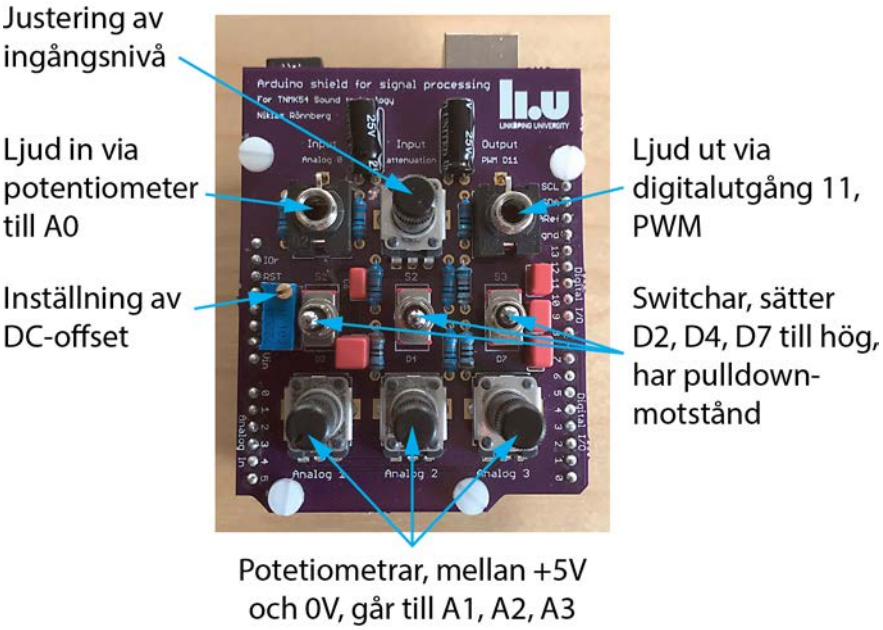
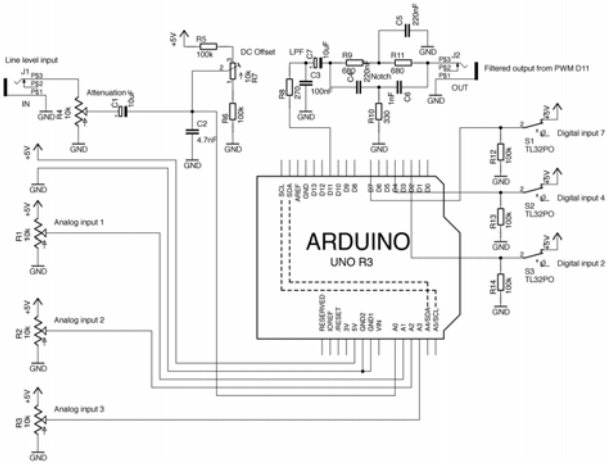
Laboration 2

I denna laboration ska ni testa på hårdvarunära programmering och wavetable-syntes. Till er hjälp har ni ett enkelt shield till Arduino. Ett shield är som ett skal som man trycker på Arduino för att få extra funktioner. Så här är shieldet konfigurerat:



I huvudsak kommer ni att använda er av potentiometern nere till vänster som är kopplad till analogingång 1.

Arduino är enkel och billig, men egentligen inte så väl lämpad för denna uppgift. Klockfrekvensen i ett Arduino är 16MHz, och den har bara 32kB i minne för program och 2kB SRAM, men för denna uppgift är den lagom utmanande och tillräcklig.

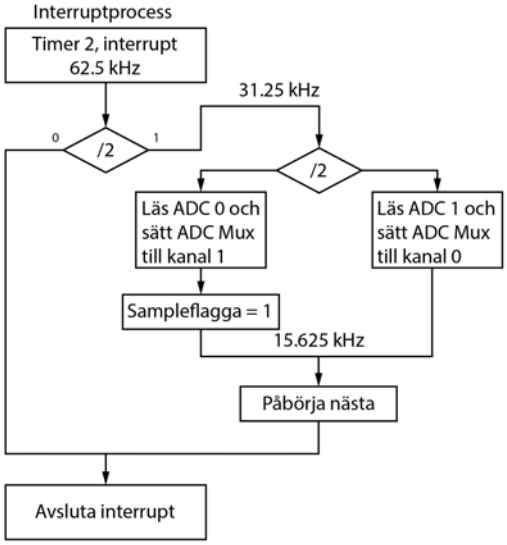


I ovanstående schema ser ni kopplingarna mellan shield (dvs alla komponenter förutom Arduinot) och de olika in- och utgångarna på Arduinot. Ingångssignalen dämpas med potentiometer R4, passerar genom DC-blockerare C1 och lyfts sedan med kopplingen kring R7 (dvs DC-offseten förändras), och går sedan till analogingång 0. Signalen måste lyftas DC-mässigt eftersom Arduinot (som har enkel spänningsmatning) samplar inkommande signal

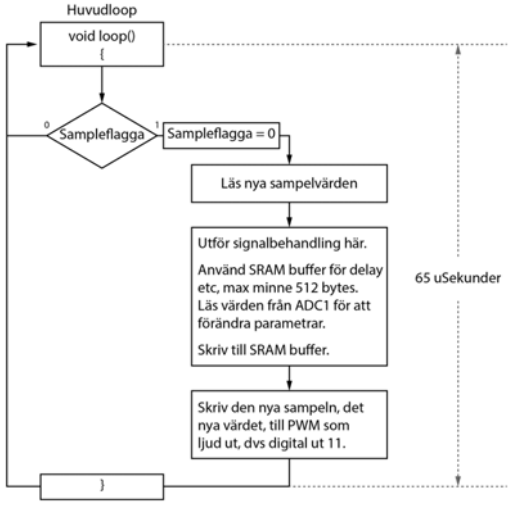
mellan 0 och +5V. Genom R7 lyfts signalens mittpunkt till +2.5V. Lägg märke till att ett lågpassfilter saknas på ingången, och fundera över vad det innebär i sammanhanget.

Utgången, pulsbreddsmodulation (PWM) på digitalpin 11, passerar genom ett lågpassfilter, passerar sedan genom en kondensator C7 som flyttar signalen till att svänga runt 0V igen, och passerar sedan ett notchfilter som filtrerar bort rester från DA-omvandlingen.

Arduinos funktioner för att läsa och skriva information från de analoga ingångarna till de digitala PWM-utgångarna är inte så bra eller i alla fall inte så snabba. Därför körs denna del av koden direkt mot Atmega32-processorn med AVR-kod genom nedanstående interruptprocess. Då denna del av laborationen är lite utanför focus för kursen är koden färdigskriven, men vill ni och kan ni så skriv gärna om och förbättra.

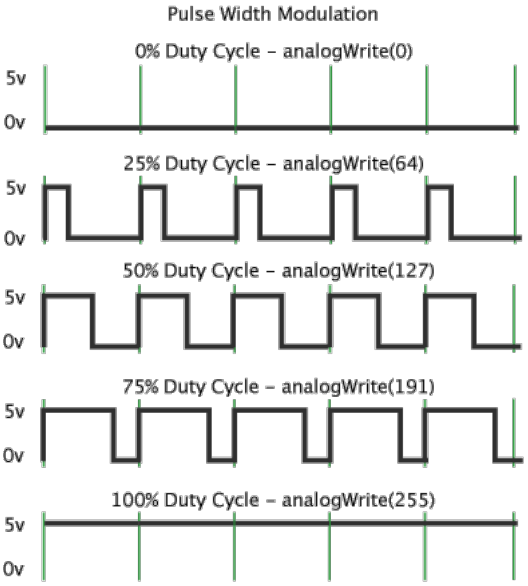


Ert fokus för laborationen är att skriva huvudprogrammet som utför själva uppspelningen av wavetableljuden. I interruptprocessen sätts en flagga (`sampleFlag`) som talar om för huvudloopen att en ny sample har tagits. När denna flagga är 1, sätts den till 0 och signalbehandlingen utförs. En del av funktionaliteten är lite onödig för denna laboration, men grundfunktionaliteten i koden kommer att användas för alla Arduinolaborationer på kursen.

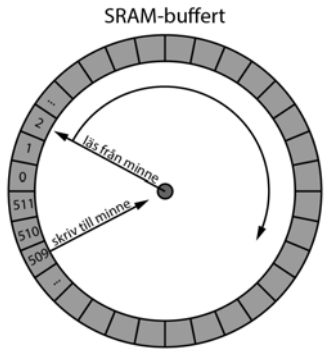


Beroende på vilken typ av syntes eller signalbehandling ni utför så sker olika saker i huvudloopen, men den slutar alltid med att ett värde skrivs till digitalutgång 11 som har PWM. Vanligtvis skulle vi använda analogWrite i Arduino, men då det är lite för långsamt använder vi OCR2A. I huvudloopen har ni redan fått kontroll av flaggan samt loopbackfunktion av sampling till utgången.

I Arduino, och många andra mikrokontroller, används pulsbreddsmodulation (PWM) för att skapa en ”analog” signal från en digital utgång. En digitalutgång är normalt låg (0 volt, dvs `analogWrite(0)`) eller hög (+5 volt, dvs `analogWrite(255)`), men med PWM kan pulsbredden på signalen ändras. Vid 50% (till exempel) är halva cykeln låg (0 volt) och halva hög (+5 volt), vilket i snitt ger 2.5 volt på utgången. Detta är emellertid inte helt sant, utan vi behöver ett lågpassfilter för att omvandla PWM-signalen till en analog signal med rätt spänning.



För att laborationen ska gå att genomföra skapas en SRAM-buffert dit vi kan skriva samplevärden och läsa samplevärden från, i denna laboration kommer vi att fylla bufferten med en vågform (wavetable) som sedan kommer att samplas. Vi använder en buffert på 512 steg (eller 512 bytes). Arduinot har sammanlagt 2kB SRAM, och detta minne ska räcka till alla variabler som används av programmet. SRAM-bufferten läser vi från 0 till 511 och så runt från 0 igen. Vi lagrar ett samplevärde på varje position så det ryms 512 samplingsvärden eller knappt 0.033 sekunder ljud i bufferten.



Inför laborationen

Ni ska med hjälp av Arduinot skapa en tongenerator som läser en vågform och spelar upp den i realtid i relation till frekvens som är ställd med en potentiometer.

Förberedelse – 1. Kolla upp Arduino

Kolla in https://www.arduino.cc/ för att ha bättre koll på kod och struktur i Arduino, och kolla igenom länkarna i slutet av detta dokument.

Förberedelse – 2. 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.

Funktionen `fillSramBufferWithWaveTable` är den funktion som ni ska skriva under laborationen och som kommer att fylla SRAM-bufferten med vågformen.

Förberedelse – 3. Skapa en vågform i Matlab

Skapa valfri intressant vågform i Matlab. Gör något mer intressant än de enkla grundvågformerna. Prova att mixa olika vågformer, experimentera med olika fas, osv. Kom ihåg att vektorn i Matlab ska ha 512 steg så att vågformen sedan stämmer med SRAM-bufferten i denna laboration. I slutet av denna laboration ska ni försöka implementera den vågformen i Arduinomiljön.

Uppgiften

För att testa er wavetablesyntes behöver ni koppla ihop ett Arduino med en av de analoga småsyntarna och högtalaren. Det är lite ont om utrustning tyvärr, så samsas med de övriga är ni snälla.

Den globala variabeln `sampleFlag` är en `boolean`, och deklareras som en `volatile`. Det innebär att variabeln kan komma att ändras bortom kontrollen av koddelen där den används. I Arduino kan detta betyda att variabeln kan ändras i exempelvis en interrupt vilket är fallet här (kolla bilden ovan om interruptprocessen). Ljudet, dvs den globala variabeln `badc0` som är en `volatile byte`, skrivs till den ”analoga” utgången (PWM på digitalpin 11).

Skapa en fyrkantsvåg

Börja med att skapa wavetablen för fyrkantsvågen. En fyrkants våg är antingen hög (1) eller låg (0), men tänk på att vi arbetar med 8-bitarstal. Bestäm i vilken fas fyrkantsvågen ska starta i, fundera över om det har betydelse för hur det låter, och i vilka fall fasen spelar någon roll.

_____

_____

_____

Tänk på att fylla hela SRAM-bufferten (`sramBuffer`) med en period av vågformen. SRAM-bufferten är deklarerad som en global array för 512 bytes. Fyller ni inte hela bufferten kommer inte vågformen att spelas upp korrekt, är perioden längre än bufferten kommer inte hela perioden att användas och är perioden kortare blir "upplösningen" av vågformen sämre och vågformen kommer att börja om sin cykel innan bufferten gör det.

I funktionen `fillSramBufferWithWaveTable` ska ni skapa en variabel `soundValue` som kommer att innehålla värdet på ljudsampeln som ska sparas till `sramBuffer`. Skapa variabeln som en `float` och tilldela den värdet 0 (egentligen skulle vi kunna hoppa över att använda denna variabel, men kanske gör det koden något enklare att förstå och så blir koden mer lik mellan de olika vågformerna om vi använder variabeln). Skapa sedan en `for`-loop som räknar från 0 till och med 511. Syntaxen för en for-loop ser ut så här:

```
for (initialization; condition; increment) {
  //statement(s);
}
```

I for-loopen skapar ni en `if`-sats som kollar om antalet steg är under 255 så ska vågformen vara hög (dvs `soundValue = 192`), och är den 255 eller högre ska vågformen vara låg (dvs `soundValue = 64`). Vi gör på detta sätt för att vågformen ska svänga mellan 64 och 192 runt 128. Varför används 192 och 64 och inte 255 och 0?

_____

_____

_____

```
if (condition1) {
  // do Thing A
} else if (condition2) {
  // do Thing B
} else {
  // do Thing C
}
```

Tilldela sedan `sramBuffer` på position `increment` värdet i `soundValue`.

Läs mer här:
https://www.arduino.cc/reference/en/language/variables/data-types/array/
https://www.arduino.cc/reference/en/language/structure/control-structure/for/
https://www.arduino.cc/reference/en/language/structure/control-structure/if/

Ett snyggare sätt än att hårdkoda antalet varv i `for`-loopen vore att köra loopen från 0 till storleken av `sramBuffer`. Storleken av en array kontrollerar vi genom `sizeof()`.

Läs mer här:
https://www.arduino.cc/reference/en/language/variables/utilities/sizeof/

Spela upp ljudet

Nästa steg är att sampla och spela upp vågformen från bufferten. Detta görs i huvudloopen (`void loop()`).

Tonhöjden på ljudet sätts genom potentiometerns värde. Tonhöjd = frekvens = hur ofta wavetablen samplas vid uppspelning. Detta görs genom att förändra `bufferIndex` genom att summera `bufferIndex` med värdet i `badc1`. Värdet på potentiometern stoppas i variabeln `badc1`, kolla i `Timer2`-interrupten. Värdet på `badc1` är ett 8-bitarsvärde.

Varje varv som huvudloopen körs ska `bufferIndex` öka med 1, även om `badc1` är inställd på 0. Annars ska `bufferIndex` öka med `badc1`. Sedan måste räkningen av `bufferIndex` begränsas så att värdet inte överstiger 511. Det är ju 512 steg som räknas från 0 till 511. Men, eftersom vågformens period är 512 steg (i `sramBuffer`), så ska räkningen gå runt när

den når 511. Steg 50 + 50 ska bli 100, och steg 500 + 50 ska bli 39. Detta åstadkoms med modulo (`%`).

Läs mer här:
http://arduino.cc/en/Reference/Modulo

Läs samplingsvärdet ur SRAM-bufferten (`sramBuffer`) på position `bufferIndex`, och lägg värdet i en ny variabel `sramBufferSampleValue`. Avslutningsvis ska ni skicka det nya ljudet, dvs `sramBufferSampleValue`, till utgången (`OCR2A`). Eftersom ni skapat vågformen att svänga mellan 0 och 255, behöver ni inte ”lyfta” ljudsampeln (DC-offseten).

Föra över koden till Arduino

För att föra över koden till Arduino behöver ni dubbelkolla att ni använder rätt kommunikationsport under Tools -> Port i Arduino IDEt. Nästa sak som behövs kontrolleras är att ni kompilerar för rätt plattform under Tools -> Board. Ni ska använda Arduino/Genuino Uno.

Lyssna på ljudet

När ni har fört över koden till Arduinot startar Arduinot om och ni ska höra ljudet. Hur förändras ljudet när ni justerar frekvensratten? Är det något annat än själva tonhöjden (pitchen) som förändras? Om något annat förändras, vad förändras och vad kan det i så fall bero på?

_____

_____

_____

Skapa en sågtandsvåg

Skapa sedan en sågtandsvåg. Hur ska den se ut för att vara i samma fas som den fyrkantsvåg ni skapade?

_____

_____

_____

Sätt `soundValue` till 0 (eller 255 beroende på vilken fas ni vill använda). Sågtandsvågen ska gå från hög (1) till låg (0) eller låg till hög, i 8-bitarstal, över 512 steg. Gör detta med en `for`-loop där ni minskar värdet på `soundValue` (från 255) med 255/512. Här kan ni springa på vissa problem med `float` och division av heltal, men tänk efter så löser ni detta lätt. Eftersom `soundValue` är en float sparas värdet med decimaler men ett 8-bitarstal är ett heltal (egentligen mellan 0 och 255), ni kan därför använda er av avrundning av talen innan ni stoppar in dem i `sramBuffer`.

Vi kan använda olika typer av avrundning: `round`, `ceil`, och `floor` tack vare `math`, som är ett mattebibliotek och är inkluderat i projektet genom:

```
#include <math.h>
```

Läs mer här:
https://www.arduino.cc/en/math/h
http://www.nongnu.org/avr-libc/user-manual/group__avr__math.html

http://www.nongnu.org/avr-libc/user-manual/group__avr__math.html#ga0f0bf9ac2651b80846a9d9d89bd4cb85

Men, behövs egentligen värdena avrundas eftersom SRAM-bufferten är deklarerad som en array med bytes? Och vad skulle vara fördelen med att avrunda värdena ”manuellt”?

_____

_____

_____

Skapa en triangelvåg

En triangelvåg skapar ni på liknande sätt som fyrkantsvågen och sågtanden. Fundera i vilken fas ni vill påbörja ljudet i.

Sätt `soundValue` till 0 (eller 255 beroende på vilken fas ni vill använda). Sedan ser ni till att räkna `soundValue` från 0 till 255 till 0, eller från 255 till 0 till 255, på 512 steg. Detta kan ni göra genom en med, eller flera `for`-loopar utan, `if`-satser. Ni väljer arbetssätt.

Ett sätt är att sätta `soundValue` till 255, och sedan minska `soundValue` med 255/256 (och avrunda värdet uppåt när `soundValue` läggs till i `sramBuffer`, sedan ökas `soundValue` på samma sätt men med avrundning uppåt. Detta kommer dock att ge en nedåtgående vågform from 255 till 0, och sedan från 0 till 255. Det innebär att triangelvågen kommer att vara lite plattare när den vänder.

Hur kan detta påverka övertonsserien i vågformen? Hur skulle ni kunna skapa en mer ”ren” triangelvåg? Tänk att den ska gå från 255 till 1 och sedan från 0 till 254 (avrundningar åt rätt håll hjälper till…). Gör det, och lyssna och se om ni hör någon skillnad.

_____

_____

_____

Skapa en sinusvåg

Att skapa sinusvågformen är lite klurigare än de enkla vågformerna vi hittills har gjort. Sätt `soundValue` till 0, och skapa därefter en ny variabel för delta av `soundValue`. Skapa den som en `float` och tilldela den värdet av (2 * π)/storleken av `sramBuffer`. Pi skriver vi med `M_PI` och den funktionen kommer från `math`.

Inne i `for`-loopen ska vi skapa en ny variabel (`sinusSample`). Det innebär att den variabeln kommer att skapas och skrivas över varje varv i loopen, och det funkar utmärkt i detta fall. Här ska 127 multipliceras med `sin` av `soundValue`. `Sin` räknar ut sinusen av en vinkel (i radian), och resultatet är ett värde mellan -1 och 1. Vi multiplicerar med 127 eftersom värdet ska vara i en spann av 255, och så adderar vi 127 för att förändra DC-offseten. Därefter ska `soundValue` uppdateras till att bli `soundValue` plus delta av `soundValue`, och slutligen kan `sinusSample` läggas till i `sramBuffer`.

Läs mer här:
https://www.arduino.cc/reference/en/language/functions/trigonometry/sin/

Vad händer med sinus-vågformen när ni spelar upp den i olika frekvens? Vad kan dessa skillnader bero på?

Skapa er vågform som ni tagit fram i Matlab

Matlab är väldigt trevligt att arbeta med när det gäller matematiska formler och uträkningar, och inte minst ljud. Lika enkla verktyg finns dock inte i det ordinarie utbudet av funktioner i Arduino, men försök att ändå skapa er vågform i Arduino. Kom ihåg att värdena i SRAM-bufferten ska vara mellan 0 och 255.

Läs mer här:
https://en.wikipedia.org/wiki/DC_bias

Vilken egen vågform valde ni? Hur skapade ni den i Matlab? Hur implementerade ni den i Arduino?

Låter vågformen annorlunda när ni spelar upp den i Matlab jämfört mot när ni spelar upp den via Arduino? Vad är det som skiljer? Vad beror skillnaden på?

Mix av vågformer

Det är enkelt att mixa vågformer. I princip är det bara att ta sample1 i den ena vågformen + sample1 i den andra vågformen. Utmaningen när vi gör detta med ett begränsat tillgängligt minne är att värdena för de två vågformerna bör skapas, summeras och sedan läggas till i SRAM-buffer på en gång. Risken är stor att minnet i Arduinot tar slut om ni skapar två wavetable på 512 byte vardera och sedan mixar dessa två till en tredje. Alltså måste ni tänka efter hur ni skapar vågformerna så att vågformernas samplevärde kan skapas samtidigt men att ni bara lagrar det nya mixade värdet i SRAM-buffer. Kom ihåg att dela med 2, så att ni inte överstyr ljudet. Ljudet måste ligga mellan -127 och 127 när ni summerar och dividerar, sedan ska DC-offseten ändras tillbaka.

Vilka vågformer testade ni att mixa? Vad var svårast att lösa? Hur förändrades ljudet jämfört med de ”rena” vågformerna?

Att spela upp mer än en ton

Det går att spela upp mer än en ton åt gången. Detta kan ni lösa genom att läsa två samplingsvärden samtidigt, summera dessa (och dela med 2 för att undvika att signalen överstyr (dvs att ni har värden som är högre än 255) och kom ihåg att arbeta med rätt DC-offset) för att sedan spela upp summan. Om ni läser den ena samplingen hälften så ofta som den andra, kommer den första att klinga en oktav nedanför. Om ni i stället spelar den andra tonen 50% snabbare än den första så kommer den att klinga en perfekt femma högre (enklast här är att multiplicera `badc1` med 3 även om det ger en kvint en oktav högre upp). Tänk på att det ni ska ändra är värdet som ni samplar från potentiometern. Använd det alternativa indexet, `bufferIndex2` för den andra tonen.

Vilka intervall provade ni? Hur lät det?

_____

_____

_____

Läs mer här:
https://en.wikipedia.org/wiki/Perfect_fifth
https://en.wikipedia.org/wiki/Interval_ratio
https://en.wikipedia.org/wiki/Interval_(music)
https://pages.mtu.edu/~suits/notefreqs.html
https://en.wikipedia.org/wiki/Piano_key_frequencies

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/

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

Wavetable synthesis
https://en.wikipedia.org/wiki/Wavetable_synthesis

What is Wavetable Synthesis? – Wavetable Synthesis Explained
https://musicproductionnerds.com/what-is-wavetable-synthesis