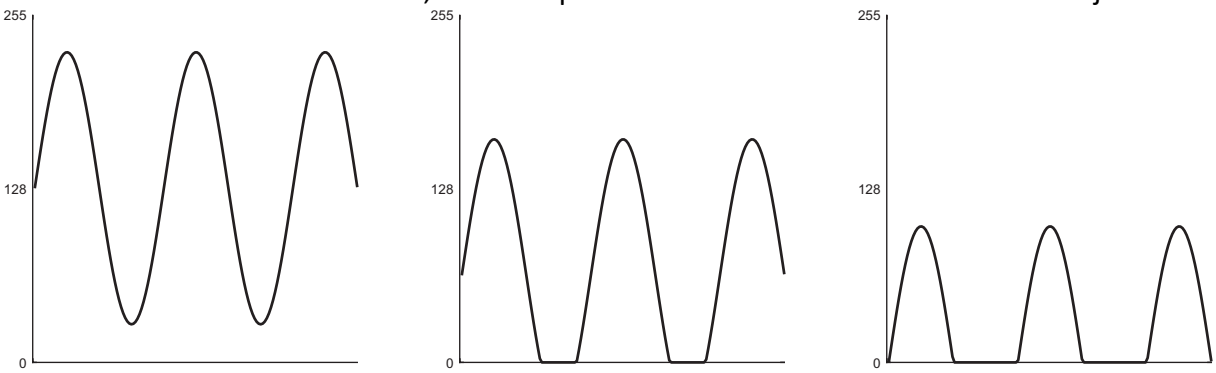
Arduino – DC-offset, precision och talområde

Arduinot har mycket enkla AD och DA-omvandlare och ont om minne, och arbetar därför med 8 bitars precision för samplingen. Egentligen kunde Arduinot arbete med 12 bitar på AD-sidan, men genom att använda 8 bitar sparas beräkningskraft och lagringsutrymme. Datatypen för att representera ett 8-bitars tal heter byte i Arduinos kompilator. Typen bör inte användas för beräkningar, eftersom den bara kan representera heltalen 0 till 255 (`00000000` till `11111111` i binär representation). Negativa tal och tal större än 255 kan inte representeras, och beräkningar som ger sådana tal, även som mellanresultat, blir alltså felaktiga om man räknar med datatypen `byte`. För beräkningar bör i stället dataypen `int` användas, ett 16-bitars heltal som kan representera både positiva och negativa tal inom ett större talområde (-16384 till 16383). Det finns datatyper med bättre precision och större talområde, men `int` är fullt tillräckligt för våra behov här. Använd alltså variabler av typen `int` för alla beräkningar, och använd datatypen `byte` endast för direkta indata och utdata för 8-bitars sampelvärden.

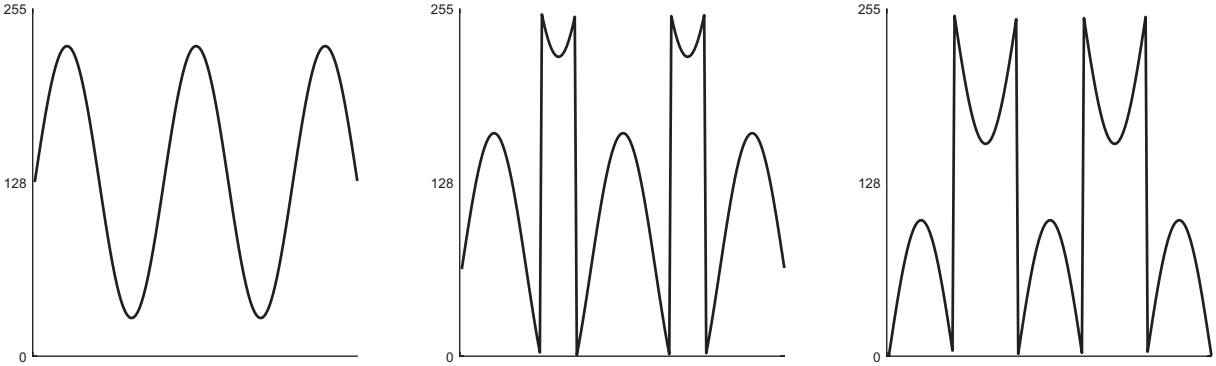
Läs mer här:
https://www.arduino.cc/reference/en/language/variables/data-types/byte/
https://www.arduino.cc/reference/en/language/variables/data-types/int/

När man talar om signaler i allmänna ordalag och ritar kurvor i grafer så antas de ofta ha medelvärdet 0. Det blir oftast enklare så. I vårt fall har vi däremot en signal som endast kan anta värdena 0 till 255, vilket gör det lämpligt att sätta medelvärdet till 127 eller 128, mitt i det tillgängliga talområdet. Om man inte gör så kommer signalen att klippas när den går utanför talområdet åt ena hållet, vilket skapar oönskade övertoner och artefakter i ljudet:



Plottar av 128+100*sin(t), 64+100*sin(t) och 100*sin(t), klippt till [0..255] och med bildtexter som anger de uttrycken.

I vissa fall kan heltalsberäkningar till och med göra en så kallad ”wrap around” så att negativa värden tolkas som stora positiva värden. Talet -1 kommer att misstolkas som 255, talet -2 som 254, och så vidare. Även tal som är för stora misstolkas i sådana fall: 256 kommer att tolkas som 0, 257 som 1, och så vidare. Detta ger en annan sorts övertoner och skapar också artefakter i ljudet:

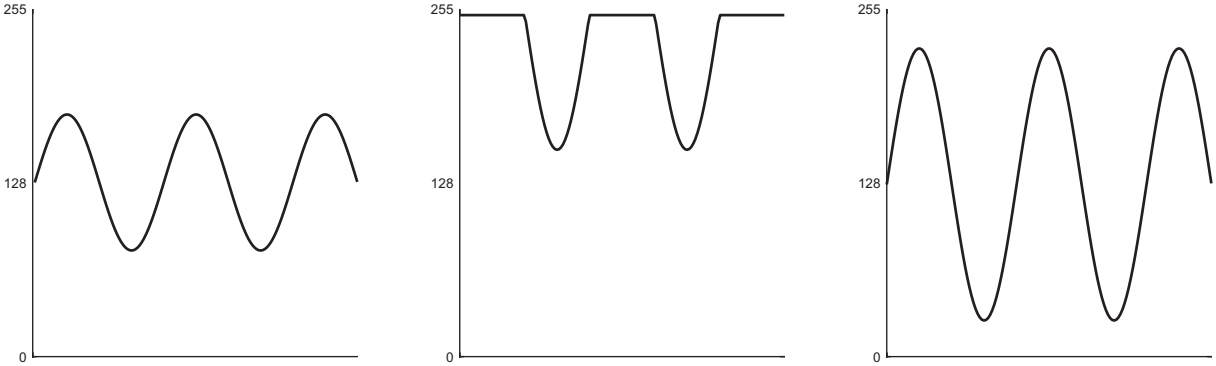


Plot av samma tre kurvor som ovan, men nu med wrap-around ”modulo 256”.

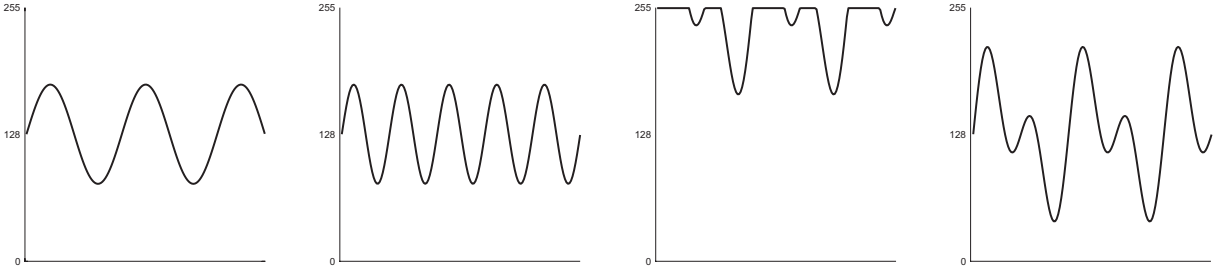
Värdet 128 är alltså det neutrala värdet, det som motsvarar ”ingen signal”. Signalen sägs ha ett offset på 128. I verkligheten motsvaras detta av att medelvärdet av såväl in- och utsignalerna ligger på halva matningsspänningen i stället för på 0 volt. Något oegentligt kallas detta för ”DC-offset” på engelska. (DC är egentligen en förkortning av ”Direct Current”, på svenska ”likström”, men används synonymt med ordet ”likspänning”.).

Läs mer här:
https://en.wikipedia.org/wiki/DC_bias
https://manual.audacityteam.org/man/dc_offset.html

När man räknar på signaler med ett DC offset så måste man hålla tungan rätt i munnen för att hålla slutresultatet inom rätt talområde, och bibehålla samma offset i utsignalen. Enklast är att helt enkelt subtrahera offset från alla ingående variabler, utföra beräkningarna, och till sist lägga till offset på slutresultatet igen:



Plottar av a(t)=128+50*sin(t), 2*a(t) (klippt till 0..255) samt det korrekta uttrycket 2*(a(t)-128)+128.



Plottar av a(t)=128+50*sin(t), b(t)=128+50*sin(2*t), a(t)+b(t) (klippt till 0..255) samt det korrekta uttrycket (a(t)-128)+(b(t)-128)+128.

Naturligtvis är det litet onödigt att beräkna t ex (a(t)-128)+(b(t)-128)+128, eftersom det är matematiskt ekvivalent med a(t)+b(t)-128, men additioner och subtraktioner är väldigt enkla och snabba att göra, och att skriva ut alla operationer på det här sättet gör programkoden läsligare och mer robust mot slarvfel. De flesta moderna kompilatorer kan dessutom hitta konstanta uttryck i koden och förenkla dem i kompileringen.

Notera att man inte kan subtrahera offset från en signal i 8-bitars precision (byte). Negativa tal kan ju inte representeras, så alla värden som ligger under offsetvärdet skulle bli felaktigt beräknade. Använd variabler av typen int för att utföra beräkningarna. Mellanresultat lagrade i byte ger fel i koden som kan vara svåra att detektera och lokalisera.