Arduino 2 – filter

I denna laboration ska ni fortsätta med mer hårdvarunära programmering och nu använda Arduino för filtrering. Kommer ni inte ihåg hur Arduinot är konfigurerat eller hur shieldet är kopplat, läs i handledningen till Arduino 1.

Inför laborationen

Ni ska med hjälp av Arduinot skapa ett antal digitala filter som filtrerar en inkommande ljudsignal från en extern ljudkälla, samt om ni hinner era vågformer från laboration 2, och sedan spela upp ljudet i realtid där ni förändrar brytfrekvensen med en potentiometer.

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 filtertyper

Läs först igenom följande text om filtertyper:
http://blog.prosig.com/2011/10/04/understanding-filter-characteristics/

Läs sedan igenom beskrivningen av digitala filter (med fokus på IIR (infinite impulse response) filter) här:
https://en.wikibooks.org/wiki/Digital_Signal_Processing/Digital_Filters

Läs sedan igenom om moving average (med fokus på exponential moving average) här:
https://en.wikipedia.org/wiki/Moving_average

Uppgiften

För att testa filtren behöver ni koppla ihop ett Arduino (ljud in) med en minitelekabel till ljudutgången från en laptop eller mobiltelefon, och sedan ut från Arduinot till exempelvis hörlurar eller via minitelekabel till en extern högtalare.

Spela upp inkommande ljudsample

Ljudingången på shieldet samplas i 15625 Hz och det samplade ljudvärdet lagras i `badc0`. Kolla i Arduino 1 för hur ni ansluter ljud. I `void loop`, ska ni skicka det samplade ljudvärdet till ljudutgången, `OCR2A`. Vill ni göra det hela lite snyggare och mer lättläst bör ni använda den globala variabeln `soundSampleFromADC` för att ”mellanlagra” sampelvärdet. Nu ska ni kunna höra det samplade ljudet i högtalaren. Ett litet problem kan vara att ljudet förväntas ha ett sväng på fem volt, dvs mellan 0V och +5V, men det är inte säkert att den ljudkälla ni använder har en så pass stark signal. Därför får ni kanske höja ljudvolymen på högtalaren, och stå ut med lite extra kvantiseringsbrus och artefakter från 8-bitarssamplingen.

Hur låter det? Har ni kvantiseringsbruset? Hur låter frekvensomfånget i det samplade ljudet?

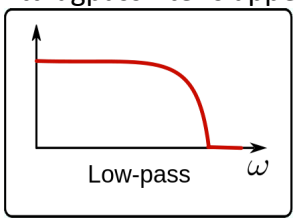
_____

_____

_____

Skapa ett lågpassfilter

Ett lågpassfilter släpper igenom låga frekvenser och dämpar höga frekvenser. Börja med att skapa två nya globala variabler. Den ena ska vara en `float` och kommer att innehålla alpha för filtret, i det här fallet kan vi jämföra det med brytfrekvensen för filtret. Den andra variabeln, en `int`, ska senare bli tilldelad den filtrerade ljudsampeln. Kom ihåg att döpa variablerna smart! ;-)



I `void loop` ska den fördefinierade variabeln `soundSampleFromADC` få det samplade värdet från ljudingången (`badc0`). Sedan ska ni tilldela potentiometerns (`badc1`) värde till variabeln ni har deklarerat och som ska innehålla filtrets alphavärde. Alphavärdet ska ligga mellan 0 och 1, men potentiometern samplas mellan 0 och 255. Använd därför `map` för att mappa värdena. Tänk på att map-funktionen används på heltal och returnerar heltal, och ni vill ha ut decimalvärden.

Läs mer här:
https://www.arduino.cc/reference/en/language/functions/math/map/

För att skapa lågpassfiltret ska medelvärdet av nuvarande sampelvärde (`soundSampleFromADC`) och föregående sampelvärde, vägt med alphavärdet, skapas och lagras i den nya variabeln som ni skapade. Utgående sampel blir sedan föregående sampel nästa varv i `void loop`.

Denna typ av filter kallas för medelvärdesfilter (exponential moving average), och medelvärdet tas mellan föregående sampel och nuvarande sampel. När en större del av föregående sampel är med i medelvärdet, kommer också mer av tidigare samples påverka medelvärdet vilket skapar en ”slätare” kurva vilket ger en lägre brytfrekvens lågpassfiltret. Detta är ett digital IIR-filter (infinite impulse response filter).

Läs mer här:
https://en.wikipedia.org/wiki/Moving_average#Exponential_moving_average
https://en.wikipedia.org/wiki/Infinite_impulse_response

På ljudspråk skulle vi kunna säga att föregående ljudsampel (`soundSampleFromADC`) och nuvarande ljudsampel (er variabel) ska mixas, och att alphavärdet är det som styr ljudnivån på dessa två ljud. Så, om alphavärdet är 0.5, ska de både ljudsamplena vara lika starka, om alpha är 0 ska bara den gamla ljudsampeln höras och om alpha är 1 ska bara den nya ljudsampeln höras.

När allt är klart ska den filtrerade ljudsampeln, dvs er variabel, skickas till utgången (`OCR2A`).

Testa filtret. Hur låter det? Är kontrollen på potentiometern naturlig eller skulle ni förvänta er att filtrets brytfrekvens ändrades åt andra hållet?

_____

_____

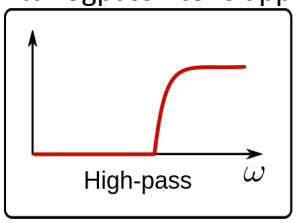
_____

Även om alphavärdet kan ligga mellan 0 och 1 så är kanske extremvärdena inte så användbara, begränsa därför min- och maxvärdet i samband med mappningen, till ungefär 4% respektive ungefär 78% av 255.

Ändra mappningen och testa hur det låter? Är det bättre jämfört mot fullt ut till 0 och 1?

Skapa ett högpassfilter

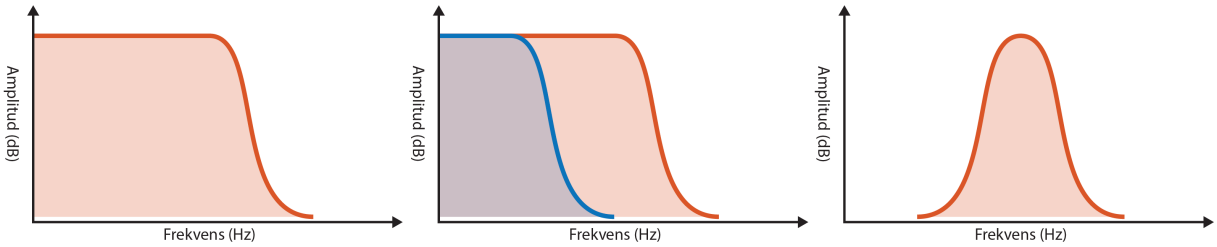
Ett högpassfilter släpper igenom höga frekvenser och dämpar låga frekvenser. Om man har tillgång till ett lågpassfilter kan detta användas för att ta bort den låga informationen från originalsignalen för att skapa ett högpassfilter. Detta kan ni enkelt göra genom att subtrahera er lågpassample från inkommande ljudsample (`soundSampleFromADC`), och sedan lagra den nya sampeln i en ny global `int` för högpass. För att detta ska fungera måste ni först korrigera DC-offseten av de två ljudsamplen, sedan subtrahera, och därefter återställa DC-offseten.



Testa högpassfiltret. Hur låter det jämfört mot lågpassfiltret? Varför fungerar brytfrekvensen tvärtom jämfört mot lågpassfiltret?

Skapa ett bandpassfilter

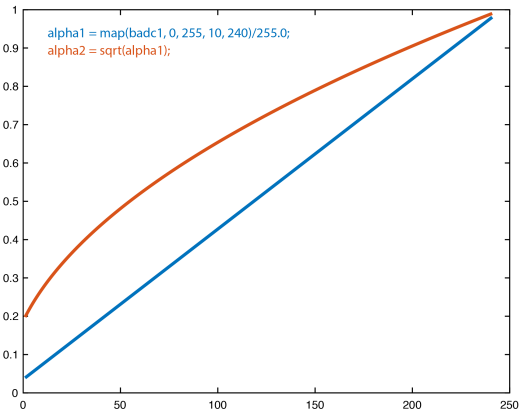
Ett bandpassfilter släpper igenom frekvenser i mitten kring brytfrekvensen, medan det dämpar höga och låga frekvenser. Liksom i Matlab behövs två värden för att skapa ett bandpassfilter. I Matlab behövs en undre och en övre brytfrekvens. I denna laboration använder vi oss inte av exakta brytfrekvenser utan vi använder alphavärdet för att ställa brytläget i filtren. Bandpassfiltret skapas sedan av två lågpassfilter där ljudsampeln från filtret med högre brytfrekvens subtraheras med ljudsampeln som är filtrerat med en lägre brytfrekvens. Då tas högfrekvent information först bort, och därefter tas de lägre frekvenserna bort.



Skapa därför en ny variabel för alpha2, och sätt den nya variabeln till kvadratroten av den gamla alphavariabeln. Kvadratroten ur ett värde räknar vi ut med `sqrt`, och detta tack vare att `math.h` är inkluderad i projektet.

Läs mer här:
https://www.arduino.cc/reference/en/language/functions/math/sqrt/

Genom att använda kvadratroten så kommer det andra alphavärdet att vara något högre än den första alphan, och därigenom sätts bandbredden för bandpassfiltret. Detta blir liknande Matlab, där två olika brytfrekvenser anges för ett bandpass- eller ett bandstoppfilter.



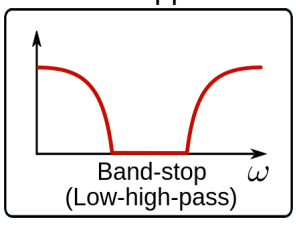
Att använda kvadratroten på detta sätt fungerar, men är kanske inte optimalt, då bandbredden på filtret kommer att bli beroende av frekvensen (alphan) för det första lågpassfiltret. Hur skulle detta kunna lösas snyggare?

Därefter ska ni skapa en ny global `int` för att spara det lågpassfiltrerade ljudet med högre brytfrekvens i. Och slutligen skapar ni ytterligare en global `int` för den bandpassfiltrerade ljudsampeln, vilken räknas ut på liknande sätt som för högpassfiltret, fast där ni tar det övre lågpassfiltret minus det första lågpassfiltret.

Testa bandpassfiltret. Hur låter det jämfört mot de andra två filtren?

Skapa ett bandstoppfilter

Ett bandstoppfilter är som ett inverterat bandpassfilter och dämpar frekvenserna kring brytfrekvensen, men släpper igenom låga respektive höga frekvenser. Ett bandstoppfilter skapas enkelt genom att ta bort den bandpassfiltrerade ljudsampeln från den inkommande ljudsampeln (`soundSampleFromADC`).



Testa bandstoppfiltret. Hur låter det jämfört mot de andra filtren?

En liten nackdel med den här enkla typen av filter, exponential moving average, som vi testar i laborationen är att vi inte vet exakt vad brytfrekvensen är i filtret. Detta går naturligtvis att räkna på, läs mer här om ni är nyfikna: https://dsp.stackexchange.com/questions/40462/exponential-moving-average-cut-off-frequency

Testa att filtrera vågformerna från den första Arduinolaborationen

Sätt ihop koden från den första arduinolaborationen med denna laboration. Ni måste då sätta en fast frekvens för pitchen på vågformen eftersom potentiometern just nu styr alphavärdet för filtret. Testa de olika filtertyperna på era olika vågformer, och testa olika frekvens på vågformerna.

Hur fungerar lågpassfiltret på de olika vågformerna, och varför?

_____

_____

Hur fungerar högpassfiltret på de olika vågformerna, och varför?

_____

_____

Hur fungerar bandpassfiltret på de olika vågformerna, och varför?

_____

_____

Hur fungerar bandstoppfiltret på de olika vågformerna, och varför?

_____

_____