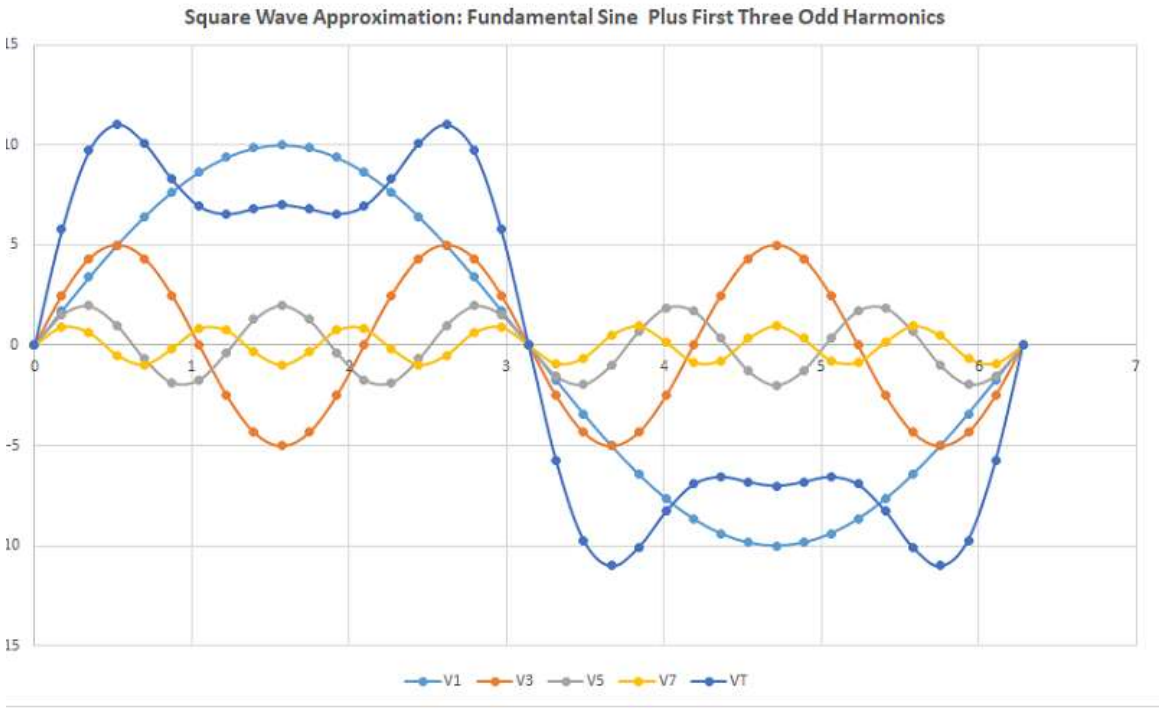
ROYAL ST. GEORGE'S COLLEGE
ADVANCED COMPUTER ENGINEERING SCHOOL

Fundamental Waveform Generation and Processing Techniques



The purpose of this workbook is to investigate techniques for the creation and manipulation of elementary DC and AC signal waveforms. An early exposure to such concepts and properties will provide a strong foundation for further undergraduate study.

ACE: _____

COURSE: ICS4U-E

INSTRUCTOR: C. D'ARCY

CREATED: JANUARY 2026

INSPIRATION: G. DAVIDGE (ACES '24)
B. SNELGROVE (SGC '71)

CÉDRIC VILLANI ON JOSEPH FOURIER'S 'MATHEMATICAL POEM'
HTTPS://WWW.YOUTUBE.COM/WATCH?V=UE-RIHFSDLK

COVER IMAGE
HTTP://DARCY.RSGC.ON.CA/ACES/TEI4M/WAVEFORMGENERATION/HARMONICS.XLSX

Inspiration

Graham Davidge (ACES '24)

Over his time in our ACES program Graham achieved a number of remarkable results. To my mind his ICS4U Short ISP, and 8-Channel Spectrum Analyzer was perhaps the most notable,

https://www.youtube.com/watch?v=Vl6h5RWQ3bs

At the time I knew very little about the concepts Graham was implementing but vowed at some point to explore his achievement in greater detail so I could pass them on to ACES that follow.

This workbook is that promise to myself to return to the project in the hope of unpacking its principles for current ACES.

Bob Snelgrove (SGC '71)

A hitherto unfamiliar Georgian alumnus approached me at a dinner in 2023 with a view to sharing his engineering journey with me. As a now-retired professional Bob mused about how his career might have been impacted had RSGC offered an ACES-like program when he attended the College.

Bob contacted me again in the fall of 2025 wondering if he might offer support for those ACES wanting to know more about the fundamentals of audio signals and mixing. With Graham's project in the back of my mind I accepted Bob's offer to attend the DES with a view to outlining a possible way forward.

My takeaway from our three meetings was that I needed to introduce and build some background knowledge before ACES could tap into what Bob was offering. This workbook is my attempt to fill that gap.

80 Bloor St. West

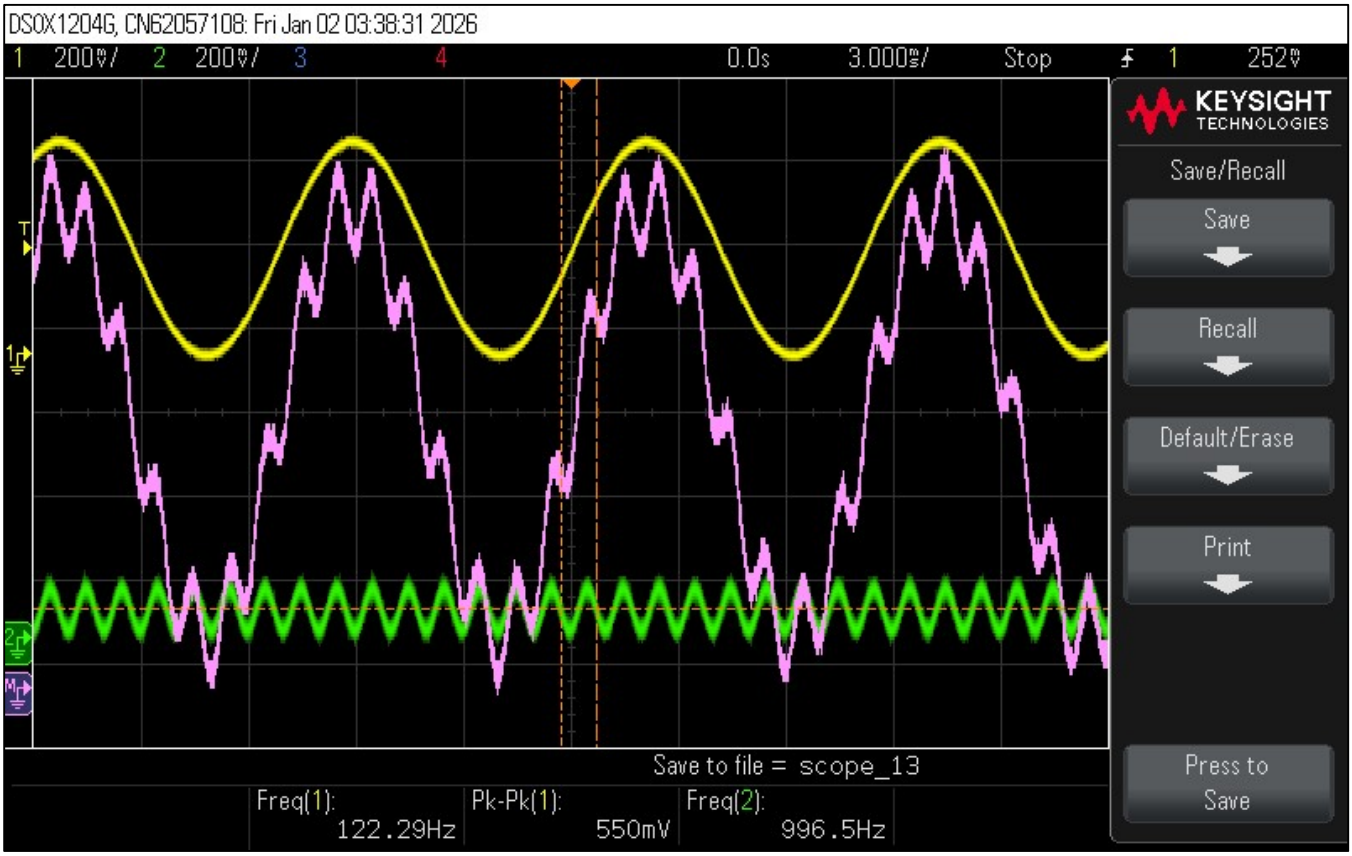
Concepts never come easy to me. Whereas others (even my talented students) appear to master new things relatively quickly, despite how motivated I am to learn, I am always surprised how much time it takes me to process everything. The act of writing helps to solidify my research gains.

In the particular case of this current compilation of research, I found unexpected inspiration from the proposed 78-storey condominium development on Bloor Street. Rising from the foundation deep below the surface all the way to the penthouse I see the structure as a similar process or journey.

I found a strong parallel for my goal of understanding the concepts developed in this workbook. I likened what Graham's and Bob's contribution to this odyssey to the 50th floor of the tower. As marvelous as their guidance was I couldn't access them as my starting point for current ACES. I needed to build the foundation first. Hopefully, eventually, I'll get to the 50th floor and, with luck and time, perhaps beyond.

TABLE OF CONTENTS

A 122 Hz waveform (yellow) added to a 1 kHz waveform (green) yields the pink waveform.



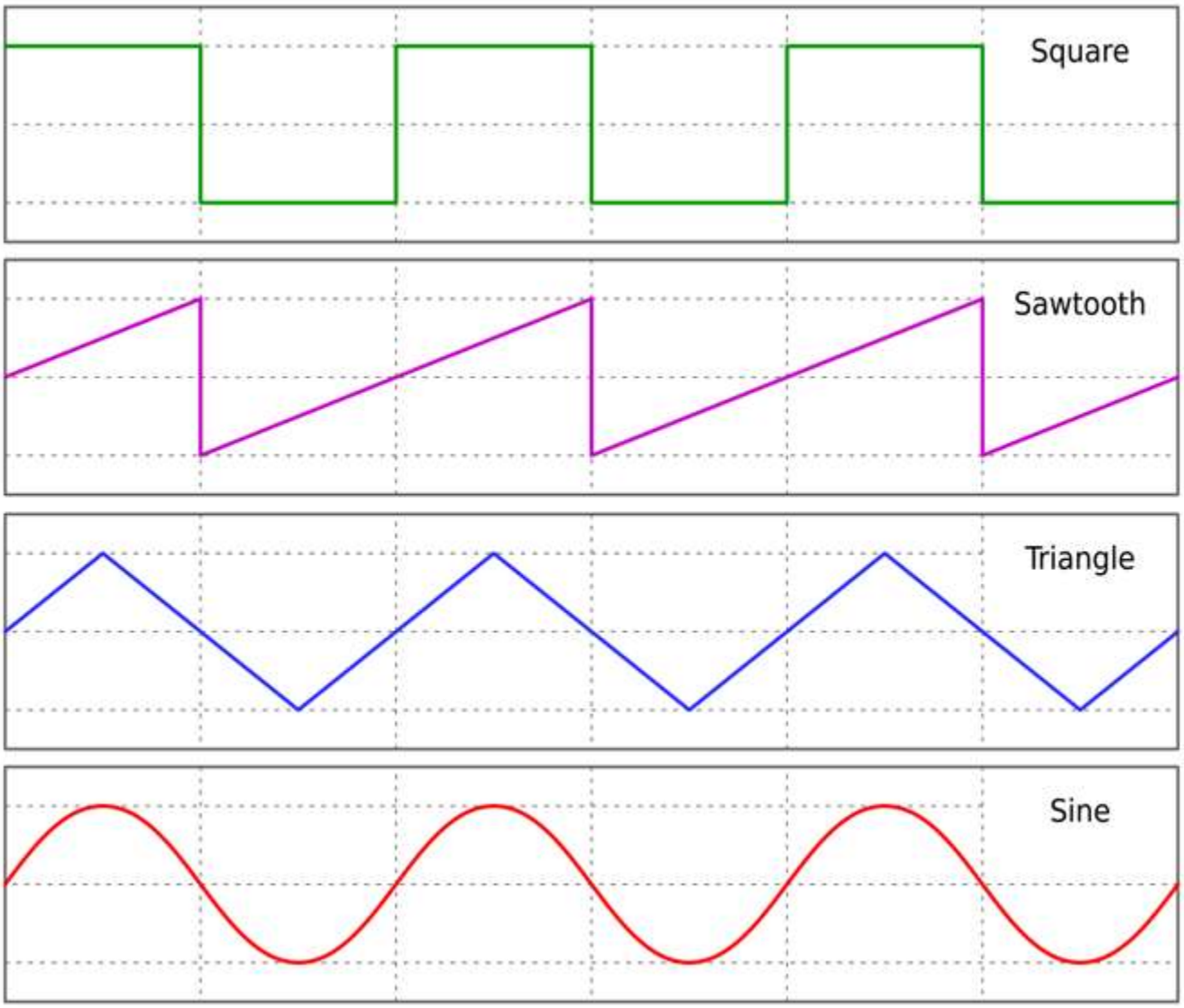
AC Waveforms

To this point in the ACES program direct current (DC) has been the norm. Pulsed DC waveforms, either from the Arduino's `analogWrite()` function to blink an LED or position a Servo horn, or the 555 Timer to drive buzzer, have been the exception.

With a potential electrical engineering undergraduate program in the mix for some ACES it would be advantageous to broaden our scope to include a deeper dive into some fundamental properties of AC waveforms.

Types

See: https://en.wikipedia.org/wiki/Waveform



The *independent* axis can be considered as *time*. No scale is provided as the period can be stretched or compressed as desired.

The *dependent* axis can be considered as *voltage*. No scale is provided as the waveform can be stretched vertically and be biased up or down into either the DC or AC ranges.

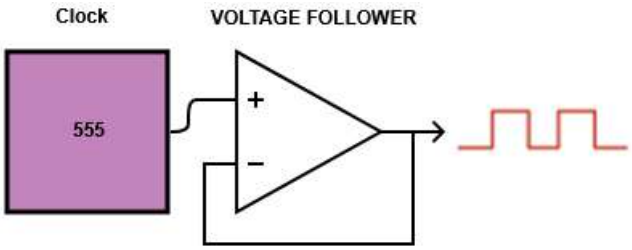
Simplest (?) Waveform Generation

See: Falstad: R-2R Resistor Ladder https://tinyurl.com/22ym9wp8

For actual applications the operational amplifier acts as a buffer providing a low impedance alternative to the high impedance output of the generators. It is optional for testing.

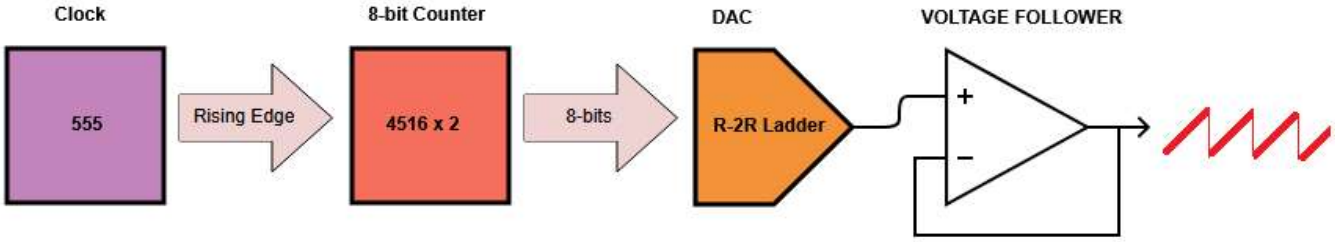
Square

Could this be the simplest *square* waveform generator?



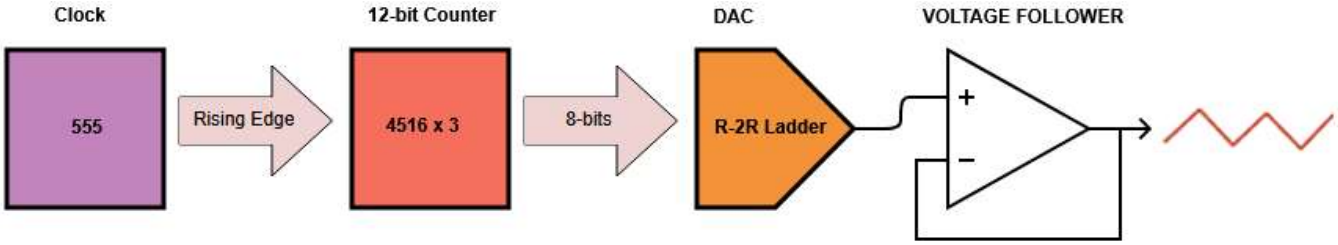
Sawtooth (Ramp)

Could this be the simplest *sawtooth (ramp)* waveform generator from a square wave?



Triangle

Could this be the simplest *triangle* waveform generator from a square wave? Adding a 3rd 4516 (4th?) enables a 12-bit counter, $b_{11}...b_0$. However, using b_8 to influence the up/$\overline{\text{down}}$ pins is the trick. Can you think how you might do it?



Exercise. Sine

I'll leave this one for you to research and implement the simplest *sine* waveform generator from a square wave.

Graphs

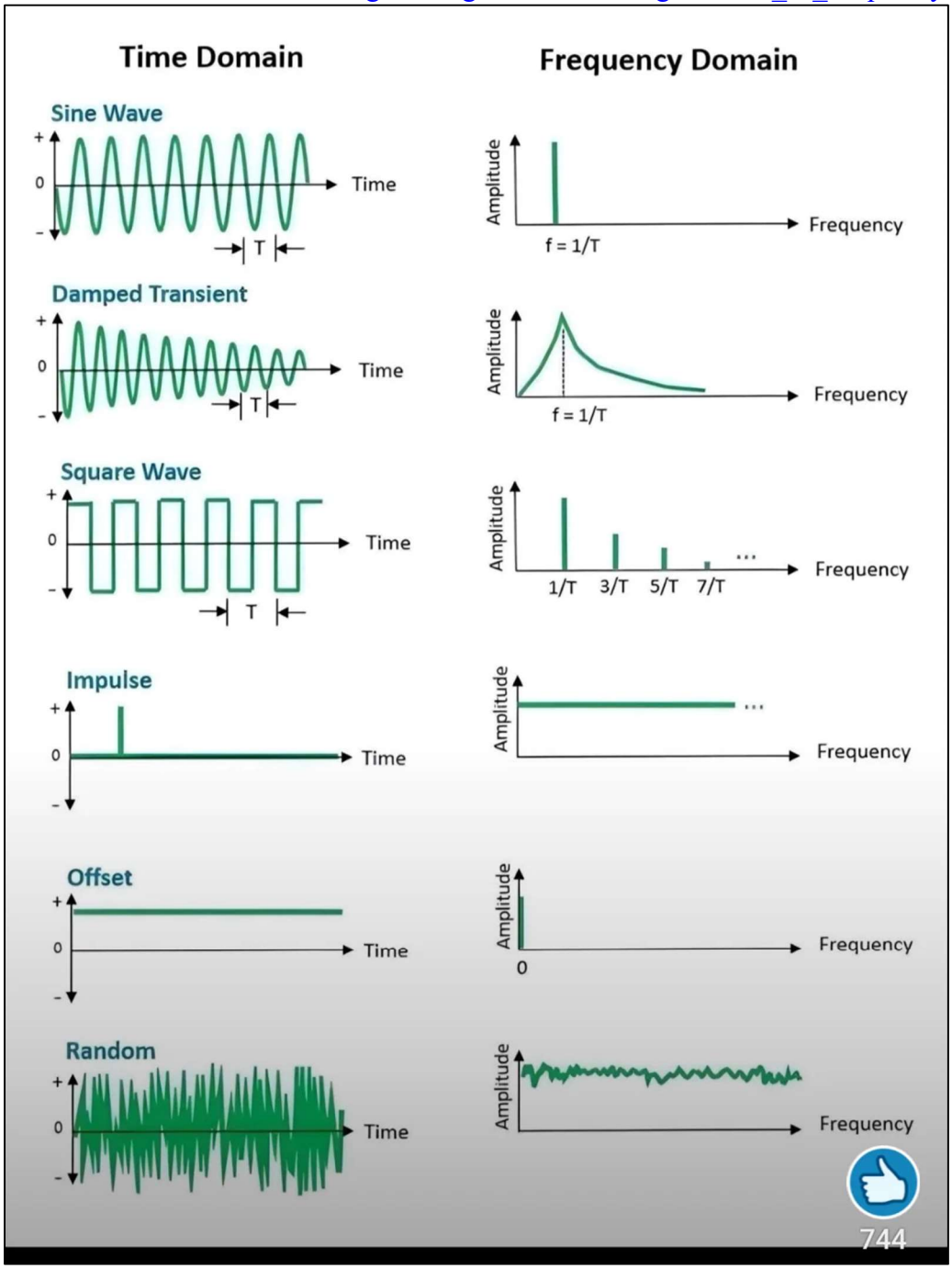
Voltage vs Time

The left column below presents the standard representation of a waveform with *voltage* expressed and a function of *time*.

Amplitude vs Frequency

Some analytic techniques find it preferable to transfer the variables to an *amplitude* vs *frequency* representation as shown in the right column.

See: https://www.reddit.com/r/ElectricalEngineering/comments/1oag0s2/time_vs_frequency/



Monitoring Waveforms

Arduino Serial Plotter

Despite its severe limitations, the Arduino IDE's Serial Plotter utility can provide some degree of insight as shown below.

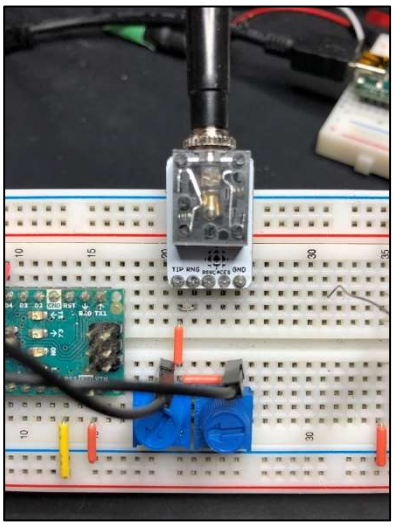

```
// PROJECT   : 555Monitor
// PURPOSE   : Captures the oscillations from the 555 and converts them to a frequency
// COURSE    : TEJ3M
// AUTHOR    : C. D'Arcy
// DATE      : 2025 11 04
// MCU       : 328p (Nano)
// STATUS    : Working
// REFERENCE: https://www.allaboutcircuits.com/tools/555-timer-astable-circuit/

#define DIGITALCAP 2      //Monitor 555_3 on Nano_2
#define ANALOGCAP A0      //Monitor 555_2 on Nano_A0
#define CAPTURE 2         //(Optional) Monitor 555_3 as an External Interrupt on Nano_2
#define VCC 5.0           //Voltage supply maximum

void setup() {
  Serial.begin(9600);
  while (!Serial) {}
}

void loop() {
  Serial.print(VCC * analogRead(ANALOGCAP) / 1024);
  Serial.print(',');
  Serial.print(VCC * digitalRead(DIGITALCAP));  //Scale it to stabilize the Serial plotting
  Serial.println();
}
```

Audio Jack, Headphones and Simple Mixer



Audio confirmation of your generated waveforms is a simple but effective metric. To this end a cable from your headphones with a 3.5 mm plug can be inserted into an ACES audio jack to provide a general degree of support for your prototyping.

Audio plugs come in different configurations. The tip (T) carries a signal and the Sleeve (S) provides a ground reference.
The TS mode is mono. The TRS offers stereo. The TRRS provides a channel for a microphone.

A crude mixer can be created from dual frequency inputs as shown to the right. Potentiometers serve as volume control. In this example the left and right signals have separate volume control before being combined into one and presented as single signal on both the left (Tip) and Right (Ring) channels.

Electret Microphone

AI: Electret and piezo microphones are distinct technologies: electret microphones (ECM) use a polarized membrane to detect air pressure changes for high-fidelity audio, while piezo transducers (PZO) detect physical vibrations (contact) and often require high-impedance preamps to avoid sounding "tinny". They are not generally interchangeable without modification. Both microphone designs require significant amplification to be usable.

https://www.digikey.ca/en/products/detail/CMA-4544PF-W/102-1721-ND/1869981

A convenient electret microphone with built-in Automatic Gain Control (AGC) is the Adafruit 1713 device.

From the datasheet: *The chip at the heart of this amp is the MAX9814 (https://adafru.it/d9r), and has a few options you can configure with the breakout. The default 'max gain' is 60dB, but can be set to 40dB or 50dB by jumpering the Gain pin to VCC or ground. You can also change the Attack/Release ratio, from the default 1:4000 to 1:2000 or 1:500. The output from the amp is about 2Vpp max on a 1.25V DC bias, so it can be easily used with any Analog/Digital converter that is up to 3.3V input. If you want to pipe it into a Line Input, just use a 1uF blocking capacitor in series.*

FNIRSI 2C53T Hand-Held 3-in-1 (Oscilloscope, Multimeter, Signal Generator)

Nothing beats an oscilloscope for the graphic monitoring waveforms and displaying their parameters. Although the price range of these devices can reach the $1000s it was decided in the Fall of 2025 to purchase a class set of the FNIRSI 2C53T model. The versatility of the unit for an Amazon price of $150 seemed like good entry-level value.

From the Product Page: *FNIRSI-2C53T is a versatile and highly practical three-in-one dual-channel digital oscilloscope launched by FNIRSI, designed for professionals in the maintenance and research industries. This device combines the functionalities of an oscilloscope, a multimeter, and a signal generator.*

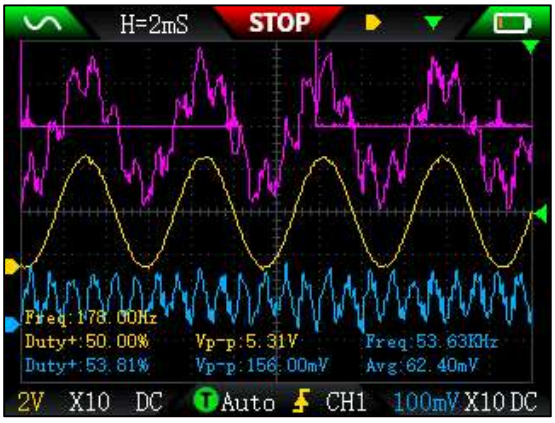
*The **oscilloscope** uses FPGA + ARM + ADC hardware architecture, featuring a 250MS/s sampling rate, a 50MHz analog bandwidth, and an integrated high-voltage protection module that supports peak voltage measurements up to ±400V. It also supports waveform screenshot saving and viewing for secondary analysis.*

*The **multimeter** function offers 4.5 digits with 19999 counts true RMS, supporting AC/DC voltage and current measurements, as well as capacitance, resistance, diode, and continuity measurements, making it an ideal multifunctional instrument for professionals, factories, schools, enthusiasts, or home use.*

*The built-in **DDS function signal generator** can output 13 types of function signals, with a maximum output frequency of 50KHz and a step size of 1Hz. The output frequency, amplitude, and duty cycle are adjustable. It features a 2.8-inch 320*240 resolution high-definition LCD screen and a built-in 3000mAh rechargeable lithium battery with a standby time of up to 6 hours. Its compact size provides users with more powerful practical functions and excellent portability.*

The 2C53T's **Fast Fourier Transform (FFT)** feature is interesting, if not limited. However, it opens the door to a much deeper conceptual and mathematical investigation beyond the scope of this workbook. Fourier analysis is a process by which *voltage-time* representations of a waveform are transformed to an *amplitude–frequency* interpretation. Enabling the FFT feature for a channel results in a small purple amplitude–frequency plot appearing at the top of the display window. Documentation of the plot in the user manual is lean on detail but as far as I have determined the first bar represents the DC component of the waveform and the other bars the various frequencies.

This scope has the ability to perform a limited number of **mathematical transformations** on the two input channels. One such transformation is `CH1+CH2`.



In the capture to the right, `CH1` is connected to a 178 Hz sine wave from [Technoblogy's ATtiny Waveform Generator] (yellow) and `CH2` is monitoring a 1 kHz sine wave (blue) from the `2C53T` signal generator (despite what the parameters are suggesting below). The sum of the two waveforms appear in purple.

Modulation

On-Off Keying (OOK)

AI: Arduino OOK (On-Off Keying) is a simple digital modulation technique used to transmit data by turning a carrier signal (often 433 MHz for radio) on for a binary '1' and off for a '0'. It is commonly used with Arduino boards to control remote devices like lights, sensors, and garage doors.

Pulse Width Modulation (PWM)

AI: Arduino Pulse Width Modulation (PWM) is a technique for simulating analog voltage levels using digital outputs by rapidly switching a signal between high (5 V or 3.3 V) and low (0V) states. By adjusting the duty cycle (on-time percentage) using `analogWrite()`, it controls power for motor speed, LED brightness, and servo angles.

Bit Angle Modulation (BAM)

AI: Bit Angle Modulation (BAM), or Binary Code Modulation, is an efficient Arduino technique for LED brightness control by switching them rapidly based on binary bit significance. It requires less processing power than PWM and is ideal for multiplexing, driving LEDs by stretching bit pulses in (1,2,4,8...) ratios. Bit Angle Modulation is also referred to a Binary Code Modulation (BCM).

AVR Timers

See: http://darcy.rsgc.on.ca/ACES/Datasheets/AVR130.pdf

Overview

In principle, a timer is a simple counter. Its advantage is that the input clock and operation of the timer is independent of the program execution. The deterministic clock makes it possible to measure time by counting the elapsed cycles and take the input frequency of the timer into account.

About Timers

Generally, the megaAVR microcontrollers have two 8-bit and one 16-bit Timer/Counters (*timer*). A timer with 16-bit resolution is certainly more flexible to use than one with 8-bit resolution. However, the saying "bigger is better" does not necessarily apply to the microcontroller world. For many applications, it is sufficient to have 8-bit resolution. Using a higher resolution means a larger program overhead, which costs processing time and should be avoided in speed optimized code. It also means higher device cost. Because of the flexibility of the AVR timers, they can be used for different purposes. The number of timers determines the amount of independent configurations.

Registers

Configuring Timer/Counters requires manipulating individual bits within dedicated 8-bit memory locations (registers).
See: http://darcy.rsgc.on.ca/ACES/TEI4M/Assembly/ATmega328RegisterSummary.pdf

Timer Events

The timer of the AVR can be specified to monitor many events. Status flags in the `TIMSK` register show if an event has occurred. The ATmega328P can be configured to monitor events for the 8-bit timers TC0 and TC2. and one 16-bit timer (TC1).

Interrupts

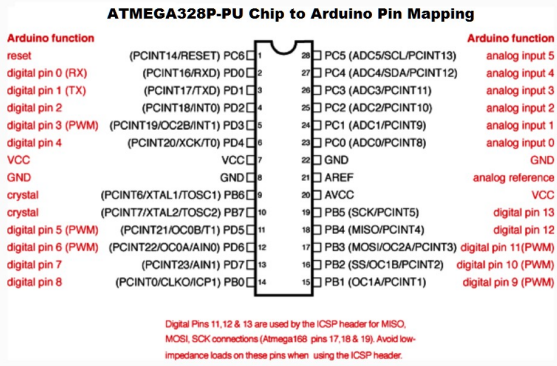
AI: ATmega328P interrupts are hardware/software signals that temporarily pause the main program to execute high-priority tasks immediately. They enable efficient, non-blocking responses to external events (pins 2, 3), timers, or serial communication. When triggered, the CPU saves its current state, runs an Interrupt Service Routine (ISR), then resumes.

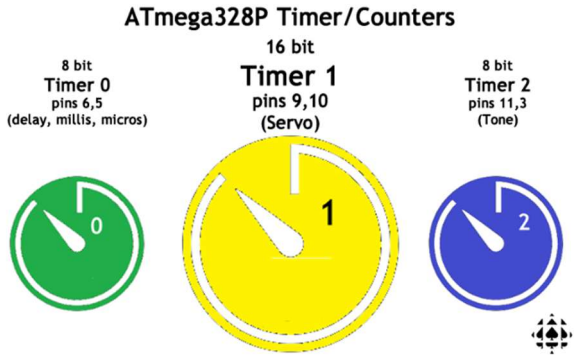
PWM Square Wave Generation

AnalogWrite() PWM

See: Secrets of Arduino PWM: https://www.righto.com/2009/07/secrets-of-arduino-pwm.html
From a software perspective the simplest method of generating a PWM square wave is through the use of the Arduino library's `analogWrite()` function. This functions takes control of one of the timers (3 on the ATmega328p) and can present a square wave with a **fixed** frequency (980 Hz on Timer 0 and 490 Hz on Timers 1 and 2). The **variable** duty cycle defines the simulated analog voltage over 256 quantization levels (8-bit resolution) resulting in a voltage range of 0 V to 5 V.

Note. Other built-in Arduino functions also depend on timers to perform their duties as shown below right. Care must be taken not to overlap the use of the timers.





Register-Level PWM

Overcoming the limitation of the `analogWrite()`'s fixed frequency and generating more versatile waveforms is achieved through direct manipulation of the Timer/Counter dedicated registers. These will be investigated in the pages that follow. By way of example, the ATmega328p's Timer/Counter 1 offers 16 different modes that can be tailored to either produce directly or support the synthesis of virtually any manner of signal.

Table 16-4. Waveform Generation Mode Bit Description[1]

Mode	WGM13	WGM12 (CTC1)	WGM11 (PWM11)	WGM10 (PWM10)	Timer/Counter Mode of Operation	TOP	Update of OCR1x at	TOV1 Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCR1A	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICR1	BOTTOM	BOTTOM
9	1	0	0	1	PWM, Phase and Frequency Correct	OCR1A	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICR1	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCR1A	TOP	BOTTOM
12	1	1	0	0	CTC	ICR1	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICR1	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCR1A	BOTTOM	TOP

Exercise 1. Timer 0 Normal Mode 0 with Overflow Interrupt

The simplest register-level manipulation of a timer that results in a square wave is this one: Timer 0, Normal Mode 0 with overflow interrupt handling.

```
1  // PROJECT  :Timer0NormalOverflow
2  // PURPOSE  :Register Level implementation of Timer0 Normal Mode 0
3  // COURSE   :ICS4U-E
4  // AUTHOR   :C. D'Arcy
5  // DATE     :2025 12 06
6  // MCU      :328P
7  // STATUS   :Working
8  // REFERENCE:
9  // NOTES:   Can't use setup & loop as they have already initialized the
10 //          millis() function and, hence, are using ISR(TIMER0_OVF_vect)
11
12 #include "prescalers.h"  //local library of Timer Prescalers
13
14 int main() {
15   DDRB |= (1 << PB5);        //Configure Pin 13 for output
16   configureTimer0Mode0();    //Timer 0: Normal Mode 0 with
17   sei();                     //enable global Interrupt System
18   while (1){}                //hold
19 }
20
21 ISR(TIMER0_OVF_vect) {
22   PINB |= (1 << PB5);
23 }
24
25 void configureTimer0Mode0() {
26   TCCR0A = 0;               //Normal Mode 0;
27   TCCR0B = T0psNone;        //Fovf = Fclk/PS
28   TIMSK0 = (1 << TOIE0);    //Timer Overflow Interrupt Enable
29 }
```

Timer0 Registers

15.9.1 TCCR0A – Timer/Counter Control Register A

Bit	7	6	5	4	3	2	1	0	
0x24 (0x44)	COM0A1	COM0A0	COM0B1	COM0B0	–	–	WGM01	WGM00	TCCR0A
Read/Write	R/W	R/W	R/W	R/W	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

15.9.2 TCCR0B – Timer/Counter Control Register B

Bit	7	6	5	4	3	2	1	0	
0x25 (0x45)	FOC0A	FOC0B	–	–	WGM02	CS02	CS01	CS00	TCCR0B
Read/Write	W	W	R	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

15.9.6 TIMSK0 – Timer/Counter Interrupt Mask Register

Bit	7	6	5	4	3	2	1	0	
(0x6E)	–	–	–	–	–	OCIE0B	OCIE0A	TOIE0	TIMSK0
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

1. Assume an ATmega328p under the following conditions,
   - f_{CLK} = 16 MHz
   - 8-bit Timer 0 in Normal Mode WGM02:0 = 0 (Mode 0)
   - 16-bit Timer 0 in Normal Mode WGM03:0 = 0 (Mode 0)
   - ISR for Timer/Counter0 Overflow **toggles** Pin 13 enabled
   - ISR for Timer/Counter1 Overflow **toggles** Pin 13 enabled
   - OCnA and OCnB disconnected

ATmega328P Interrupt Vector Table

```
 1  Reset
 2  External Interrupt Request 0  (pin D2)          (INT0_vect)
 3  External Interrupt Request 1  (pin D3)          (INT1_vect)
 4  Pin Change Interrupt Request 0 (pins D8 to D13) (PCINT0_vect)
 5  Pin Change Interrupt Request 1 (pins A0 to A5)  (PCINT1_vect)
 6  Pin Change Interrupt Request 2 (pins D0 to D7)  (PCINT2_vect)
 7  Watchdog Time-out Interrupt                     (WDT_vect)
 8  Timer/Counter2 Compare Match A                  (TIMER2_COMPA_vect)
 9  Timer/Counter2 Compare Match B                  (TIMER2_COMPB_vect)
10  Timer/Counter2 Overflow                         (TIMER2_OVF_vect)
11  Timer/Counter1 Capture Event                    (TIMER1_CAPT_vect)
12  Timer/Counter1 Compare Match A                  (TIMER1_COMPA_vect)
13  Timer/Counter1 Compare Match B                  (TIMER1_COMPB_vect)
14  Timer/Counter1 Overflow                         (TIMER1_OVF_vect)
15  Timer/Counter0 Compare Match A                  (TIMER0_COMPA_vect)
16  Timer/Counter0 Compare Match B                  (TIMER0_COMPB_vect)
17  Timer/Counter0 Overflow                         (TIMER0_OVF_vect)
18  SPI Serial Transfer Complete                    (SPI_STC_vect)
19  USART Rx Complete                               (USART_RX_vect)
20  USART, Data Register Empty                      (USART_UDRE_vect)
21  USART, Tx Complete                              (USART_TX_vect)
22  ADC Conversion Complete                         (ADC_vect)
23  EEPROM Ready                                    (EE_READY_vect)
24  Analog Comparator                               (ANALOG_COMP_vect)
25  2-wire Serial Interface  (I2C)                  (TWI_vect)
26  Store Program Memory Ready                      (SPM_READY_vect)
```

The text in brackets are **predefines** that can be used with the ISR(.....) macro if using AVR-GCC in the Arduino IDE. for example,

```
ISR(INT0_vect){
   ...
}
```

(a) Consider the characteristics of the square wave at Pin 13 and complete the tables below.

Timer 0 (8-bit)		
Prescaler	Frequency (Hz)	Duty Cycle (%)
None		
8		
64		
256		
1024		

Timer 1 (16-bit)		
Prescaler	Frequency (Hz)	Duty Cycle (%)
None		
8		
64		
256		
1024		

(b) Create the sketch `Timer0NormalOverflow` from the code presented. Add the `prescalers.h` header file from our course page. Run the sketch and satisfy yourself that the calculated frequencies in your **Timer0** table above are 'reasonable' for the five different prescaler values. The contents of the `prescalers.h` header file appear below.

```
// RSGC ACES: ICS4U
// Prescale constants for ATmega328p Timers
#define T0Stopped 0b00000000    // Timer0 stopped
#define T0psNone  0b00000001    // T0:2^24/2^8  (no prescale)> 2^? ovf/s = ? Hz
#define T0ps8     0b00000010    // T0:2^24/2^3/2^8 (prescale)> 2^? ovf/s = ? Hz
#define T0ps64    0b00000011    // T0:2^24/2^6/2^8 (prescale)> 2^? ovf/s = ? Hz
#define T0ps256   0b00000100    // T0:2^24/2^8/2^8 (prescale)> 2^? ovf/s = ? Hz
#define T0ps1024  0b00000101    // T0:2^24/2^10/2^8(prescale)> 2^? ovf/s = ? Hz
#define T1Stopped 0b00000000    // Timer1 stopped
#define T1psNone  0b00000001    // T1:2^24/2^16  (no prescale)> 2^8 ovf/s > 128Hz
#define T1ps8     0b00000010    // T1:2^24/2^3/2^16 (prescale)> 2^5 ovf/s > 16Hz
#define T1ps64    0b00000011    // T1:2^24/2^6/2^16 (prescale)> 2^2 ovf/s > 2Hz
#define T1ps256   0b00000100    // T1:2^24/2^8/2^16 (prescale)> 1 ovf/s   > 0.5Hz
#define T1ps1024  0b00000101    // T1:2^24/2^10/2^16(prescale)> 0.25 ovf/s> 0.125Hz
#define T2Stopped 0b00000000    // Timer2 stopped
#define T2psNone  0b00000001    // T2:2^24/2^8  (no prescale)> 2^? ovf/s > ? Hz
#define T2ps8     0b00000010    // T2:2^24/2^3/2^8 (prescale)> 2^? ovf/s = ? Hz
#define T2ps32    0b00000011    // T2:2^24/2^5/2^8 (prescale)> 2^? ovf/s = ? Hz
#define T2ps64    0b00000100    // T2:2^24/2^6/2^8 (prescale)> 2^? ovf/s = ? Hz
#define T2ps128   0b00000101    // T2:2^24/2^7/2^8(prescale)> 2^? ovf/s = ? Hz
#define T2ps256   0b00000110    // T2:2^24/2^8/2^8(prescale)> 2^? ovf/s = ? Hz
#define T2ps1024  0b00000111    // T2:2^24/2^10/2^8(prescale)> 2^? ovf/s = ? Hz
```

(c) Save the sketch as `Timer1NormalOverflow` and modify it to run and confirm the same 5 tests for (the 16-bit) **Timer1**.

(d) Use your `2C53T` handheld scope to confirm a few frequencies for both **Timer0** and **Timer1**.

The square wave generation strategy in the previous exercise suffers from a number of shortcomings that include,

- a limited number of frequency options
- no flexibility for duty cycle
- software-only response to Timer/Counter Event trigger

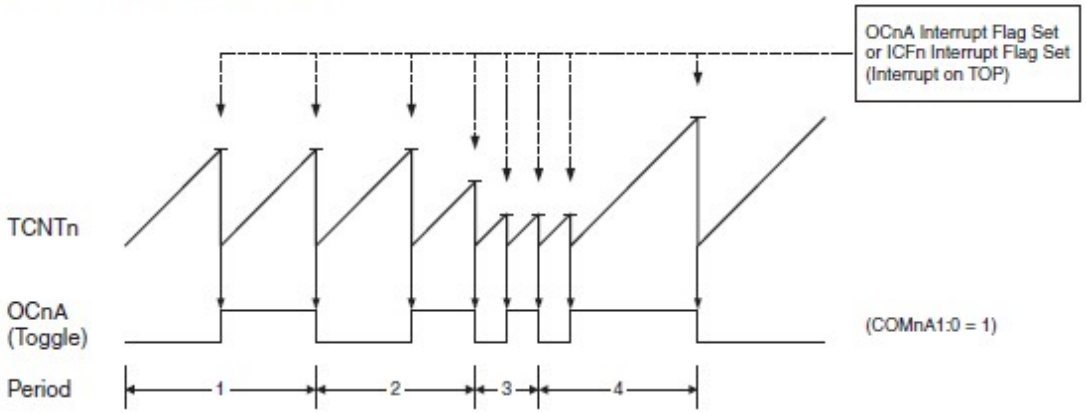
Subsequent exercises address these three issues.

Exercise 2. Timer 1 CTC Mode 4 (Clear Timer on Compare Match)

See: http://darcy.rsgc.on.ca/ACES/Datasheets/ATmega328PTimer1.pdf.
Assume an ATmega328p under the following conditions,

- f_{CLK} = 16 MHz
- 16-bit Timer 1 in **Clear Timer on Compare Match (CTC) Mode** WGM13:0 = 4 (Mode 4)
- OC1A (Pin 9) under COM1A1:0 = 1
- OC1B (Pin 10) under COM1B1:0 = 0
- OCR1A set to 0x7FFF
- OC1A (Pin 9) configured for output and an LED attached to ground for confirmation
- ISR for Timer/Counter1 Compare Match A **disabled** (all hardware for this one)

Figure 16-6. CTC Mode, Timing Diagram



(a) Complete the table under the conditions specified above.
(b) Develop and run a register-level sketch `Timer1CTCMode4` under the conditions specified above.
(c) Confirm the results suggested in the table above with your handheld scope. Adjust your calculations as necessary.
(d) The formula for the oscillation frequency, f_{osc}, under these conditions can be expressed as,

$$f_{osc} = \frac{f_{clk}}{2 \times PS \times (1 + OCR1A)}$$

The middle A (Octave 4) has a frequency of 440 Hz. Using a prescaler (PS) of your choice, determine a value for `OCR1A`, that will provide a 440 Hz square wave signal on `OC1A`. Adjust your sketch and confirm with your scope.

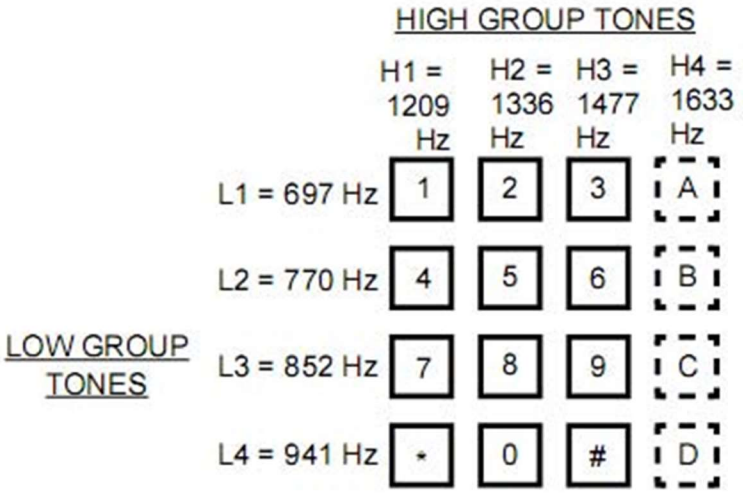
(e) Using the online tone generator linked from our course page to play 440 Hz. Now, connect `OC1A` to the **Tip** pin of your ACES Audio breakout board (see above) plug and listen to the tone on your headphones. Do they match?
(f) Finally, enable an interrupt for the `Timer/Counter1Compare Match A` event and implement the ISR to toggle the ATmega328p's onboard LED.

Timer 1 (16-bit)		
Prescaler	Frequency (Hz)	Duty Cycle (%)
None (`T1psNone`)		
8 (`T1ps8`)		
64 (`T1ps64`)		
256 (`T1ps256`)		
1024 (`T1ps1024`)		

Exercise 3. Dual Tone Multi Frequency (DTMF)

See: https://onlinesound.net/download/frequency-tone-sounds/dtmf

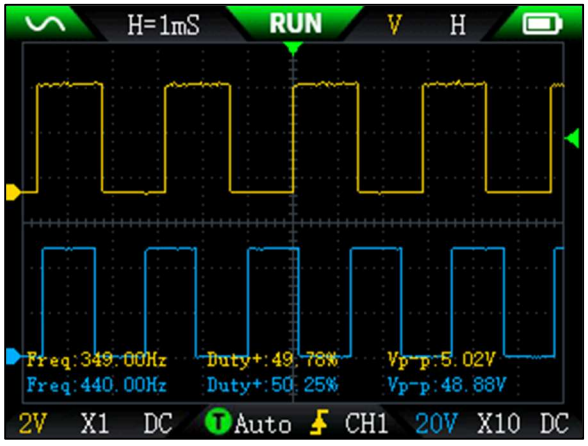
The North American standard for *Dual Tone Multi Frequency* (DTMF) telephony signals was established in 1963 by Bell Laboratories and remains in use today. Each of the 12 keys were assigned a low and high frequency, paired together on transmission, as summarized in the chart below, right.



In addition to the frequency pairs for the 12 telephone keys, a fourth column (A, B, C, D) for optional implementation was established as were frequency pairs for *Dial Tone* (350 Hz and 440 Hz), *Ringing Tone* (400 Hz and 440 Hz) and *Busy Tone* (480 Hz and 620 Hz).

(a) Play the Dial Tone file on our ACES web site, http://darcy.rsgc.on.ca/ACES/TEI4M/WaveFormGeneration/DTMFDialTone.mp3

(b) You are asked to recreate the DTMF **Dial Tone** signal (*square waves for now*) and confirm it both on your headphones and your scope. Here are my suggestions.



(i) Use `Timer1` and `Timer2` for the low and high tones, respectively. Leaving `Timer0` available for the `millis()`, `micros()` and `delay()` functions is advantageous (ie. the Busy Tone with timed pauses).

(ii) Use the CTC mode. `Timer1` (Mode 4) and `Timer2` (Mode2).

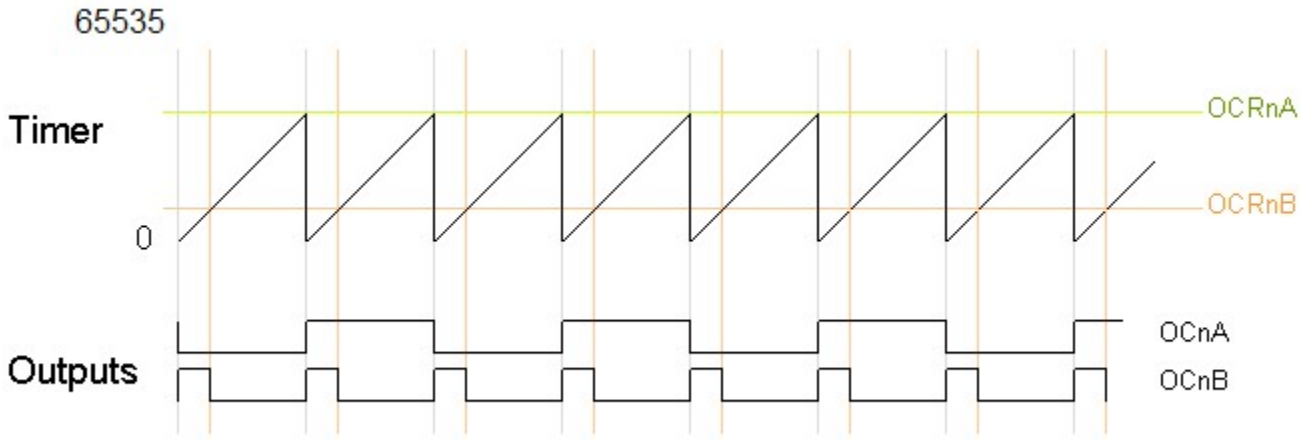
(c) Create the simple mixer as described in the previous section, Audio Jack, Headphone and Simple Mixer and mix the two frequencies together, into your headphones.

Exercise 4. Servo Driving. Fast PWM. Influencing the Duty Cycle

To this point the Normal and CTC Timer modes have limited their influence to *frequency* alone. In this exercise we'll investigate modification of the waveform's *duty cycle* as well and discover the behaviour behind the Arduino's high level `analogWrite()` function to enable *Pulse Width Modulation (*PWM*)* waveforms on select MCU pins.

The investigation below focuses on creating a continuous back and forth sweep of the horn of the FS5103B servo you received in your Grade 11 toolkit (https://www.pololu.com/product/3424).

Of the multiple PWM modes provided by the AVR Timers, **Timer1 Mode 15** provides the best fit for this purpose. In this mode `OCR1A` sets the `TOP` of the counting, thereby affecting the waveform's frequency. `OCR1B` determines the duty cycle. Examine the timing diagram below to appreciate how both characteristics are determined.



This exercise recreates the OC1B (Pin 10) waveform at the bottom of the graph. The signal wire of your FS5103B servo (white) receives this waveform. Here's how Timer 1 Mode 15 unfolds,

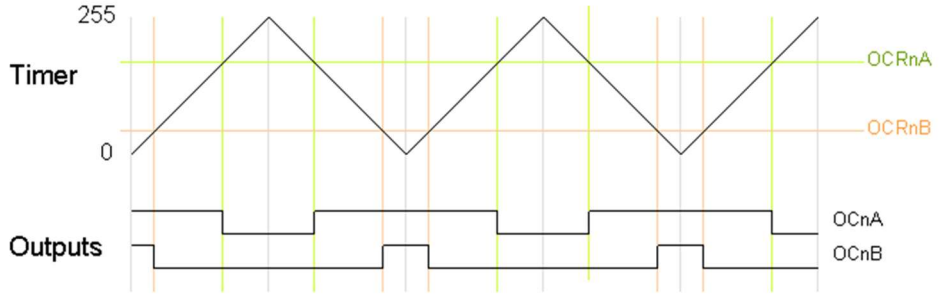
- Timer1's counter, `TCNT1`, counts continuously from 0 to the `TOP` defined by `OCR1A` before starting again at 0 (`BOTTOM`).
- Manipulation of `OCR1A` defines the **frequency**.
- For Servo driving, `OCR1A` should be defined to create a 50 Hz waveform (20 ms period)
- `OC1B` (Pin 10) is configured to be to set at the `BOTTOM` and cleared when `TCNT1` reaches `OCR1B`.
- Manipulation of `OCR1B` defines the **duty cycle**.
- For correct positioning of the Servo's horn, `OCR1B` should be defined to limit the duty cycle to between 5% (0° or 1 ms) and 10% (180° or 2 ms).
- Implementation of the `TIMER1_COMPA_OVF` interrupt can be used to modify `OCR1B`, thereby affecting a horn sweep.

Not all servos are precise enough to respond in an identical manner so some experimentation of the high and lower values for `OCR1B` should be undertaken.

Exercise 5. AnalogWrite() Revisited

Familiarity with AVR Timer architecture allows us to examine the `analogWrite()` function more closely.

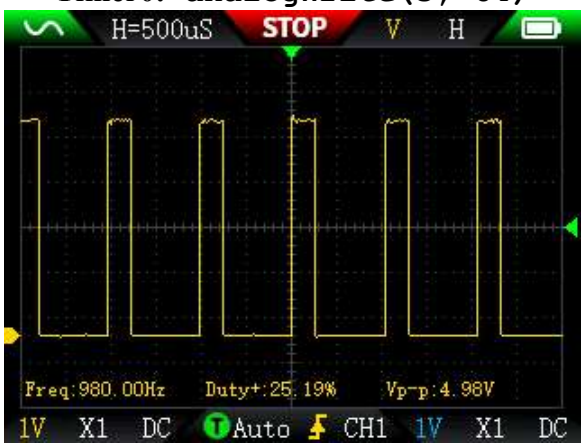
Each of the three Timers offers one or more *Phase Correct* Modes in which the count alternates between increasing and decreasing. Compare matches can be triggered in both directions resulting in periods that are half their one-direction counterparts as shown in the graph below.



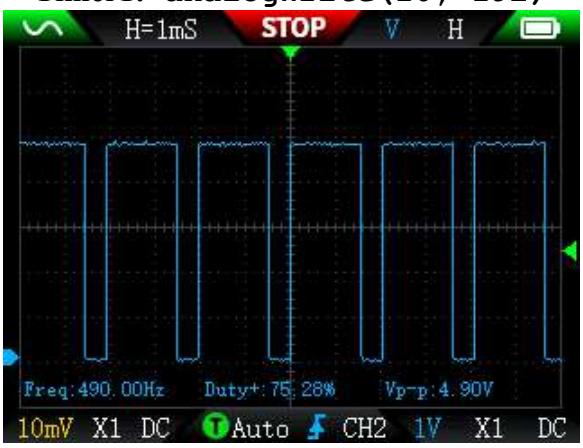
	Timer 0	Timer 1	Timer 2
Digital Pins	5, 6	9, 10	3, 11
TCNTn (bits)	8	16	8
Timer Mode	Fast PWM (3)	PWM, Phase Correct, 8-bit (1)	PWM, Phase Correct, 8-bit (1)
Clock (MHz)	16 000 000	16 000 000	16 000 000
Resolution (steps)	256	256	256
Prescaler	64	64	64
Frequency (Hz)	980	490*	490*

* Divided by 2 due to up/down counting of Phase Correct Mode.

Timer0: `analogWrite(5, 64)`



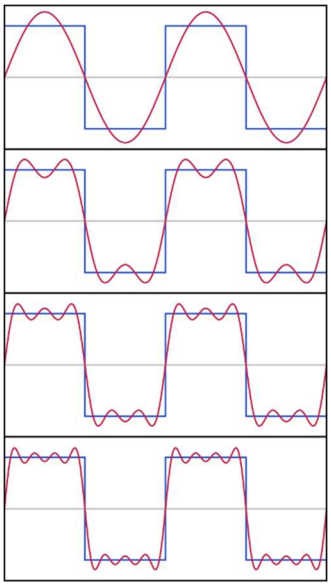
Timer1: `analogWrite(10, 192)`



(a) Implement a test sketch that displays the contents of the `TCCR2A`, `TCCR2B`, and `OCR2B` register value **after** the statement `analogWrite(3,127)`.

(b) Use the results from (a) to develop the sketch `Timer2Sweep` (*register-level as much as* possible) that allows the user to control the duty cycle of a PWM signal on `OC2B` (Pin 3) through the rotation of the shaft of a trim pot. Monitor this on your `2C53T` scope.

Harmonics



See: Fourier Series Demystified:
https://www.youtube.com/watch?v=mgXSevZmjPc

The discovery in the 19th Century by Joseph Fourier that any complex function can be represented by summing a series of sine waves changed the direction of science and technology. Consider a simple square wave in this context for example as decomposed in the graphic to the right. Download the Excel workbook by following the link on the cover of this workbook and explore this concept.

A mathematical process known as *Fourier Analysis* can determine separate the contributory sine waves that comprise a square wave or any other function. Once identified filters and other processes can be applied to manipulate specific sinusoidal constituents that make up the original function.

In mathematics, *the harmonic series* is defined as,

$$\sum_{n=1}^{\infty} \frac{1}{n} = 1 + \frac{1}{2} + \frac{1}{3} + \cdots + \frac{1}{n} + \cdots \to \infty$$

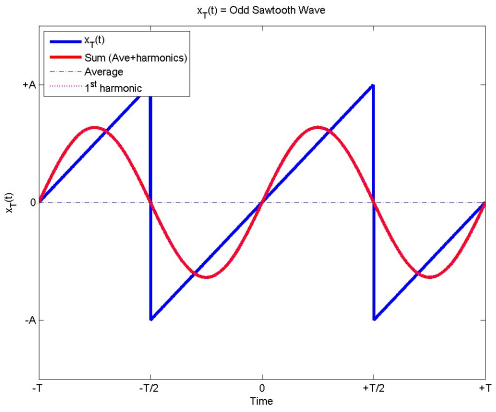
A square wave as a function of time, *t*, in theory, consists of an infinite series of sinusoidal waveforms consisting solely of the odd harmonics and can defined as follows,

$$f(t) = \frac{4}{\pi} \sum_{n=1}^{\infty} \frac{1}{n} \sin k\pi t \text{ , where } k = 2n - 1$$

Why should this matter?

Bandwidth

(AI) Bandwidth in communication is the maximum rate of data transfer across a network path within a specific time, typically measured in bits per second (bps), Mbps, or Gbps. It signifies capacity, not speed, representing how much information can be sent, with higher bandwidth allowing faster transmission.



Adding more harmonics to a signal increases its bandwidth because harmonics introduce additional, higher frequencies into the signal's spectrum. Since bandwidth is defined by the difference between the highest and lowest frequency components present in a signal it would be useful to identify and remove as many high frequency harmonic components without otherwise affecting the quality of the (data) signal.

Filters

See: Impedance and Reactance: https://electronicsclub.info/impedance.htm

Concepts and Terminology

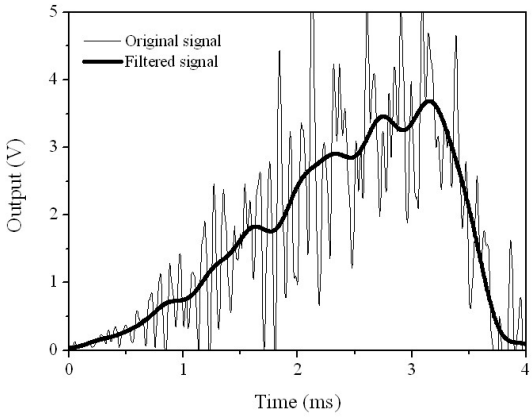
AI: Passive elements are electronic components that consume, store, or dissipate energy rather than generating it, requiring no external power to operate. Key examples include resistors, capacitors, inductors, and transformers. They cannot provide power gain, amplify signals, or control current flow, unlike active components like transistors.

Resistance (***R***) is a property of resistors that provides opposition to the flow of electrons (current) through which electrical energy is dissipated as heat or light.

AI: *Reactance* is the opposition to alternating current (AC) flow caused by inductance or capacitance, measured in ohms (Ω) and denoted by ***X***. Unlike resistance, it does not dissipate power as heat, but rather stores and releases energy in electric or magnetic fields.

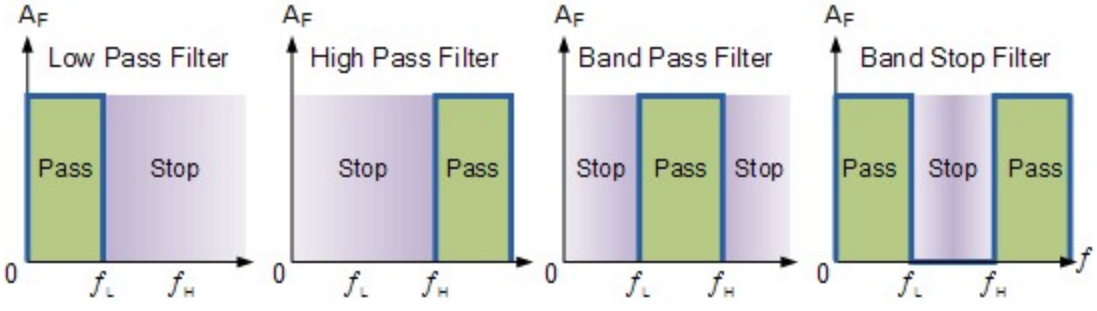
AI: *Impedance* (***Z***) is the total opposition an electrical circuit presents to the flow of alternating current (AC), combining resistance (***R***) with reactance (***X***) from capacitors and inductors, measured in ohms (Ω).

The objective of any **filter** is to remove unwanted elements. In power and communication applications for example, signals are typically comprised of multiple frequencies that crept in unintentionally (noise) or were placed there on purpose (carrier signal plus modulation). The role of an electronic frequency filter is to strip away the unwanted frequencies exposing the desired core or pure frequency. Filters for AC waveforms are constructed using passive elements to allow a desired frequency range to pass through while blocking (attenuating) unwanted frequency ranges.



Filters fall into two general categories: *Passive* and *Active*. These are defined by the nature of their circuit elements.

Low pass filters exploit the allow a low frequency range to pass but *attenuate* higher frequencies. High pass, band pass and notch filters are other common designs. *Passive* low pass filters use passive elements (resistors, capacitors and inductors) to create combined impedance from the resistive and reactive properties of their components. As you may recall from your Grade 10 Capacitor Visualizer project, the capacitor blocked (filtered) the DC voltage source indicated by the red LED turning off. Check out this Falstad simulation: https://tinyurl.com/2djlchtk



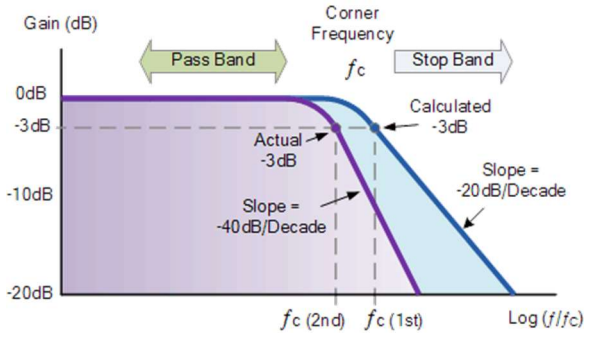
Passive Low Pass RC Filter

A passive low pass RC filter uses a resistor and capacitor in series to allow frequencies below and desired threshold (cut-off or corner) to pass while attenuating (shunting to ground) higher frequencies.

Bode (Pronounced BOH-dee) Plot

See: https://www.youtube.com/watch?v=KX7GNqy3k7w&t=401s

The Bode plot to the right exposes the response of a low pass-filter design.



The purple region suggests a range of lower input frequencies with little to no attenuation in the output frequency. This is the *Pass* band.

The cyan region covers a range of input frequencies for which significant blocking results. This is the *Stop* band.

Bode Plot Details

The horizontal axis is frequency, ω (rad/s), on a logarithmic scale (0.1, 10, 100, 1000, etc.). Each interval (0.1-1, 1-10, 10-100, etc.) is referred to as a *decade*. The vertical axis is Gain (amplitude/attenuation) measured in decibels (dB) where,

$$\text{dB} = 20 \times \log_{10}\left(\frac{A}{A_0}\right)$$

In this formula A is the measured amplitude (output) and A_0 is the reference amplitude (input).

Placing additional low pass filters in series with one another steepens the attenuation. With a 1st order low pass filter (one RC pair) attenuation can be expected at a rate of -20 dB/ decade. Adding a second RC pair (2nd order) can be expected to deliver a -40 dB reduction per decade.

Cut-off (Corner) Frequency

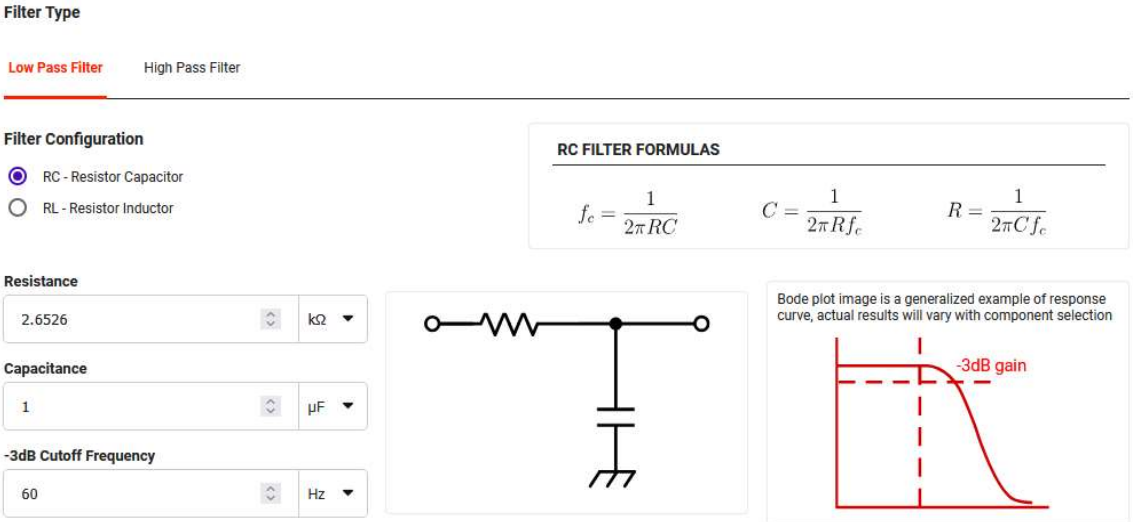
The frequency threshold between passing and blocking is the *cut-off* (corner) or frequency (f_c). It is the frequency at which there is a 50% reduction in signal power (-3 dB) representing the minimum change in sound volume that the human ear can easily distinguish.

General guidelines exist for determining an appropriate a cut-off frequency, f_c, as follows,

1. The f_c should be above the range of frequencies to be passed through and then some for the -3dB attenuation.
2. The f_c should be significantly lower (10×) than the frequency of the carrier frequency or noise range that is to removed.
3. In digital monitoring situations the f_c should be less than half of the sampling frequency (Nyquist Criteria).
4. The f_c can be closer to the frequency range of the passed signals if higher order filters are employed as they have a steeper roll-off (1st order: -20 dB/decade, 2nd order: -40 dB/decade, 3rd order: -60 dB/decade).

Passive Low Pass Filter Design

Low Pass/High Pass Filter Calculator: https://www.digikey.ca/en/resources/conversion-calculators/conversion-calculator-low-pass-and-high-pass-filter



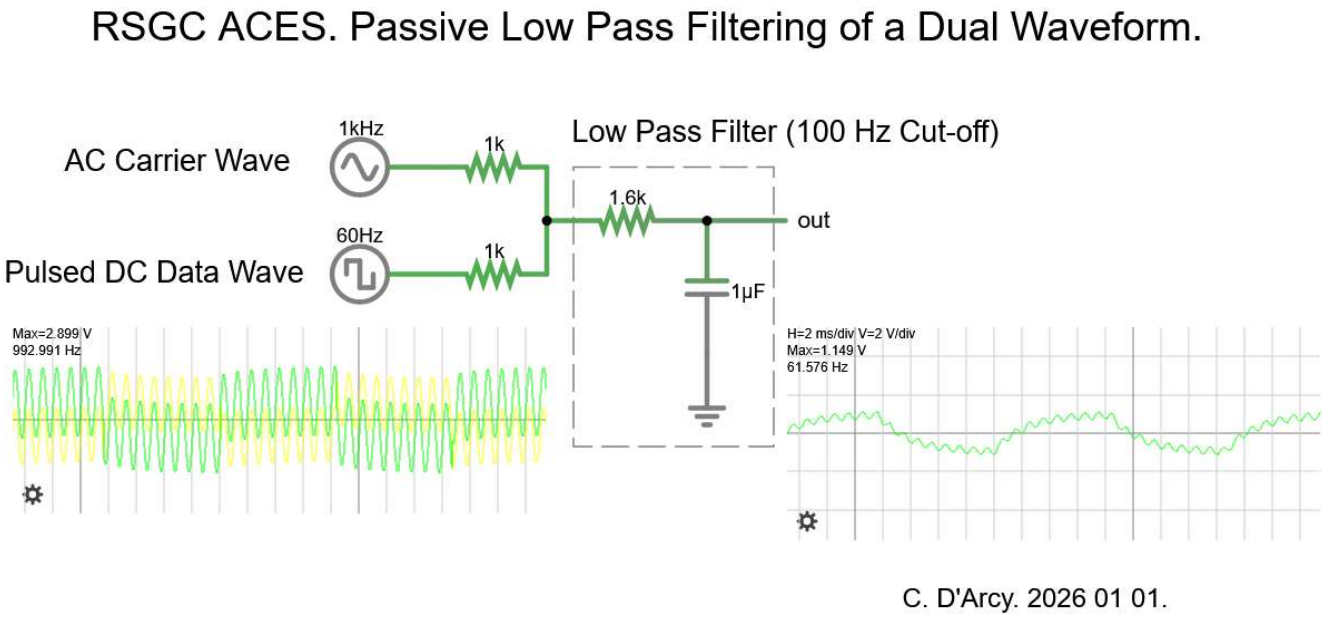
The formula for the cut-off frequency is,

$$f_c = \frac{1}{2\pi RC}$$

Example

Falstad: https://tinyurl.com/25z7dns5

Consider a 60 Hz PWM data signal added to a 1 kHz AC carrier wave prior to transmission. To expose the data signal the receiver must remove (filter away) the higher frequency carrier wave. Review the Falstad simulation. Modify the carrier wave to simple noise and observe the output.



Investigation. Stable Analog Voltage from Filtered PWM

"*Having a component expect an analog voltage signal and receiving a PWM is similar to asking for a gentle massage on the head and getting hit with a hammer every 20 seconds (the average will be the same but… it's not the same).*" – Luis Llamas.

References

- Secrets of Arduino PWM: https://www.righto.com/2009/07/secrets-of-arduino-pwm.html
- Arduino PWM and `analogWrite()` Explained: https://controllerstech.com/arduino-pwm-analogwrite-tutorial/
- Impedance: https://electronicsclub.info/impedance.htm
- How Low Pass Filters Work: https://www.youtube.com/watch?v=oHKwwvcn77Y
- Analog Output from PWM and a Low Pass Filter: https://www.luisllamas.es/en/analog-output-pwm-low-pass-filter/
- Analog output from PWM and an LPF: https://www.youtube.com/watch?v=Fsb7kxDxYGw

Fundamental Concepts

(a) A PWM waveform can be treated mathematically as consisting of a DC component (the average voltage defined by the duty cycle) together with AC components (harmonics).

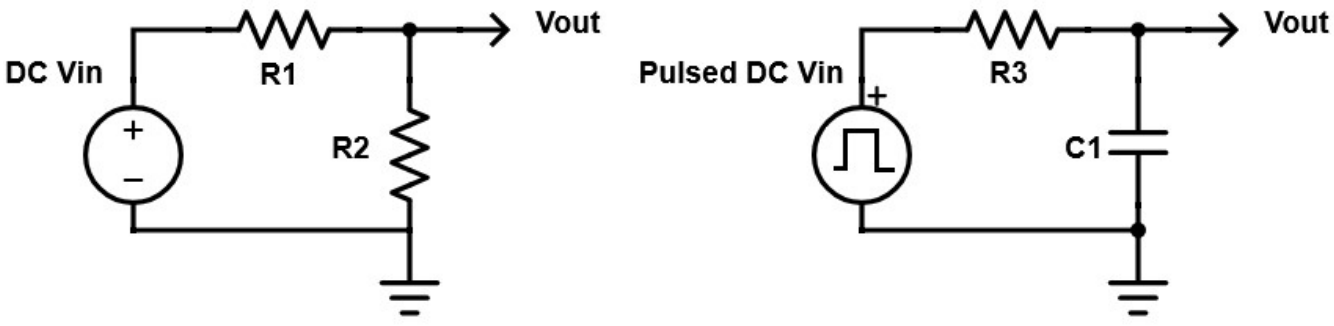
(b) Opposition to current flow by the three passive components is called *impedance* (Z).
(i) Impedance in **resistors** is simply called *resistance* (R).
(ii) Impedance in capacitors is called capacitive reactance (X).
(iii) Impedance in inductors is called inductive reactance (L).
(iv) Impedance is a combination of resistance and reactance and measured in Ohms (Ω).
(v) Impedance lends itself to Ohm's Law.

(c) The output of a **voltage divider** constructed from two or more passive components in series (R, L, or C) is of interest here and can be expressed by (1),

$$V_{\text{out}} = V_{\text{in}} \times \frac{Z_2}{Z_1 + Z_2}$$

(d) The Grade 10 course introduced the **R-R voltage divider** concept resulting from two resistors in series under steady DC voltage (below, left). The DC voltage out is given by (2),

$$V_{\text{out}} = V_{\text{in}} \times \frac{R_2}{R_1 + R_2}$$



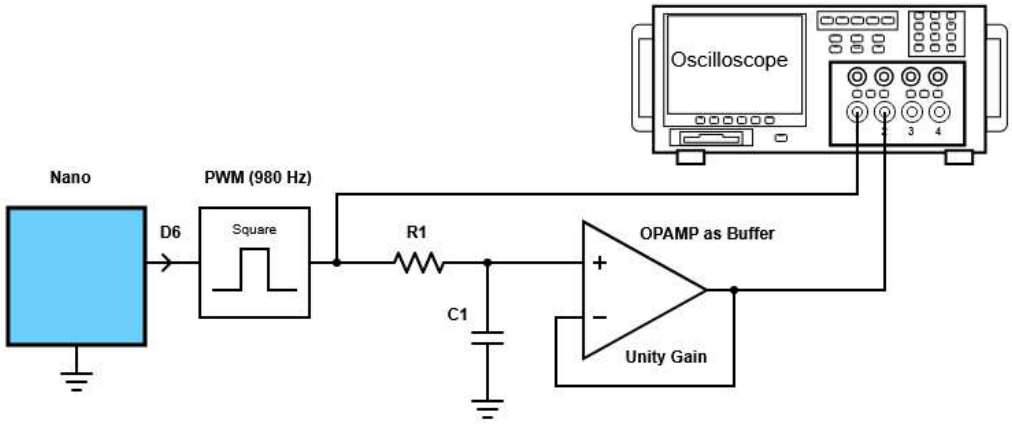
(e) The Arduino's `analogWrite()` function produces a bistable, pulsed DC square wave intended to mimic an average analog voltage over the time domain. While not strictly an AC signal in that it does not change polarity, a pulsed DC can be analyzed mathematically as such as a combination of a stable DC offset with ripple (AC component).

(f) Capacitors are *frequency dependent* in their opposition to current that decreases with increased frequencies. The formula for capacitor reactance is given by (3),

$$X_C = \frac{1}{2\pi f C}$$

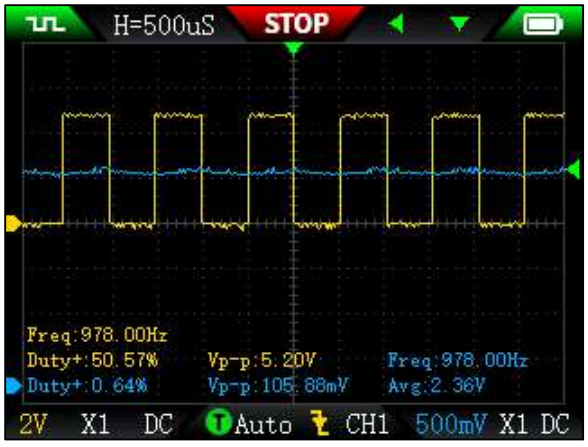
(g) A Passive Low-Pass Filter (LPF) is an **R-C voltage divider**. With a properly constructed LPF the square edges of the wave can be made to produce a smooth, constant analog voltage (above, right).

Exercise 6. Stable Analog Voltage from AnalogWrite () with Buffering



In this exercise you are asked to filter a PWM square wave through a passive low-pass filter to create a stable analog voltage.

Reference: How Low Pass Filters Work (https://www.youtube.com/watch?v=oHKwwvcn77Y)



(a) Assemble a prototype from the schematic above. Implement the sketch `AnalogStable.ino` to output a PWM signal through the use of `analogWrite(6,128)` to produce a 980 Hz square wave with a 50% duty cycle. Present this signal on `CH1` of your scope to confirm.

(b) Design a low-pass filter with a suitable cut-off frequency. An adjustable potentiometer for `R1` would be wise.

(c) For additional load stability (*high input impedance and low output impedance*) add a non-inverting voltage follower to the output of your filter. Monitor this filtered signal on `CH2` of your scope to confirm and compare.

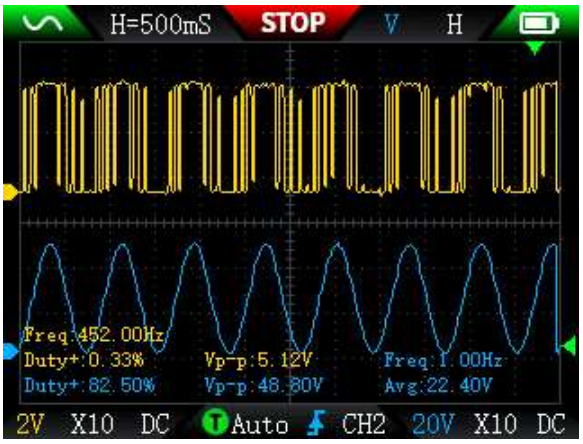
Exercise 7. Sinusoidal Pulse Width Modulation (SPWM)

The goal of the previous exercise was to generate and maintain a **stable** analog voltage from a PWM (pulsed DC) source. Since **sinusoidal** waveforms have broad applications it would be useful to create one from a similar PWM source.

Reference:

- Sine Look Up Table Generator Calculator: https://www.daycounter.com/Calculators/Sine-Generator-Calculator.phtml

Examine the scope capture to the right. The `CH1` trace at the top depicts a PWM signal with varying duty cycle on digital pin 3. Applying a low pass filter to this signal smooths it out into sinusoidal form as captured on `CH2`.



(a) Use the reference above to generate a Sine Lookup table with 256 points and a maximum amplitude of 255.

(b) Create the sketch `SPWM` that uses register-level code to establish Fast PWM Mode 3 on Timer2. Configure `OC2B` to clear on a Compare match. Use a 128 prescaler for now. Monitor the state of `OC2B` (Pin 3) in `CH1`.

(c) Within the `loop()` function, increment a `count` variable by 1 to `index` the sine lookup table created in step (a). Justify, arithmetically, the scope's frequency determination.

(d) On your prototype add a low-pass filter (1 kΩ resistor and 10 µF capacitor) to smooth the PWM signal and capture it on `CH2`. Determine, arithmetically, this filter's cutoff frequency.

(e) Press the left and right arrows to find the suitable horizontal period of 10 mS.

(f) Add a potentiometer to span `A0-A2`. Use the ADC reading from `A1` to change the `count` increment to a range from [1,4]. This should influence the frequency of the two waveforms. Test it.

Common Applications of Low Pass Filters (AI)

- **Noise Reduction in Communication Systems**: Removes unwanted high-frequency interference in radio and satellite communications to improve signal integrity
- **Signal Smoothing in Power Supplies**: Filters out high-frequency ripple and noise from AC-to-DC conversion, ensuring a steady, clean DC voltage.
- **Telephone Networks (DSL Splitters)**: Separates voice signals (low frequency) from DSL data signals (high frequency) on the same wire.
- **Musical Effects and Synthesizers**: Used in analog/virtual synthesizers (subtractive synthesis) and as tone knobs on electric guitars to reduce "fizz" or high-end hiss.
- **Equalizers and Audio Amplifiers and Analyzers**: Shapes the frequency response of sound systems to reduce treble or balance output.

Direct Digital Synthesis (DDS)

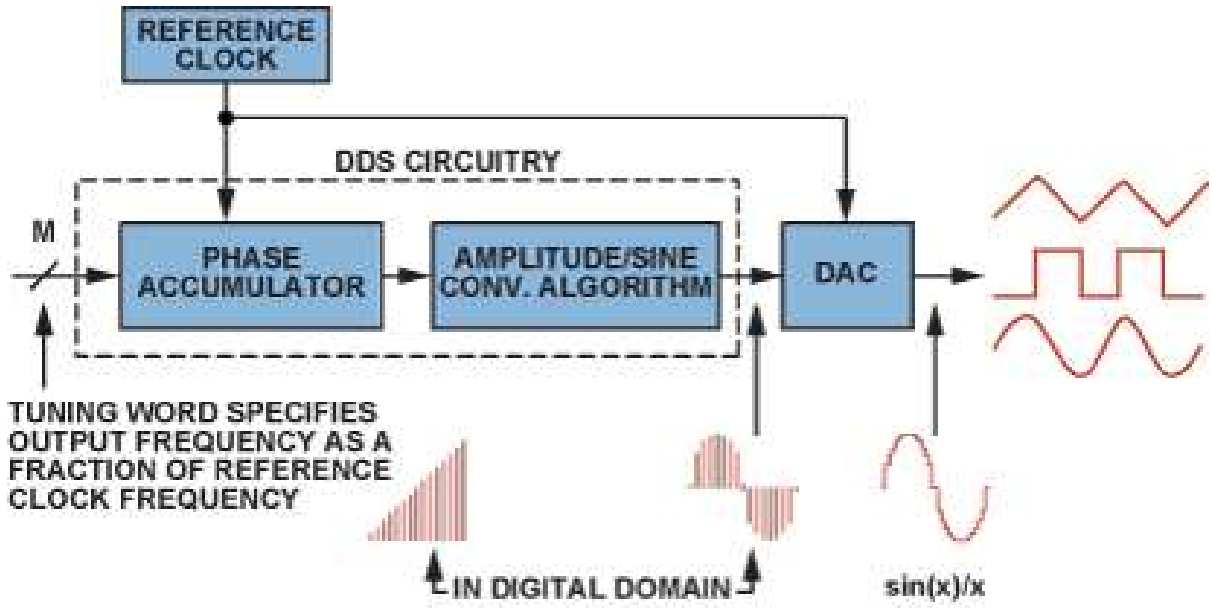
Reference:

- How DDS Works: https://www.youtube.com/watch?v=WwHouxujrNc

The most common algorithm employed by modern signal generators, including your hand-held 2C53T, to produce their variety of waveforms is referred to as *Direct Digital Synthesis* (DDS). This relatively simple algorithm employs a look up table (LUT) of typically 256 elements that form the basic steps in a single period of the defined waveform. The waveform is generated by stepping through the table entries, outputting an 8-bit binary number. This value is input to a DAC (an R/2R ladder would suffice) to obtain a remarkably accurate analog voltage approximation. Additional filtering through a passive RC filter smooths out the ripple even further.

A reference clock, $\mathbf{F_{CLK}}$, controls the pacing of the algorithm.

A step size of 1 on each tick of the reference clock (each element in the LUT is used once in each cycle) provides the fundamental frequency. Increasing the step size on each clock results in the moving through the LUT faster (elements in the LUT are skipped), thereby increasing the frequency. Limiting the step increment to multiple clock ticks reduces the frequency of the waveform output. The step size is referred to as the *tuning word*, *M*.



The index into the LUT is maintained by a variable referred to as the *phase accumulator*. With each tick of the reference clock, the tuning word is added to the phase accumulator. Assuming the phase accumulator in a ***n***-bit binary number, it can be determined between the reference clock and the tuning word how many steps are required to overflow the phase accumulator, hence defining the frequency of the output waveform, F_{OUT}. This relationship can be expressed as follows (4),

$$F_{OUT} = \frac{M \times F_{CLK}}{2^n}$$

Exercise 8. DDS: Sawtooth (Ramp)

Exercise 9. DDS: Square Wave

Exercise 10. DDS: Triangle Wave

Exercise 11. DDS: Sine Wave

This investigation explores a DAC strategy resulting in a sinusoidal waveform from digital input.

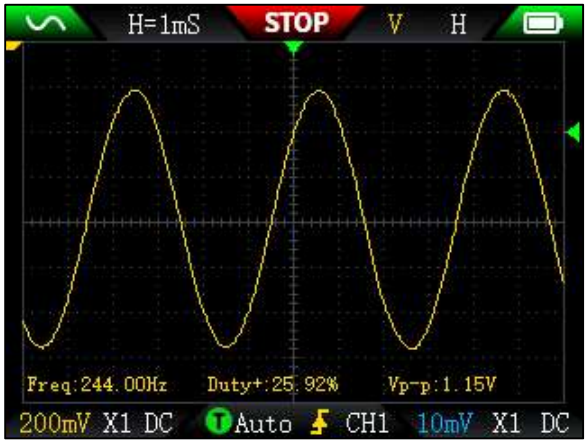
Reference:

- How DDS Works: https://www.youtube.com/watch?v=WwHouxujrNc
- Sine Look Up Table Generator Calculator: https://www.daycounter.com/Calculators/Sine-Generator-Calculator.phtml

(a) Watch the Youtube video above.

The only full 8-bit I/O Port on the Nano is the ATmega328p's PORT D that maps to digital Pins 0 through 7. We'll use the Sparkfun Pocket Programmer's on the Nano's ISP header to have access to the full 8 bits.

(b) Wire all of `PORTD` to the R2R Resistor network you have been supplied with (10X-R2R-103LF) and place a 0.1 µF capacitor on the output of the R2R ladder to smooth the waveform (see video). Connect your scope to the output of the R2R ladder.



(c) Create the sketch `DDSSineWave`. Maximizing the use of register-level code, translate the code provided in the video to create a sine wave similar to the capture to the right. The link above can be used to generate a Lookup Table (LUT) of Sine values that can been impressed upon `PORTD`. Initially, step through the table with a count value of one.

(d) Add a potentiometer as a voltage divider to your prototype that maps its output to the range from 1 to 8. Use this mapped value as the count value to vary the step size. Monitor the change in frequency on your scope.

Exercise 12. DDS: 2C53T Signal Generation

Your `2C53T` offers a selection of 13 Output Waveforms that include Sine Wave, Square Wave, Saw tooth Wave, Half Wave, Full Wave, Step Wave, Reverse Step Wave, Index Up, Index Decrease, Direct Current, Multi-audio, Sink Pulse and Lorentz Wave. Where appropriate, selectable parameters include frequency, duty cycle and amplitude. Pressing the pause button for a few seconds toggles the on/off feature of the signal. As far as I can tell, the Save screen feature is inactive for the Signal Generator.

(a) Cable your headphones to the ACES Audio breakout board. Load the Signal Generator menu on your scope. Set the frequency to 440 Hz and an amplitude of 1 V. Scroll through the various waveforms and listen for identifiable features of each.

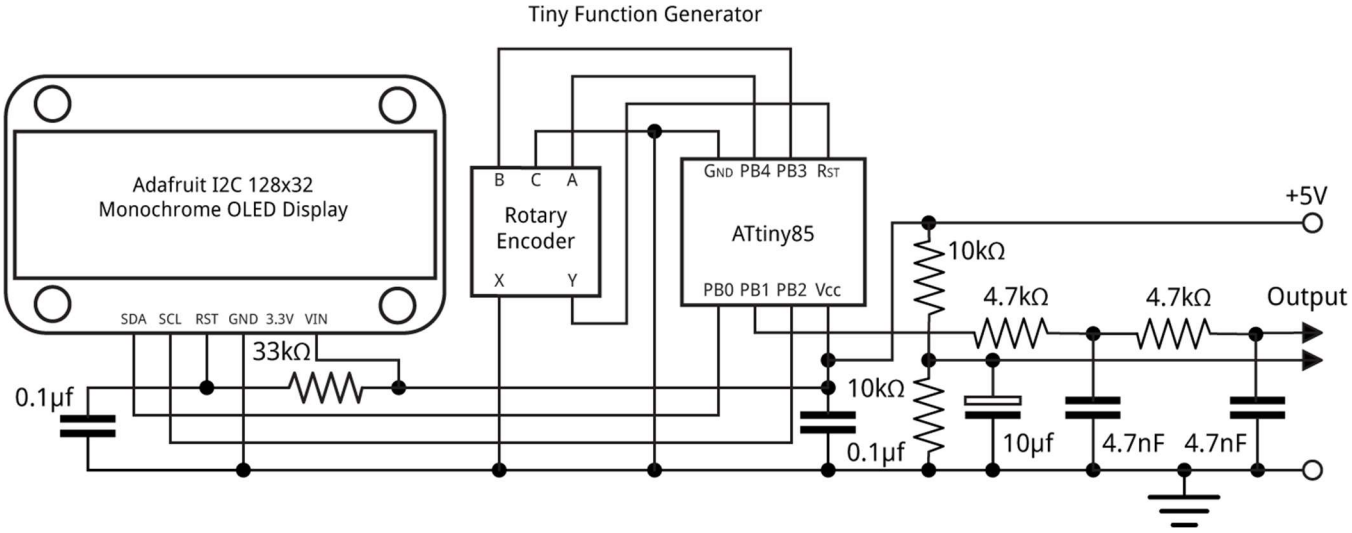
Project. DDS: ATtiny85 Filtered PWM

References

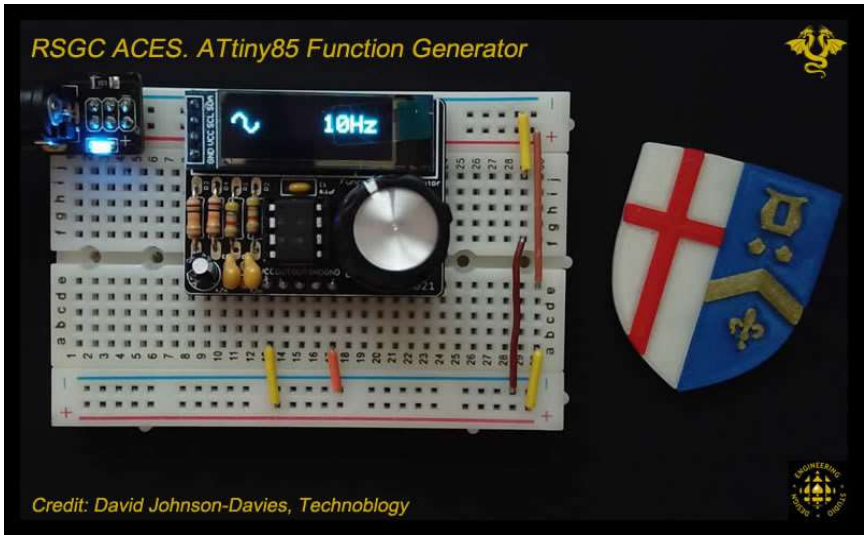
- Technoblogy: ATtiny85 Function Generator: http://www.technoblogy.com/show?20W6
- Sine Look Up Table Generator Calculator: : https://www.daycounter.com/Calculators/Sine-Generator-Calculator.phtml

One of the most accessible DDS waveform projects for ACES comes form David Johnson-Davies' Technoblogy that employs an ATtiny85-based lookup table to generate waveforms from filtered PWM output. Here's his project's opening description followed by his schematic for his PCB,

This article describes a simple function generator based on an ATtiny85. It can generate triangle, sawtooth, square, and rectangular waves, a pulse train, and noise. The frequency can be adjusted using a rotary encoder between 1Hz and 5kHz in steps of 1Hz, and the selected waveform and frequency is displayed on an OLED display.



Inspired by the project, an ACES class set of PCBs were ordered and populated in January 2021.



ISP Idea: ACES Spectrum Analyzer

Once every few years a project emerges that captures the interest and imagination of a group of Grade 12 ACES. In 2015 it was RSGC's 50th Anniversary Clock project that each ACE received. In 2019 it was the Desktop fan that each graduate took home with him.

The quirky 8-bit MSGEQ7 IC provided the inspiration for a number of spectrum analyzer projects including G. Davidge's remarkable 8-Channel Graphic Equalizer:

https://www.youtube.com/watch?v=Vl6h5RWQ3bs

Indeed, it was this project that provided the basis for the compiling of this workbook.

(**AI**) Consider a typical structure and core curriculum embodied within an electrical or computer engineering (ECE) undergraduate program,

1. **First Year (Foundations)**: Calculus, linear algebra, physics (mechanics, electricity/magnetism), programming (e.g., C, Python), and introduction to engineering design.
2. **Second Year (Core Engineering)**: Circuit analysis (nodal/mesh), digital logic design, electronic devices, signals and systems, and differential equations.
3. **Third Year (Advanced Theory)**: Electromagnetics, analog/digital communications, control systems, microprocessors/embedded systems, and probability/statistics.
4. **Fourth Year (Specialization & Design)**: Power systems, digital signal processing (DSP), robotics, or machine learning, along with a mandatory capstone project.

With this in mind I would welcome one or more ACES leveraging the momentum provided within this workbook to pursue the development of an **ACES desktop audio responsive spectrum analyzer**, from the ground up.

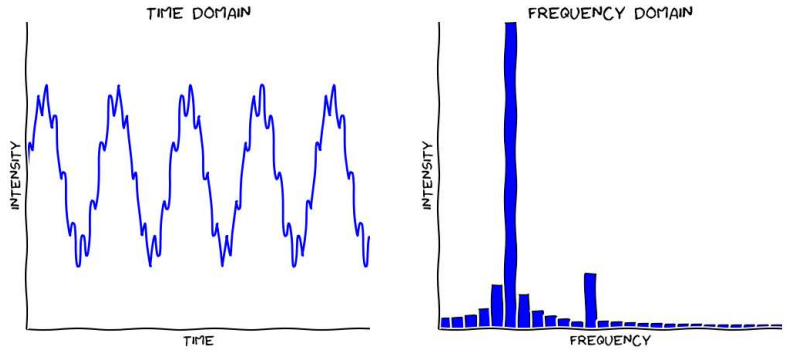
This project would prepare you remarkably well for an ECE program while embracing all three preferred domains. Should one or more ACES take on this project the likelihood that the encasement would be CNC milled by one of our ACES partners is a distinct possibility. Just imagine how it would look on your desk at university…

ACES curious enough to take the next step in a project might consider two paths. The first would a hardware strategy involving passive and active filtering around center frequencies. The second is a software approach requiring more mathematics in support of Fourier Transforms. An introduction to the latter is presented on the following page.

Fourier Transforms

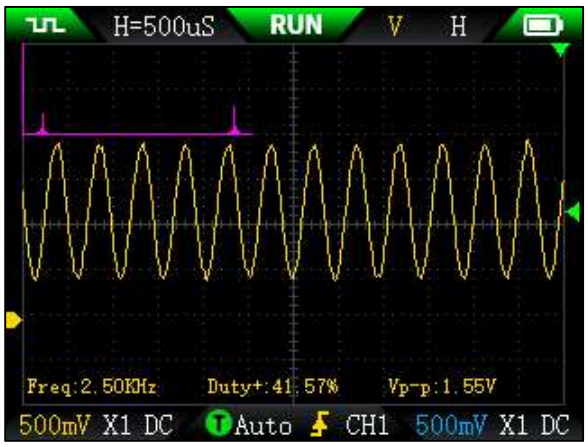
Reference:

- Fun with Fourier Transforms: https://learn.adafruit.com/fft-fun-with-fourier-transforms/background



Your hand-held scope has the ability to introduce you to the FFT. The purple graph in the top left corner shows two significant results for this pure 2.5 KHz sine wave. The leftmost one is its DC component (the average power of the signal) and rightmost reflects the strength of the 2.5 KHz frequency.

The pure sine wave was generated by an online tone generator and picked up by the Adafruit MAX9814 microphone.



Appendix

Video Gold

- Cédric Villani on Joseph Fourier's 'mathematical poem': https://www.youtube.com/watch?v=Ue-riHFsdlk
- How Low-Pass Filters Work: https://www.youtube.com/watch?v=oHKwwvcn77Y
- How Capacitors Work as Filters: https://www.youtube.com/watch?v=ZyS5CQJgEuA
- Direct Digital Synthesis - How DDS Works and an Arduino Example: https://www.youtube.com/watch?v=WwHouxujrNc
- Direct Digital Synthesis - How it Works and a Demo on Arduino: https://www.youtube.com/watch?v=xiby-mkIgQ0
- The Fourier Series and Fourier Transform Demystified: https://www.youtube.com/watch?v=mgXSevZmjPc
- Fourier Series for Beginners Pt. 1: https://www.youtube.com/watch?v=DFRBL2FsTC4

Formulas

Gain (Amplitude/Attentuation)

$$\mathrm{dB} = 20 \times \log_{10}\left(\frac{A}{A_0}\right)$$

Timer/Counter CTC Oscillation Frequency

$$f_{\mathrm{osc}} = \frac{f_{\mathrm{clk}}}{2 \times \mathrm{PS} \times (1 + \mathrm{OCR1A})}$$

Voltage Divider Output

$$V_{\mathrm{out}} = V_{\mathrm{in}} \times \frac{Z_2}{Z_1 + Z_2}$$

Capacitive Reactance

$$X_C = \frac{1}{2\pi f C}$$

Direct Digital Synthesis (M is the Tuning Word)

$$F_{OUT} = \frac{M \times F_{CLK}}{2^n}$$

Euler's Formula

$$e^{i\pi} = \cos\pi + i\sin\pi$$

Euler's Identity

$$e^{i\pi} + 1 = 0$$