multiplicera `soundSampleFromADC` med det värdet men skala om det till en fjärdedel, se till att få ut ett decimaltal och se också till att det som lägst är 1 (om `badc1` är 0 så får ni ju inte ut något ljud alls…).

Nu ska ljudsampeln hårdlimiteras för att undvika att utgången överstyr. Vi vill förvisso ha overdrive, men inte fulöverstyrning av ljudutgången eller av Arduinots PWM-utgång. Eftersom ni har justerat DC-offseten ska ni skriva två `if`-satser som tar ljud över max och under min, och sätter dem till max respektive min.

Avslutningsvis ska ni lyfta upp ljudsampeln till rätt DC-offset igen och lägga det i `sramBufferSampleValue`, för att sedan skicka det till PWM-utgången `OCR2A`.

Om allt funkar som det ska och ni har kopplat det hela rätt ska ni höra ljudet, samplat och med inställbar mängd overdrive/distorsion i högtalaren.

Vad sker med ljudet vad gäller dynamiskt omfång och frekvensinnehåll när ni kör det genom overdriven?

_____

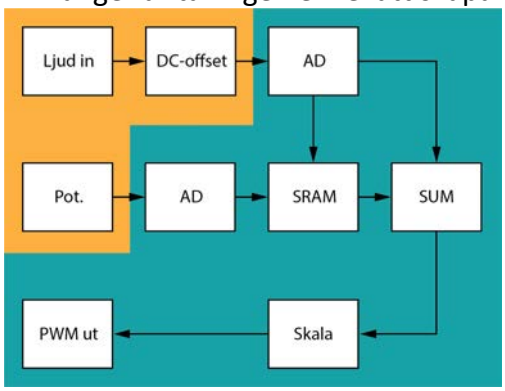
_____

_____

Välj sedan en av följande effekter att jobba vidare med: flanger eller ringmodulator.

Skapa en flanger

En flanger är tämligen enkel att skapa utifrån detta ganska enkla ramverk. Börja med att kommentera bort koden till overdriven, eller spara om projektet i ett nytt filnamn.



Kort uttryckt så fungerar en flanger genom att originalljudet mixas med en fördröjd signal av densamma, och där mängden av fördröjning varieras. När originalljud och fördröjt ljud summeras uppstår kamfiltereffekter med fasutsläckning respektive fasförstärkning, vilket varierar övertonsserien i ljudet.

Därför ska vi använda `sramBuffer` för att spara en kort snutt av ljudet, och vi kommer att variera tiden vi skjuter på uppspelningen av det lagrade ljudet. Alltså kommer `sramBuffer` kontinuerligt att fyllas och skrivas över, och vi kan inte försena ljudet mer än 512 samplingar. Kolla upp illustrationen för SRAM-buffer från laboration 2.

Stoppa först inkommande ljudsample i `soundSampleFromADC` och skriv den till `sramBuffer` på position `bufferIndex`. Läs sedan position `bufferIndex2` från `sramBuffer` och stoppa det samplevärdet i `soundSampleFromSramBuffer`.

Räkna sedan upp `bufferIndex`, och sätt `bufferIndex2` till `bufferIndex` med fördröjningen enligt potentiometerns värde, `badc1`.

Eftersom `sramBuffer` har 512 positioner måste vi anpassa `bufferIndex` och `bufferIndex2` så att dessa inte blir för höga. Detta görs enkelt med modulo (`%`) för `bufferIndex`, och med bitwise AND (`&`) för `bufferIndex2`. Dessa två metoder är lite olika, fundera på vad skillnaden är och varför dessa två olika sätt används.

_____

_____

_____

Läs mer här:
http://arduino.cc/en/Reference/Modulo
https://www.arduino.cc/reference/en/language/structure/bitwise-operators/bitwiseand/

När våra index är korrekta ska de två ljudsamplen, originalsamplen och den fördröjda sampeln, summeras. Kom ihåg att fixa DC-offseten när ni summerar, och tänk på att anpassa ljudnivån så att det inte överstyr då ni summerar två samples. Därefter skickar ni det summerade ljudet, dvs `sramBufferSampleValue`, till utgången.

Om allt funkar som det ska och ni har kopplat det hela rätt ska ni höra ljudet, samplat och med inställbar fördröjning av det buffrade ljudet i högtalaren. Det kommer att låta som ett kamfilter, dvs en viss utsläckning av vissa frekvenser, om ni inte vrider potentiometern fram och tillbaka och skapar den riktiga flangereffekten för hand.

Varför är flangereffekten så liten vid ytterpunkterna (max och min) på potentiometern?

_____

_____

_____

Fundera på hur detta skulle gå att snygga till och skapa en automatisk flangereffekt. I stället skulle i så fall potentiometerns värde kunna användas till att ställa hur mycket effekt som ska användas, eller till att sätta frekvensen på LFOn som skapar den automatiska effekten.

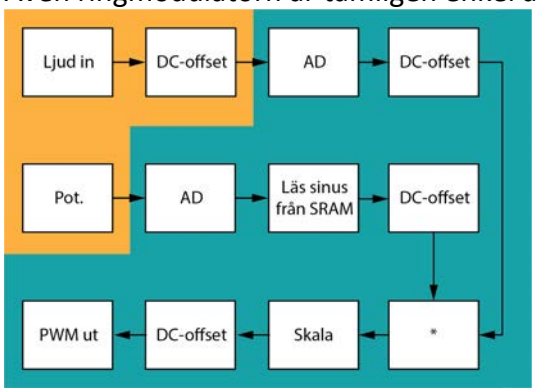
_____

_____

_____

Skapa en ringmodulator

Även ringmodulatorn är tämligen enkel att skapa utifrån detta ramverk. Börja med att kommentera bort koden från föregående övning, eller spara om projektet i ett nytt filnamn. I denna effekt behöver ni ta hänsyn till DC-offseten.



Stoppa först inkommande ljudsample i `soundSampleFromADC` och ta bort DC-offseten. Läs sedan sinusvågen som ni skapade i laboration 2 på position `bufferIndex` till `sramBufferSampleValue` och ta bort DC-offset även från den. Gör så att potentiometerns värde sätter frekvensen på sinusvågen.

Ringmodulation är detsamma som en multiplikation av två vågformer, dvs en frekvensmix. Multiplicera därför de två ljudsamplena med varandra. Sedan måste ni skala om till max i 8-bitars tal, och därefter lägga till DC-offseten igen. Stoppa summan i utgångsvariabeln. När de två vågformerna, eller de två samplevärdena i detta fall, är multiplicerade med varandra så har vi skapat en enkel ringmodulator.

Avslutningsvis ska ni skicka det nya ljudet, dvs `sramBufferSampleValue`, till utgången.

Om allt funkar som det ska och ni har kopplat det hela rätt ska ni höra ljudet, samplat och med inställbar frekvens av ringmodulationen. Det kommer nog inte att låta så vackert, men härligt knasigt ringmodulerat.

Lyssna på ringmodulatoreffekten. Hur påverkar sinusvågens frekvens ljudet som kommer ut från ringmodulatorn? Varför?

_____

_____

_____

Som det är nu hörs sinustonen fast den egentligen inte borde göra det. Om bara en signal kommer till ringmodulatorn, sinustonen i det här fallet, så borde det vara sinusvågens värde multiplicerat med 0. Vad kan vara anledningen till varför sinustonen hörs? Hur kan koden förbättras för att sinustonen inte ska höras om det inte är något ljud på ljudingången?

_____

_____

_____

Fundera på hur ringmodulatoreffekten kunde förbättras. Kanske kunde mängden av påverkan av sinusvågen påverkas, men hur skulle detta kunna lösas och vilken effekt på slutljudet skulle det få?

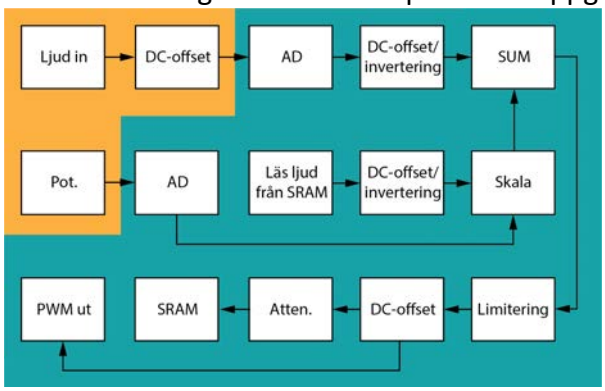
_____

_____

_____

Skapa ett reverb

Reverbet är nog den mest komplicerade uppgiften att skapa utifrån detta ramverk, men personligen tycker jag att det också är det mest belönande. Det är till och med så att jag har funderat på att faktiskt bygga in reverbet i en av modularsyntarna. Börja med att kommentera bort koden från föregående övning, eller spara om projektet i ett nytt filnamn. I denna effekt behöver ni ta hänsyn till DC-offseten i 8-bitars samplevärdet.



Läs först position `bufferIndex` från `sramBuffer` och stoppa det samplevärdet i `soundSampleFromSramBuffer`. Ta bort DC-offseten från samplevärdet från buffern. Men gör detta och vänd samtidigt på fasen. Klura lite extra på detta steg, skillnaden mellan att bara ta bort DC-offset som vi gjort i tidigare övningar, och att både ta bort DC-offset och vända fasen är faktiskt väldigt liten.

_____

_____

_____

Sedan ska ni förändra värdet på samplen från SRAM-buffern genom att multiplicera det med värdet från potentiometern, `badc1`, och sedan skala om så att det nya samplevärdet inte överstiger 255. Egentligen vill vi inte ha 255 för då kommer reverbljudet aldrig att dämpas och klinga av, utan reverbet kommer i stället att bli en loopenhet. Använd `map` på ett liknande sätt som ni gjorde för alphavärdet i laboration 4, och använd 0 till 240 (eller 250 prova er fram). I det här fallet vill ni använda er av heltal, `soundSampleFromSramBuffer` är en `int` och `badc1` är en `int`, och 255 är också en `int`.

Stoppa sedan inkommande ljudsample i `soundSampleFromADC` och ta bort DC-offseten, och summera inkommande ljudsample med den skalade samplen från SRAM-buffer. Nu finns risken att ljudet överstyr eftersom två signaler summeras. Att dividera med 2 för att undvika överstyrning skulle i detta fall inte fungera på ett bra sätt, då vi vill pressa upp nivån på ljudet så att rumseffekten blir tydlig. Gör därför en hård limitering på samma sätt som för overdriven ovan.

Fundera på hur resultatet skulle låta om division hade använts i stället för limitering. Testa gärna att ändra limiteringen när ni är klara med hela reverbkoden. Vad förändras i ljudet? Hur låter det?

_____

_____

_____

Därefter ska ni lägga till DC-offset igen och spara det nya samplevärdet i `sramBuffer` på position `bufferIndex`. Tänk nu efter hur detta reverb fungerar. Varje sample blandas med en tidigare sample och sparas sedan ned i buffern igen, så efter ett kort tag kommer den

första samplen att blandas med en ny sample och sparas, och sedan kommer den blandningen att blandas med en ny sample och sparas, och runt och runt och runt.

Avslutningsvis ska `bufferIndex` räknas upp och begränsas med modulo (`%`) så att inte indexet överstiger antalet positioner i `sramBuffer`, och samplevärdet i `sramBufferSampleValue` ska skickas till utgången.

Om allt funkar som det ska och ni har kopplat det hela rätt ska ni höra ljudet, samplat och med inställbar mängd/längd av reverbet.

Fundera på om reverbljudet kunde förbättras? Kanske kunde den ljudsample som läses från buffern lågpassfiltreras för ett mjukare reverbljud? Hur kunde ett större reverb kunna skapas, med längre efterklangstid?

_____

_____

_____

Om ni har gjort allt och ändå har tid kvar

Ett delay bygger på liknande teknik som en flanger, men med längre tidsskillnad mellan originalljud och fördröjt ljud. Hur skulle det vara möjligt att skapa ett delay? Vilka utmaningar och problem skulle vi stöta på?

_____

_____

_____

Även ett chorus bygger på en liknande teknik som en flanger, men med en fast tidsskillnad mellan originalljud och effektljud. Det som skapar effekten i ett chorus är att effektljudet är i en annan pitch/frekvens (eller åtminstone en pitchmodulerad variant av originalljudet) vid summering med originalljud. Hur kunde denna effekt skapas? Kunde det pitchmodulerade ljudet skapas med hjälp av ringmodulation? Vilka utmaningar och problem skulle vi stöta på?

_____

_____

_____

Vidare läsning

Arduino
https://www.arduino.cc/

Arduino, referense för den C-aktia koden
https://www.arduino.cc/en/Reference/HomePage

Arduino, IDE (programmeringsmiljön)
https://www.arduino.cc/en/Main/Software

Secrets of Arduino PWM
https://www.arduino.cc/en/Tutorial/SecretsOfArduinoPWM

ADC (Analog To Digital Converter) of AVR Microcontroller
http://extremeelectronics.co.in/avr-tutorials/using-adc-of-avr-microcontroller/

TNM103 - Ljudteknik

The ADC of the AVR, Analog to Digital Conversion
http://maxembedded.com/2011/06/the-adc-of-the-avr/

ATmega32, datablad
http://www.atmel.com/images/doc2503.pdf

Low-level ADC control: ADMUX
http://openenergymonitor.blogspot.se/2012/08/low-level-adc-control-admux.html

CBI - Clear Bit in I/O Register
http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_CBI.html

SBI - Set Bit in I/O Register
http://www.atmel.com/webdoc/avrassembler/avrassembler.wb_sbi.html