Teensy 2.0, Teensy LC, Teensy 4.0, 4.1 and Seeed XIAO SAMD21

Collection of examples and useful hints

Michael Koch

Version from March 30, 2025

Content

1 Integrated Development Environment

1.1 IDE Software installation

First install the Arduino IDE: https://www.arduino.cc/en/software

Then follow the instructions for installation of Teensyduino: https://www.pjrc.com/teensy/td_download.html

1.1.1 test

1.2 Assembly code listing

If you're using the Arduino IDE, using the Sketch->Export Compiled Binary menu option will generate a *.lst file in the sketch directory containing both the C code and corresponding assembly.

The assembly listing can be found in this path: C:\Users\username\Documents\Arduino\project_name\build\teensy.avr.teensy40\project_name.ino.lst

1.3 CPU Speed

CPU speed can be set in Tools --> CPU Speed

See also https://forum.pjrc.com/threads/72339-Teensy-4-1-CPU-frequency-steps

See also https://forum.pjrc.com/threads/59501-Teensy-4-0-Theoretical-Max-Clock-Speed

1.4 How to install Libraries

Step 1: Downloaded a library (from Github, for example) as a ZIP file.

Step 2: In Arduino IDE, click on Sketch > Include Library > Add .ZIP Library.

This procedure is also explained here: https://wiki.seeedstudio.com/How_to_install_Arduino_Library/

```
#include <LiquidCrystal_I2C_4004.h>    // means the file is searched in the library path
#include "LiquidCrystal_I2C_4004.h"    // means the file is searched in the local path
```

1.5 Problems

Error message: "Downloading index: package_renesas_1.2.0_index.json" or any other file that doesn't download.html

Solution: This is a bug in the Arduino IDE. As a workaround, close the Serial Monitor until the download process is finished. After that, you can open Serial Monitor again.

2 Functions, alphabetically sorted

2.1 analogReadResolution();

Sets the resulution of analog inputs in bits. For Teensy 4.0 the default is 10.

2.2 analogReference()

Doesn't exist in Teensy 4.x, because the 3.3V supply voltage is used as reference.

2.3 asm volatile("nop;");

The duration of the "nop" command is in most cases not predictable.

Alternative syntax (I don't know why it's different): __asm__ __volatile__ ("nop\n\t");

2.4 delay()

delay(uint32 milliseconds);

2.5 delayMicroseconds()

delayMicroseconds(uint32 microseconds);

2.6 delayNanoseconds()

delayNanoseconds(uint32 nanoseconds);

Possible workaround for nanoseconds delay, for controllers where the above function isn't available:

```
for (int i = 0; i < NNN; i++)
  asm("nop");
```

or:

```
for (int i = 0; i < NNN; i++)
  asm volatile("nop;");
```

See also: https://www.eevblog.com/forum/microcontrollers/delay-function-atmel-sam-d21-without-asf/

2.7 digitalRead()

digitalRead() is quite slow in Seeed XIAO SAMD21. This workaround is faster:

```
// slow:

if (digitalRead(myPin) == 0)
{
}

// faster:

if (PORT->Group[g_APinDescription[myPin].ulPort].IN.reg & (1ul << g_APinDescription[myPin].ulPin))
{
}

// even faster, if "port" and "pinMask" are constants:

if (PORT->Group[port].IN.reg & pinMask)   // true if pin is high
{
}
```

For the values of "port" and "pinMask", see the list in next chapter.

Found here: https://forum.seeedstudio.com/t/direct-port-access-in-arduino-ide/253807/5

2.8 digitalWrite()

2.9 digitalWriteFast()

Same as digitalWrite(), but faster. No library required. It's undocumented.

It's not available for Seeed XIAO SAMD21, but you can use this workaround for writing pins faster:

```
volatile EPortType port = g_APinDescription[D10].ulPort;
volatile uint32_t pin = g_APinDescription[D10].ulPin;
volatile uint32_t pinMask = (1ul << pin);

void setup()
{
  pinMode(D10, OUTPUT);      // initialize the digital pin as an output
  Serial.begin(38400);
  delay(2000);              // give the serial monitor some time to get ready for listening
  Serial.println(port);     // this is 0 for pin D10
  Serial.println(pinMask);  // this is 64 for pin D10
}

void loop()
{
  digitalWrite(DI, HIGH);                  // set pin high, this takes about 1.5us
  digitalWrite(DI, LOW);                   // set pin low,  this takes about 1.5us
  PORT->Group[port].OUTSET.reg = pinMask;  // set pin high, this takes about 0.5us
  PORT->Group[port].OUTCLR.reg = pinMask;  // set pin low,  this takes about 0.5us
  PORT->Group[0].OUTSET.reg = 64;          // set pin high, this takes about 0.2us (sometimes even faster if in cache?)
  PORT->Group[0].OUTCLR.reg = 64;          // set pin low,  this takes about 0.2us (sometimes even faster if in cache?)
}
```

Found here: https://forum.seeedstudio.com/t/direct-port-access-in-arduino-ide/253807/5

This is a list of the values of "port" and "pinMask" for all 11 pins ot the Seeed XIAL SAMD21:

Pin	port	pinMask
D0	0	4
D1	0	16
D2	0	1024
D3	0	2048
D4	0	256
D5	0	512
D6	1	256
D7	1	512
D8	0	128
D9	0	32
D10	0	64

2.10 IntervalTimer

IntervalTimer allows float. You're not limited to integer microseconds. So you can use timer.begin(myfunction, 22.6757) to get 44100 Hz.
Of course it will round off to the nearest number of F_BUS clock cycles, which is 150 MHz (for Teensy 4.0) when the CPU runs at 600 MHz.

Source code: IntervalTimer.h : https://github.com/PaulStoffregen/co...ntervalTimer.h

2.11 pinmode()

pinMode(led, OUTPUT); //Teensy 4.0: maximal 4mA pro Ausgangspin, für eine rote LED 470 Ohm verwenden

pinMode(5, OUTPUT_OPENDRAIN);

pinMode(A0, INPUT_DISABLE); // use this mode for analog inputs in Teensy (not required in Seeed XIAO SAMS21)

pinMode(5, INPUT_PULLUP); // Pullup resistor

pinMode(5, INPUT_PULLDOWN); // Pulldown resistor

2.12 random()

long random() returns a (signed) long random number.

unsigned long random(unsigned long x) returns an unsigned long random number smaller than x.

If you need full-range uint32 random numbers, you can use this code:

```
uint64_t  prng_state;

uint32_t  prng_u32(void)
{
    uint64_t  x = prng_state;
    x ^= x >> 12;
    x ^= x << 25;
    x ^= x >> 27;
    prng_state = x;
    return (x * UINT64_C(2685821657736338717)) >> 32;
}
```

The state must be initialized at the beginning to a non-zero value.

2.13 Read multiple pins simultaneously

You can read in multiple pins at the same time. Typically after startup all of the pins will be on GPIO ports 6-9

So you could read in all of the pins on port 6 using either:
uint32_t port6_val = GPIO6_PSR;
or IMXRT_GPIO6.PSR;

There are several ways to find out which pins or on which port, I often look that the spreadsheet I setup during the beta tests of the board. Example page in GPIO pin order.

Pin	Name	GPIO	Serial	I2C	SPI	PWM	CAN	Audio	XBAR	FlexIO	Analog	SD
1	AD_B0_02	1.02	Serial1(6) TX		SPI1(3) MISO	PWM1_X0	2_TX		IO-16			
0	AD_B0_03	1.03	Serial1(6) RX		SPI1(3) CS0	PWM1_X1	2_RX		IO-17			
24/A10	AD_B0_12	1.12	Serial6(1) TX	Wire2(4) SCL		PWM1_X2					A1:1	
25/A11	AD_B0_13	1.13	Serial6(1) RX	Wire2(4) SDA		PWM1_X3	GPT1_CLK				A1:2	
19/A5	AD_B1_00	1.16	Serial3(2) CTS	Wire(1) SCL		QT3_0				3:0	A1:5, A2:5	
18/A4	AD_B1_01	1.17	Serial3(2) RTS	Wire(1) SDA		QT3_1				3:1	A1:6, A2:6	
14/A0	AD_B1_02	1.18	Serial3(2) TX			QT3_2		SPDIF_OUT		3:2	A1:7, A2:7	
15/A1	AD_B1_03	1.19	Serial3(2) RX			QT3_3		SPDIF_IN		3:3	A1:8, A2:8	
17/A3	AD_B1_06	1.22	Serial4(3) TX	Wire1(3) SDA				SPDIF_LOCK		3:6	A1:11, A2:11	
16/A2	AD_B1_07	1.23	Serial4(3) RX	Wire1(3) SCL				SPDIF_EXTCLK		3:7	A1:12, A2:12	
22/A8	AD_B1_08	1.24				PWM4_A0	1_TX			3:8	A1:13, A2:13	
23/A9	AD_B1_09	1.25				PWM4_A1	1_RX	1:MCLK		3:9	A1:14, A2:14	
20/A6	AD_B1_10	1.26	Serial5(8) TX					1:RX_SYNC		3:10	A1:15, A2:15	
21/A7	AD_B1_11	1.27	Serial5(8) RX					1:RX_BCLK		3:11	A1:0, A2:0	
26/A12	AD_B1_14	1.30			SPI1(3) MOSI			1:TX_BCLK		3:14	A2:3	
27/A13	AD_B1_15	1.31			SPI1(3) SCK			1:TX_SYNC		3:15	A2:4	
10	B0_00	2.00			SPI(4) CS0	QT1_0		MQS_RIGHT		2:0		
12	B0_01	2.01			SPI(4) MISO	QT1_1		MQS_LEFT		2:1		
11	B0_02	2.02			SPI(4) MOSI	QT1_2	1_TX			2:2		
13	B0_03	2.03			SPI(4) SCK	QT2_0	1_RX			2:3		
6	B0_10	2.10				PWM2_A2, QT4_1		1:TX3_RX1		2:10		
9	B0_11	2.11				PWM2_B2, QT4_2		1:TX2_RX2		2:11		
32	B0_12	2.12						1:TX1_RX3	IO-10	2:12		
8	B1_00	2.16	Serial2(4) TX			PWM1_A3		1:RX_DATA	IO-14	2:16, 3:16		
7	B1_01	2.17	Serial2(4) RX			PWM1_B3		1:TX_DATA	IO-15	2:17, 3:17		
37	SD_B0_00	3.12		Wire1(3) SCL	SPI2(1) SCK	PWM1_A0			IO-04			CMD
36	SD_B0_01	3.13		Wire1(3) SDA	SPI2(1) CS0	PWM1_B0			IO-05			CLK
35	SD_B0_02	3.14	Serial5(8) CTS		SPI2(1) MOSI	PWM1_A1			IO-06			DATA0
34	SD_B0_03	3.15	Serial5(8) RTS		SPI2(1) MISO	PWM1_B1			IO-07			DATA1
39	SD_B0_04	3.16	Serial5(8) TX		SPI2(1) B_SSO_B	PWM1_A2			IO-08			DATA2
38	SD_B0_05	3.17	Serial5(8) RX		SPI2(1) B_DQS	PWM1_B2			IO-09			DATA3
28	EMC_32	3.18	Serial7(7) RX			PWM3_B1						
31	EMC_36	3.22				GPT1_2	3_TX	3:TX_DATA	IO-22			
30	EMC_37	3.23				GPT1_3	3_RX	3:MCLK	IO-23			
2	EMC_04	4.04				PWM4_A2		2:TX_DATA	IO-06	1:4		
3	EMC_05	4.05				PWM4_B2		2:TX_SYNC	IO-07	1:5		
4	EMC_06	4.06				PWM2_A0		2:TX_BCLK	IO-08	1:6		
33	EMC_07	4.07				PWM2_B0		2:MCLK	IO-09	1:7		
5	EMC_08	4.08				PWM2_A1		2:RX_DATA	IO-17	1:8		
29	EMC_31	4.31	Serial7(7) TX		SPI2(1) CS1	PWM3_A1						

Note: the GPIO column which this page is sorted on. where pin 1 is shown as GPIO 1.02. GPIO1 maps to GPIO6 (2->7, 3->8, 4->9) The lower numbers are when the pins are in the standard mode, and the higher numbers are when they are in high-speed mode.... We switch all of the pins to the higher numbers during sketch startup (startup.c)
And the .02 is this is bit 2 of the 32 bit register.

Source: https://forum.pjrc.com/threads/72849-Read-multiple-input-pins-simultaneously?p=326209#post326209

Here is a sort of quick and dirty sketch to print out mapping...

```
uint32_t *port_regsp[] = {(uint32_t*)&IMXRT_GPIO6, (uint32_t*)&IMXRT_GPIO7, (uint32_t*)&IMXRT_GPIO8, (uint32_t*)&IMXRT_GPIO9};
uint8_t pin_numbers[4][32];

void setup() {
  memset(pin_numbers, 0xff, sizeof(pin_numbers));
  // put your setup code here, to run once:
  for (uint8_t pin = 0; pin < CORE_NUM_TOTAL_PINS; pin++) {
    uint32_t *port = digital_pin_to_info_PGM[pin].reg;
    uint8_t port_pin = __builtin_ctz(digital_pin_to_info_PGM[pin].mask);
    for (uint8_t i = 0; i < 4; i++) {
      if (port == port_regsp[i]) {
        pin_numbers[i][port_pin] = pin;
        break;
      }
    }
  }

  while (!Serial) ; // wait for serial.

  Serial.println("\n       31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00");
  for (int i = 0; i < 4; i++) {
    Serial.printf("GPIO%d:", i + 6);
    for (int j = 31; j >= 0; j--) {
      if (pin_numbers[i][j] != 0xff) Serial.printf(" %02u", pin_numbers[i][j]);
      else Serial.print(" --");
    }
    Serial.println();
  }
}

void loop() {
  // put your main code here, to run repeatedly:
}
```

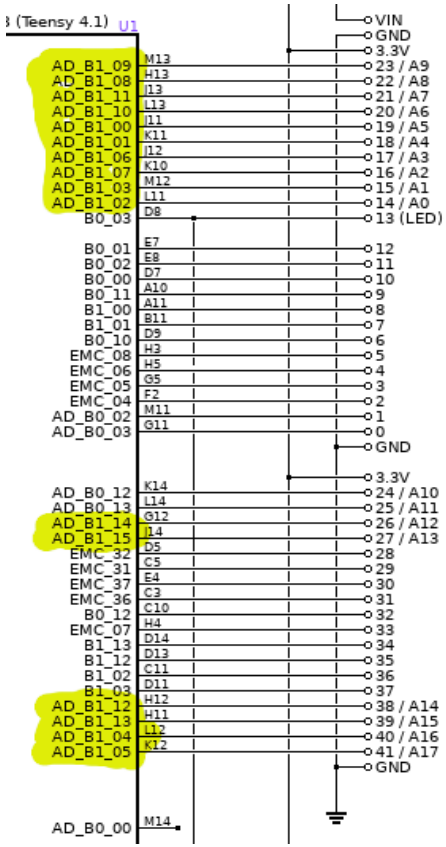
Could probably be cleaner, but here is output run on Micromod:

```
       31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 09 08 07 06 05 04 03 02 01 00
GPIO6: 27 26 -- -- 21 20 23 22 16 17 -- -- 15 14 18 19 -- -- 25 24 -- -- -- -- -- -- -- -- 00 01 -- --
GPIO7: -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- 07 08 -- -- -- 32 09 06 45 44 43 42 41 40 13 11 12 10
GPIO8: -- -- -- -- -- -- -- -- 30 31 -- -- -- 28 39 38 34 35 36 37 -- -- -- -- -- -- -- -- -- -- -- --
```

```
GPIO9: 29 -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- 05 33 04 03 02 -- -- -- --
```

On Teensy 4.1 you can read all 16 bits with a single GPIO register if you connect the data pins to all the "AD_B1_xx" GPIO pins.

Source: https://forum.pjrc.com/index.php?threads/spi-with-2-or-4-data-lines.74015/



2.14 Serial Monitor and Serial Plotter

There is no additional USB cable required for the serial monitor. The data goes through the same USB cable that's used for programming.

Use this code in setup():

```
Serial.begin(38400);
```

Note: This line is unnecessary and can be omitted if it's a virtual USB serial port. In this case data is always transmitted as fast as possible.

Write a variable x to the serial port:

```
Serial.println(x);
```

After initializing the USB serial port, you must wait until it's ready. Otherwise the first transmissions aren't visible in the Arduino IDE's serial monitor.

```
void setup()
{
  Serial.begin(38400);
  while (!Serial) ; // wait for serial port to connect. Needed for native USB
  Serial.println("hello");
}
```

Write a float variable x with 4 decimal places (the default is 2 decimal places):

```
Serial.println(x,4);    or    Serial.println(String(x,4));
```

Write a float variable x with 3 decimal places, and add a leading space if the number is not negative:

```
Serial.println((x<0 ? "" : " ") + String(x,3));
```

In the Arduino IDE, use Tools --> Serial Monitor

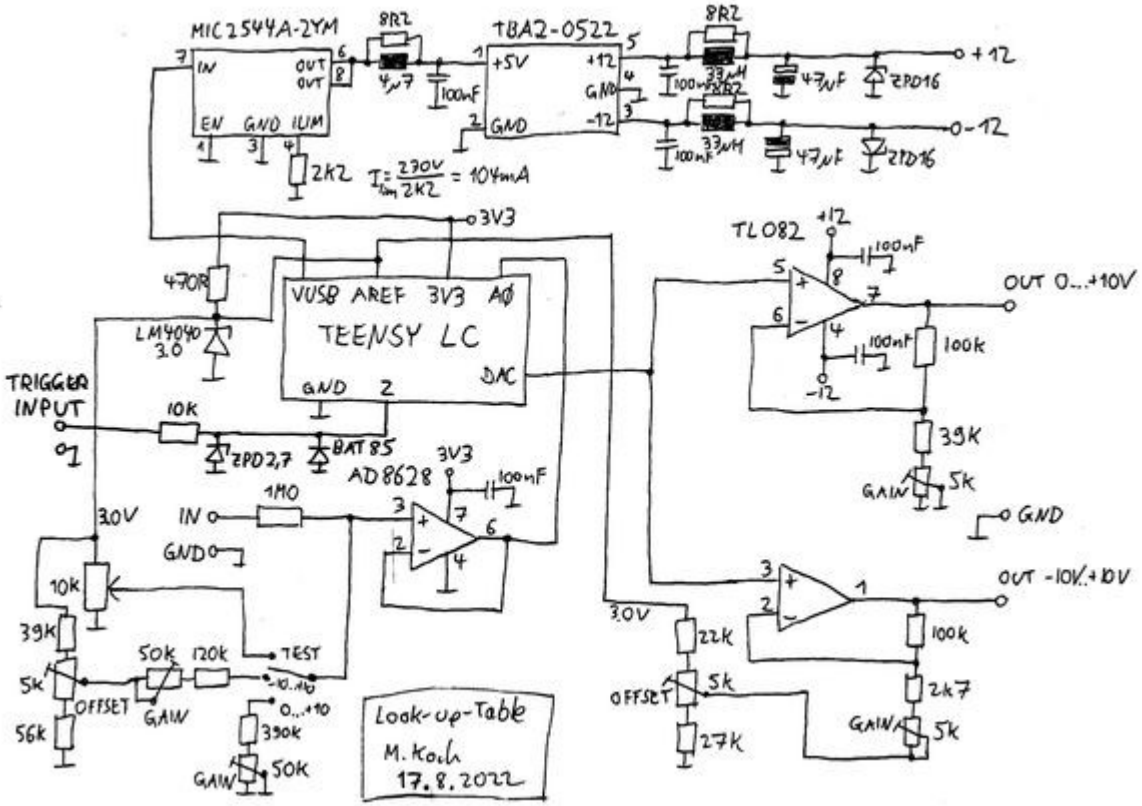
You will see the printout in the lower part of the screen.

You can also show the numbers graphically with Tools --> Serial Plotter

3 Projects

3.1 Teensy LC as an accessory for THAT analog computer

Schematic diagram: https://www.facebook.com/groups/theanalogthing/posts/400329198856542/



3.2 Summary of available software for THAT analog computer

3.2.1 Look-up-Table

This can be used for any function y=f(x) that can be expressed in C code.

http://www.astro-electronic.de/that-look-up-table2.ino

```
// Look-up-table for THAT, realized with Teensy LC
// Michael Koch 2022

boolean in_range = 0;     // 0 means the input range is [0 ... +1] machine units
                          // 1 means the input range is [-1 ... +1] machine units
boolean out_range = 0;    // 0 means the output range is [0 ... +1] machine units
                          // 1 means the output range is [-1 ... +1] machine units
int led = 13;
byte lowbyte[4096];
byte highbyte[2048];

// the setup routine runs once when you press reset:
void setup() {
  pinMode(led, OUTPUT);      // initialize the digital pin as an output

  for(int x=0; x<4096; x+=2)  // fill the look-up-table, two 12-bit entries are compressed into 3 bytes
  {
    int y0 = mu2int(function(int2mu(x)));
    int y1 = mu2int(function(int2mu(x+1)));
    lowbyte[x] = y0 & 0x00ff;
    lowbyte[x+1] = y1 & 0x00ff;
    highbyte[x>>1] = ((y0 & 0x0f00) >> 8) | ((y1 & 0x0f00) >> 4);
  }

  analogReference(EXTERNAL);   // 3.0V reference
  analogReadResolution(12);    // set to 12 bits
  analogWriteResolution(12);   // set to 12 bits

  //  Serial.begin(38400);
}

float function(float x)   // This is the user-defined function with input and output in machine units
{
```

```
  float y = x;
  if ((x>=0.35) && (x<0.45)) y = 0.0;
  if ((x>=0.45) && (x<0.55)) y = 0.5;
  if ((x>=0.55) && (x<0.65)) y = 1.0;
  return(y);
}

float int2mu(int i)       // convert from 12-bit integer to machine units
{
  if (in_range == 0)          // range is [0 ... +1] machine units
    return((float)i / 4096);
  else                        // range is [-1 ... +1] machine units
    return((float)(i - 2048) / 2048);
}

int mu2int(float f)       // convert from machine units to 12-bit integer, and clip to the allowed range
{
  if (out_range == 0)         // range is [0 ... +1] machine units
  {
    if (f < 0) f = 0;
    if (f > 0.9999) f = 0.9999;
    return((int)(f * 4096));
  }
  else                        // range is [-1 ... +1] machine units
  {
    if (f < -1) f = -1;
    if (f > 0.9999) f = 0.9999;
    return(2048 + (int)(f * 2048));
  }
}

void loop() {
  digitalWrite(led, HIGH);   // switch on the LED (for checking the sample rate, about 57kHz)
  int x = analogRead(0);
  digitalWrite(led, LOW);    // switch off the LED
  if(x & 0x0001)
    analogWrite(A12, lowbyte[x] + ((highbyte[x>>1] & 0xf0) << 4));
  else
    analogWrite(A12, lowbyte[x] + ((highbyte[x>>1] & 0x0f) << 8));
}
```

A problem appears when you try to use the exponential function exp() in the above example. Obviously it needs too much RAM. You can use the following quick-and-dirty power series approximation instead. This is not the most efficient way to approximate the exponential function.

```
float expo(float x)   // This is a power series approximation for the exponential function,
                      // because the built-in exponential function needs too much RAM
{
  float y = 1;
  float z = 1;
  float n = 1;
  for (int i = 1; i < 20; i++)
  {
    z = z * x;
    n = n * i;
    y = y + z / n;
  }
  return(y);
}
```

3.2.2 Look-Up-Table with Derivative

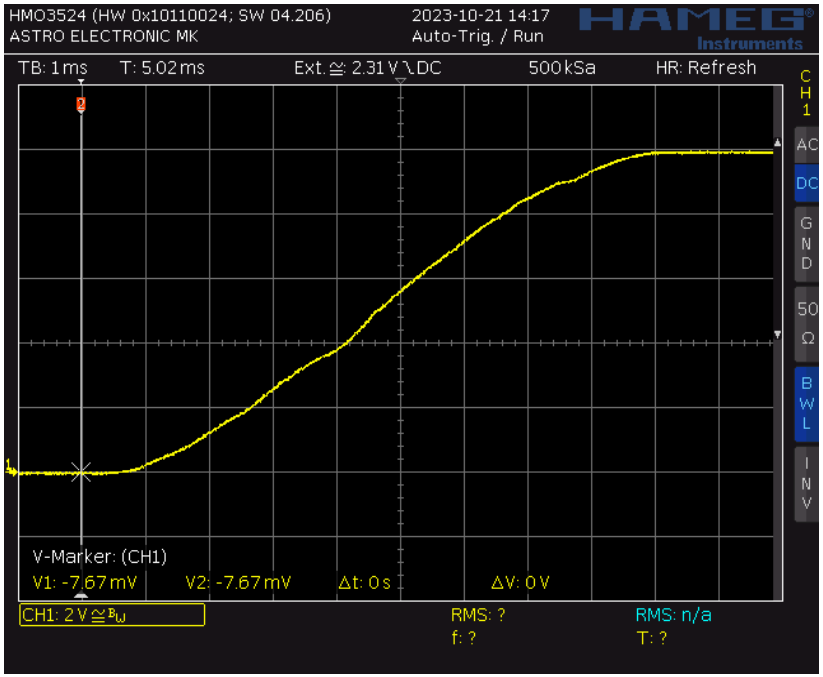
Here a user-defined table is used, which has less entries (351) than the final look-up-table (4096).

In this example the user-defined table contains the height profile of the Eichelnkopf mountain with 351 points, which are separated by 1 m horizontal distance each. The height difference from start to end is 53.2 m. The maximum slope is about 34-35%.

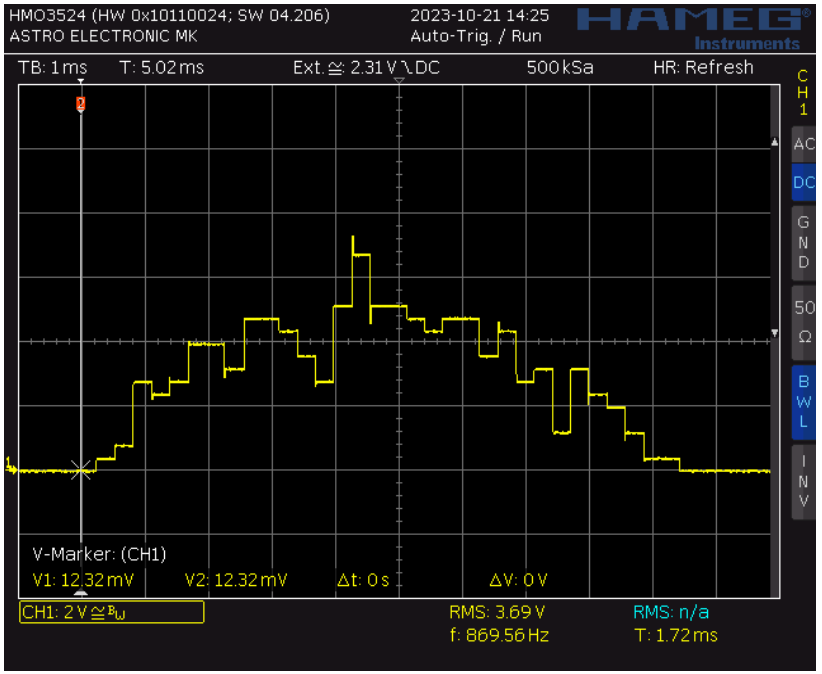
The output of the look-up-table can be either the height profile of its derivative (which is the slope).

http://www.astro-electronic.de/that_look_up_table_derivative.ino

This is the height profile of the Eichelnkopf mountain, 53.2m elevation is equivalent to 1 machine unit (10V):



This is the slope of the Eichelnkopf mountain, 50% slope is equivalant to 1 machine unit (10V):



```
// Look-up-table with derivative for THAT, realized with Teensy LC
// Michael Koch 2023

boolean in_range = 0;    // 0 means the input range is [0 ... +1] machine units
                         // 1 means the input range is [-1 ... +1] machine units
boolean out_range = 0;   // 0 means the output range is [0 ... +1] machine units
                         // 1 means the output range is [-1 ... +1] machine units
int led = 13;
byte lowbyte[4096];
byte highbyte[2048];


// This is the height profile of the Eichelnkopf mountain.
// It consists of 351 points which are separated by 1 m horizontal distance each.
// The height difference from start to end is 53.2 m.
// The maximum slope is about 34-35%.
//
// Input of look-up-table: 350m horizontal distance is equivalent to 1 machine unit.
// Output of look-up-table, in elevation mode: 53.2m or 100m elevation is equivalent to 1 machine unit.
// Output of look-up-table, in slope mode: 35% or 50% slope is equivalent to 1 machine unit.

const int number_of_points = 351;
const float curve[number_of_points] =
{
  0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0.02, 0.04, 0.06, 0.08, 0.1, 0.12, 0.14, 0.16, 0.18, 0.2, 0.24, 0.28, 0.32, 0.36,
  0.4, 0.44, 0.48, 0.52, 0.56, 0.6, 0.74, 0.88, 1.02, 1.16, 1.3, 1.44, 1.58, 1.72, 1.86, 2, 2.12, 2.24, 2.36, 2.48,
  2.6, 2.72, 2.84, 2.96, 3.08, 3.2, 3.34, 3.48, 3.62, 3.76, 3.9, 4.04, 4.18, 4.32, 4.46, 4.6, 4.8, 5, 5.2, 5.4, 5.6,
  5.8, 6, 6.2, 6.4, 6.6, 6.8, 7, 7.2, 7.4, 7.6, 7.8, 8, 8.2, 8.4, 8.6, 8.76, 8.92, 9.08, 9.24, 9.4, 9.56, 9.72, 9.88,
  10.04, 10.2, 10.44, 10.68, 10.92, 11.16, 11.4, 11.64, 11.88, 12.12, 12.36, 12.6, 12.84, 13.08, 13.32, 13.56, 13.8,
  14.04, 14.28, 14.52, 14.76, 15, 15.22, 15.44, 15.66, 15.88, 16.1, 16.32, 16.54, 16.76, 16.98, 17.2, 17.38, 17.56,
  17.74, 17.92, 18.1, 18.28, 18.46, 18.64, 18.82, 19, 19.14, 19.28, 19.42, 19.56, 19.7, 19.84, 19.98, 20.12, 20.26,
  20.4, 20.66, 20.92, 21.18, 21.44, 21.7, 21.96, 22.22, 22.48, 22.74, 23, 23.34, 23.68, 24.02, 24.36, 24.7, 25.04,
  25.38, 25.72, 26.06, 26.4, 26.66, 26.92, 27.18, 27.44, 27.7, 27.96, 28.22, 28.48, 28.74, 29, 29.26, 29.52, 29.78,
  30.04, 30.3, 30.56, 30.82, 31.08, 31.34, 31.6, 31.84, 32.08, 32.32, 32.56, 32.8, 33.04, 33.28, 33.52, 33.76, 34,
  34.22, 34.44, 34.66, 34.88, 35.1, 35.32, 35.54, 35.76, 35.98, 36.2, 36.44, 36.68, 36.92, 37.16, 37.4, 37.64, 37.88,
  38.12, 38.36, 38.6, 38.84, 39.08, 39.32, 39.56, 39.8, 40.04, 40.28, 40.52, 40.76, 41, 41.18, 41.36, 41.54, 41.72,
  41.9, 42.08, 42.26, 42.44, 42.62, 42.8, 43.02, 43.24, 43.46, 43.68, 43.9, 44.12, 44.34, 44.56, 44.78, 45, 45.14,
  45.28, 45.42, 45.56, 45.7, 45.84, 45.98, 46.12, 46.26, 46.4, 46.56, 46.72, 46.88, 47.04, 47.2, 47.36, 47.52, 47.68,
  47.84, 48, 48.06, 48.12, 48.18, 48.24, 48.3, 48.36, 48.42, 48.48, 48.54, 48.6, 48.76, 48.92, 49.08, 49.24, 49.4,
  49.56, 49.72, 49.88, 50.04, 50.2, 50.32, 50.44, 50.56, 50.68, 50.8, 50.92, 51.04, 51.16, 51.28, 51.4, 51.5, 51.6,
  51.7, 51.8, 51.9, 52, 52.1, 52.2, 52.3, 52.4, 52.46, 52.52, 52.58, 52.64, 52.7, 52.76, 52.82, 52.88, 52.94, 53,
  53.02, 53.04, 53.06, 53.08, 53.1, 53.12, 53.14, 53.16, 53.18, 53.2, 53.22, 53.24, 53.26, 53.28, 53.3, 53.32, 53.34,
  53.36, 53.38, 53.4, 53.38, 53.36, 53.34, 53.32, 53.3, 53.28, 53.26, 53.24, 53.22, 53.2, 53.2, 53.2, 53.2, 53.2,
```

```
  53.2, 53.2, 53.2, 53.2, 53.2, 53.2
};

// the setup routine runs once when you press reset:
void setup()
{
  pinMode(led, OUTPUT);         // initialize the digital pin as an output
  analogReference(EXTERNAL);    // 3.0V reference
  analogReadResolution(12);     // set to 12 bits
  analogWriteResolution(12);    // set to 12 bits

  for(int x=0; x<4096; x+=2)    // fill the look-up-table, two 12-bit entries are compressed into 3 bytes
  {
    int y0 = mu2int(function(int2mu(x)));
    int y1 = mu2int(function(int2mu(x+1)));
    lowbyte[x] = y0 & 0x00ff;
    lowbyte[x+1] = y1 & 0x00ff;
    highbyte[x>>1] = ((y0 & 0x0f00) >> 8) | ((y1 & 0x0f00) >> 4);
  }
}

float function(float x)   // This is the user-defined function with input and output in machine units
{
  int index = int(x * number_of_points);
  int index2 = index+1;
  if (index2 == number_of_points) index2--;
  //float y = curve[index] / 53.2;                    // use this line for elevation as output, 53.2m = 1 machine unit
  //float y = curve[index] / 100;                     // use this line for elevation as output, 100m = 1 machine unit
  //float y = (index2] - curve[index]) / 0.35;        // use this line for slope as output, 35% slope = 1 machine unit
  float y = (curve[index2] - curve[index]) / 0.5;   // use this line for slope as output, 50% slope = 1 machine unit
  return(y);
}

float int2mu(int i)       // convert from 12-bit integer to machine units
{
  if (in_range == 0)          // range is [0 ... +1] machine units
    return((float)i / 4096);
  else                        // range is [-1 ... +1] machine units
    return((float)(i - 2048) / 2048);
}

int mu2int(float f)       // convert from machine units to 12-bit integer, and clip to the allowed range
{
```

```
  if (out_range == 0)         // range is [0 ... +1] machine units
  {
    if (f < 0) f = 0;
    if (f > 0.9999) f = 0.9999;
    return((int)(f * 4096));
  }
  else                        // range is [-1 ... +1] machine units
  {
    if (f < -1) f = -1;
    if (f > 0.9999) f = 0.9999;
    return(2048 + (int)(f * 2048));
  }
}

void loop()
{
  digitalWrite(led, HIGH);   // switch on the LED (for checking the sample rate, about 57kHz)
  int x = analogRead(0);
  digitalWrite(led, LOW);    // switch off the LED
  if(x & 0x0001)
    analogWrite(A12, lowbyte[x] + ((highbyte[x>>1] & 0xf0) << 4));
  else
    analogWrite(A12, lowbyte[x] + ((highbyte[x>>1] & 0x0f) << 8));
}
```

3.2.3 Delay line

http://www.astro-electronic.de/that-delay-line.ino

```
// Delay line for THAT, realized with Teensy LC
// Michael Koch 2022

int led = 13;
short buffer[3072];
int buffer_length;
int pointer = 0;
IntervalTimer myTimer;

// the setup routine runs once when you press reset:
void setup()
{
  pinMode(led, OUTPUT);        // initialize the digital pin as an output
  analogReference(EXTERNAL);   // 3.0V reference
  analogReadResolution(12);    // set ADC to 12 bits
  analogWriteResolution(12);   // set DAC to 12 bits

  // delay_time = buffer_length * sample_interval
  buffer_length = 1000;                  // set the length of the buffer (must be <= 3072)
  myTimer.begin(timer_interrupt, 1000);  // set the sample interval in microseconds (must be >= 20)
  myTimer.priority(0);                   // high priority for timer interrupt
}

void timer_interrupt()
{
  digitalWrite(led, HIGH);             // switch on the LED (for checking the sample rate)
  analogWrite(A12, buffer[pointer]);   // write to DAC
  buffer[pointer] = analogRead(A0);    // read from ADC
  if (++pointer == buffer_length)      // increment pointer
    pointer = 0;
  digitalWrite(led, LOW);              // switch off the LED
}

void loop()   // nothing to do in the main loop
{
}
```

3.2.4 Direct digital synthesis

Create any periodic waveform with any floating-point frequency: http://www.astro-electronic.de/that-direct-digital-synthesis.ino

```
// Direct digital synthesis for THAT, realized with Teensy LC
// Michael Koch 2022

boolean out_range = 1;   // 0 means the output range is [0 ... +1] machine units
                         // 1 means the output range is [-1 ... +1] machine units
float freq = 440;        // Output frequency in Hz

int led = 13;
short waveform[2048];
unsigned int ph;
unsigned int dp;
IntervalTimer myTimer;

void setup()    // the setup routine runs once when you press reset
{
  pinMode(led, OUTPUT);        // initialize the digital pin as an output
  analogReference(EXTERNAL);   // 3.0V reference
  analogWriteResolution(12);   // set DAC to 12 bits

  float pi = 4 * atan(1.0);    // calculate pi
  for(int x=0; x<2048; x++)    // fill the waveform array
    waveform[x] = mu2int(function(x / 2048.0 * 2 * pi));

  dp = (unsigned int)(freq * 65536.0 * 65536.0 / 125000.0);  // calculate the value that is added to the 32-bit phase accumulator

  myTimer.begin(timer_interrupt, 8);  // set the sample interval to 8 microseconds (125kHz)
  myTimer.priority(0);                // high priority for timer interrupt
}

float function(float x)   // this function contains the user-defined waveform
{                         // x is in the [0..2*pi] range and the output is in machine units.
  float y = 0.7 * sin(x) + 0.2 * sin(3*x) + 0.1 * sin(5*x);   // sine with 3rd and 5th harmonic
  if (x < 0.05) y += 0.5;                                     // add a short impulse
  return(y);
}

int mu2int(float f)       // convert from machine units to 12-bit integer, and clip to the allowed range
{
  if (out_range == 0)         // range is [0 ... +1] machine units
  {
    if (f < 0) f = 0;
```

```
    if (f > 0.9999) f = 0.9999;
    return((int)(f * 4096));
  }
  else                        // range is [-1 ... +1] machine units
  {
    if (f < -1) f = -1;
    if (f > 0.9999) f = 0.9999;
    return(2048 + (int)(f * 2048));
  }
}

void timer_interrupt()
{
  digitalWrite(led, HIGH);               // switch on the LED (for checking the sample rate)
  ph += dp;                              // update the phase accumulator
  analogWrite(A12, waveform[ph >> 21]);  // use the most significant 11 bits as index for waveform array, and write to DAC
  digitalWrite(led, LOW);                // switch off the LED
}

void loop()   // nothing to do in the main loop
{
}
```

3.2.5 Voltage controlled direct digital synthesis

Same as before, but the frequency can be adjusted with the analog input voltage.

http://www.astro-electronic.de/that-vco-direct-digital-synthesis.ino

```
// VCO with direct digital synthesis for THAT, realized with Teensy LC
// Michael Koch 2022

boolean out_range = 1;   // 0 means the output range is [0 ... +1] machine units
                         // 1 means the output range is [-1 ... +1] machine units
boolean vco_enable = 1;  // 0 means the output frequency is constant
                         // 1 means the output frequency is proportional to the input voltage, from 0 to freq
float freq = 880.0;      // Output frequency in Hz

int led = 13;
short waveform[2048];
unsigned int ph = 0;
unsigned int dp = 0;
IntervalTimer myTimer1;
IntervalTimer myTimer2;

void setup()    // the setup routine runs once when you press reset
{
  pinMode(led, OUTPUT);         // initialize the digital pin as an output
  analogReference(EXTERNAL);    // 3.0V reference
  analogReadResolution(12);     // set ADC to 12 bits
  analogWriteResolution(12);    // set DAC to 12 bits

  float pi = 4 * atan(1.0);     // calculate pi
  for(int x=0; x<2048; x++)     // fill the waveform array
    waveform[x] = mu2int(function(x / 2048.0 * 2 * pi));

  if (vco_enable == 1)
    myTimer1.begin(update_frequency, 1000);  // set the frequency update interval to 1000 microseconds (1kHz)
  else
    dp = 34359.738 * freq;     // calculate the value that is added to the 32-bit phase accumulator
                               //  2^32 / 125000 = 34359.738

  myTimer2.begin(timer_interrupt, 8);  // set the sample interval to 8 microseconds (125kHz)
  myTimer2.priority(0);                // high priority for timer interrupt
}

float function(float x)   // this function contains the user-defined waveform
{                         // x is in the [0..2*pi] range and the output is in machine units.
```

```
  float y = 0.7 * sin(x) + 0.2 * sin(3*x) + 0.1 * sin(5*x);   // sine with 3rd and 5th harmonic
  return(y);
}

int mu2int(float f)       // convert from machine units to 12-bit integer, and clip to the allowed range
{
  if (out_range == 0)         // range is [0 ... +1] machine units
  {
    if (f < 0) f = 0;
    if (f > 0.9999) f = 0.9999;
    return((int)(f * 4096));
  }
  else                        // range is [-1 ... +1] machine units
  {
    if (f < -1) f = -1;
    if (f > 0.9999) f = 0.9999;
    return(2048 + (int)(f * 2048));
  }
}

void update_frequency()
{
  digitalWrite(led, HIGH);                 // switch on the LED
  dp = analogRead(A0) * 8.3906565 * freq;  // calculate the value that is added to the 32-bit phase accumulator
                                           //  2^32 / 125000 / 4095 = 8.3906565
  digitalWrite(led, LOW);                  // switch off the LED
}

void timer_interrupt()
{
  ph += dp;                               // update the phase accumulator
  analogWrite(A12, waveform[ph >> 21]);   // use the most significant 11 bits as index for waveform array, and write to DAC
}

void loop()   // nothing to do in the main loop
{
}
```

3.2.6 Triggered signal generator

Synchronized with THAT's trigger signal

http://www.astro-electronic.de/that-triggered-signal-generator.ino

```
// Triggered signal generator for THAT, realized with Teensy LC
// Michael Koch 2022

boolean out_range = 1;       // 0 means the output range is [0...+1] machine units
                             // 1 means the output range is [-1...+1] machine units

boolean mode = 0;            // 0 means one-shot operation
                             // 1 means loop operation

int sample_interval = 10;    // set the sample interval in microseconds, must be >= 10
                             // the total duration of the waveform is 3600 * sample_interval
                             // Examples:
                             // sample_interval = 10      duration = 36ms
                             // sample_interval = 100     duration = 360ms
                             // sample_interval = 1000    duration = 3.6s
                             // sample_interval = 10000   duration = 36s
int led = 13;
int trig = 2;
byte lowbyte[3600];
byte highbyte[1800];
IntervalTimer myTimer;
int x;

void setup()        // the setup routine runs once when you press reset
{
  pinMode(led, OUTPUT);        // initialize the digital pin as an output
  pinMode(trig, INPUT);        // initialize the digital pin as an input
  analogReference(EXTERNAL);   // 3.0V reference
  analogWriteResolution(12);   // set to 12 bits

  for(x=0; x<3600; x+=2)       // fill the array with the waveform, two entries are compressed into 3 bytes
  {
    float t0 = x * (float)sample_interval * 1e-6;       // time in seconds
    float t1 = (x+1) * (float)sample_interval * 1e-6;
    int y0 = mu2int(function(t0));
    int y1 = mu2int(function(t1));
    lowbyte[x] = y0 & 0x00ff;
    lowbyte[x+1] = y1 & 0x00ff;
    highbyte[x>>1] = ((y0 & 0x0f00) >> 8) | ((y1 & 0x0f00) >> 4);
```

```
  }

  myTimer.begin(timer_interrupt, sample_interval);  // start the timer interrupt
  myTimer.priority(0);                              // high priority for timer interrupt
}

float function(float t)   // This is the user-defined waveform with input t in seconds and output in machine units
{
  float y = exp(-50 * t) * sin(2 * 3.1415 * 440 * t);   // 440Hz sine with exponential damping
  return(y);
}

int mu2int(float f)       // convert from machine units to 12-bit integer, and clip to the allowed range
{
  if (out_range == 0)         // range is [0 ... +1] machine units
  {
    if (f < 0) f = 0;
    if (f > 0.9999) f = 0.9999;
    return((int)(f * 4096));
  }
  else                        // range is [-1 ... +1] machine units
  {
    if (f < -1) f = -1;
    if (f > 0.9999) f = 0.9999;
    return(2048 + (int)(f * 2048));
  }
}

void timer_interrupt()
{
  if (digitalRead(trig) == HIGH)    // THAT is in initial condition
  {
    digitalWrite(led, HIGH);        // switch on the LED
    x = 0;                          // in initial condition the first element of the waveform array is written to DAC
  }
  else                              // THAT is in operate mode
  {
    digitalWrite(led, LOW);         // switch off the LED
    x++;                            // increase the pointer
    if (x >= 3600)                  // is the end of the waveform array reached?
    {
      if (mode == 0)
        x = 3599;                   // in one-shot mode, the pointer stays at the last element
      else
        x = 0;                      // in loop mode, the pointer jumps to the first element
    }
  }
  if(x & 0x0001)                    // write value to DAC
```

```
    analogWrite(A12, lowbyte[x] + ((highbyte[x>>1] & 0xf0) << 4));
  else
    analogWrite(A12, lowbyte[x] + ((highbyte[x>>1] & 0x0f) << 8));
}

void loop()   // nothing to do in the main loop
{
}
```

3.2.7 Band-limited noise generator

http://www.astro-electronic.de/that-band_limited_noise.ino

```
// Band limited noise generator for THAT, realized with Teensy LC
// Michael Koch 2023

float samplerate = 10000;  // Sample rate in Hz, 1000000/samplerate should be an integer,
                           //  do not set higher than about 125000 Hz
float freq1 = 400;         // Lower band limit in Hz
float freq2 = 600;         // Upper band limit in Hz

const int length = 3000;          // length of waveform table,
                                  //  do not set higher than about 3000
float df = samplerate / length;   // distance of lines in spectrum in Hz
short waveform[length];           // waveform table
float pi = 4 * atan(1.0);         // calculate pi
int i = 0;                        // index counter
int led = 13;                     // LED output
IntervalTimer myTimer;

void setup()    // the setup routine runs once when you press reset
{
  pinMode(led, OUTPUT);        // initialize the digital pin as an output
  analogReference(EXTERNAL);   // 3.0V reference
  analogWriteResolution(12);   // set DAC to 12 bits

  for(int x = 0; x < length; x++)    // fill the waveform array
    waveform[x] = 2048 + (int)(2048 * function(x));

  myTimer.begin(timer_interrupt, (int)(1000000 / samplerate));  // set the interrupt interval
  myTimer.priority(0);                  // high priority for timer interrupt
}

float function(int x)   // calculate band-limited sum of sine waves
{
  float y = 0;
  float ph;
  randomSeed(13);       // use always the same seed for reproducible random
  for(float f = freq1; f <= freq2; f += df)
  {
    ph = 0.001 * (float)random(6283);                    // random phase [0...2*pi]
    y += sin(ph + (float)x * 2 * pi * f / samplerate);   // sum of many sine waves
  }
  y = y * 3.0 * df / (freq2 - freq1);                    // normalize to avoid clipping
```

```
  return(y);
}

void timer_interrupt()
{
  digitalWrite(led, HIGH);          // switch on the LED (only for checking the sample rate)
  if (++i >= length) i = 0;         // increment index counter
  analogWrite(A12, waveform[i]);    // read from the waveform array and write to DAC
  digitalWrite(led, LOW);           // switch off the LED
}

void loop()   // nothing to do in the main loop
{
}
```

3.2.8 Band-limited noise generator with a control voltage input

Control voltage from synthesizer (1V / octave, Arturia MatrixBrute):

http://www.astro-electronic.de/that-noise-with-cv.ino

```
// Noise generator with control voltage input, realized with Teensy LC
// Michael Koch 2023

float samplerate = 10000;         // Sample rate in Hz (for 1kHz)
const int length = 2500;          // length of waveform table,
                                  //  do not set higher than about 2800
float df = samplerate / length;   // distance of lines in spectrum in Hz
float freq1 = 900;         // Lower band limit in Hz, center frequency should be 1kHz
                           //  freq1 should be a multiple of df
float freq2 = 1110;        // Upper band limit in Hz, center frequency should be 1kHz


short waveform[length];           // waveform table
float pi = 4 * atan(1.0);         // calculate pi
int i = 0;                        // index counter
int led = 13;                     // LED output
IntervalTimer myTimer1, myTimer2;

void setup()    // the setup routine runs once when you press reset
{
  pinMode(led, OUTPUT);        // initialize the digital pin as an output
  analogReference(EXTERNAL);   // 3.0V reference
  analogWriteResolution(12);   // set DAC to 12 bits
  analogReadResolution(12);    // set ADC to 12 bits

  for(int x = 0; x < length; x++)    // fill the waveform array
  {
    waveform[x] = 2048 + (int)(2048 * function(x));
    if (x % 32 < 16)
      digitalWrite(led, LOW);     // switch off the LED as a progress indicator
    else
      digitalWrite(led, HIGH);    // switch on the LED as a progress indicator
  }

  myTimer1.begin(update_frequency, 10000);  // set the frequency update interval to 10 milliseconds
  myTimer1.priority(20);                    // lower priority for timer interrupt

  myTimer2.begin(timer_interrupt, (int)(1000000 / samplerate));  // set the interrupt interval
  myTimer2.priority(0);                  // high priority for timer interrupt
```

```
}

float function(int x)   // calculate band-limited sum of sine waves
{
  float y = 0;
  float ph;
  randomSeed(13);       // use always the same seed for reproducible random
  for(float f = freq1; f <= freq2; f += df)
  {
    ph = 0.001 * (float)random(6283);                    // random phase [0...2*pi]
    y += sin(ph + (float)x * 2 * pi * f / samplerate);   // sum of many sine waves
  }
  y = y * 2.5 * df / (freq2 - freq1);                    // normalize to avoid clipping
  return(y);
}

void update_frequency()
{
  float cv = 10.0 / 4096.0 * analogRead(A0);    // read the control voltage [0V...10V]
  float freq = 440 * pow(2, cv - 4.75);         // frequency = 440Hz * 2^(CV â^’ 4.75V)
                                                //  for Arturia Matrixbrute
  if (freq > 12500) freq = 12500;               // allowed frequency range is 1Hz to 12.5kHz
  if (freq < 1) freq = 1;
  myTimer2.update((int)(100000.0 / freq));
}

void timer_interrupt()
{
  digitalWrite(led, HIGH);          // switch on the LED (only for checking the sample rate)
  if (++i >= length) i = 0;         // increment index counter
  analogWrite(A12, waveform[i]);    // read from the waveform array and write to DAC
  digitalWrite(led, LOW);           // switch off the LED
}

void loop()   // nothing to do in the main loop
{
}
```

3.2.9 Heart Sound Synthesis for THAT with a control voltage input

```
// heart sound synthesis for THAT, realized with Teensy LC
// Michael Koch 2024

float freq = 240.0;        // heart beats per minute

int led = 13;
static const unsigned short waveform[] =
{
 0X7FF, 0X7FE, 0X7FD, 0X7FF, 0X801, 0X801, 0X7FF, 0X7FD, 0X7FA, 0X7F9, 0X7FA, 0X7FC, 0X800, 0X803, 0X804, 0X805,
 0X806, 0X805, 0X803, 0X7FF, 0X7FB, 0X7F8, 0X7F5, 0X7F1, 0X7F0, 0X7EF, 0X7EF, 0X7EE, 0X7EC, 0X7E8, 0X7E6, 0X7E3,
 0X7E2, 0X7E0, 0X7D9, 0X7D3, 0X7D1, 0X7D2, 0X7D4, 0X7D8, 0X7E0, 0X7EB, 0X7FB, 0X80C, 0X81D, 0X828, 0X82F, 0X833,
 0X835, 0X834, 0X831, 0X82C, 0X828, 0X824, 0X822, 0X822, 0X823, 0X825, 0X826, 0X826, 0X825, 0X824, 0X823, 0X823,
 0X820, 0X81C, 0X817, 0X813, 0X813, 0X814, 0X816, 0X817, 0X815, 0X812, 0X80E, 0X80E, 0X812, 0X819, 0X821, 0X827,
 0X82C, 0X82E, 0X82D, 0X828, 0X821, 0X817, 0X807, 0X7F4, 0X7DF, 0X7CB, 0X7B8, 0X7A9, 0X79D, 0X796, 0X793, 0X793,
 0X793, 0X792, 0X791, 0X78F, 0X790, 0X793, 0X799, 0X7A2, 0X7AB, 0X7B4, 0X7BB, 0X7BE, 0X7C2, 0X7C7, 0X7CD, 0X7D2,
 0X7D5, 0X7D6, 0X7D5, 0X7CF, 0X7C6, 0X7BB, 0X7AD, 0X7A0, 0X797, 0X794, 0X797, 0X7A0, 0X7AA, 0X7B1, 0X7B8, 0X7C1,
 0X7CF, 0X7E3, 0X7FE, 0X81D, 0X83E, 0X860, 0X881, 0X8A2, 0X8C7, 0X8EC, 0X90D, 0X928, 0X93F, 0X955, 0X96E, 0X98D,
 0X9A8, 0X9B3, 0X9A7, 0X98E, 0X97B, 0X97A, 0X98B, 0X99F, 0X99D, 0X984, 0X964, 0X94A, 0X928, 0X8D4, 0X821, 0X713,
 0X5E3, 0X4D7, 0X420, 0X3D6, 0X3E4, 0X40A, 0X402, 0X3BA, 0X367, 0X351, 0X39D, 0X442, 0X51A, 0X5F2, 0X6A5, 0X736,
 0X7CA, 0X871, 0X924, 0X9D1, 0XA67, 0XADE, 0XB38, 0XB7D, 0XBB6, 0XBE3, 0XC00, 0XC06, 0XBF3, 0XBC6, 0XB81, 0XB22,
 0XAAA, 0XA1C, 0X985, 0X8F5, 0X87B, 0X826, 0X802, 0X809, 0X82F, 0X868, 0X8A7, 0X8DC, 0X8FD, 0X904, 0X8F6, 0X8D9,
 0X8AF, 0X879, 0X837, 0X7EC, 0X79F, 0X759, 0X726, 0X70C, 0X70A, 0X71F, 0X73D, 0X757, 0X765, 0X765, 0X756, 0X72E,
 0X6EC, 0X697, 0X63E, 0X5F3, 0X5B4, 0X56B, 0X515, 0X4BF, 0X478, 0X451, 0X44A, 0X457, 0X462, 0X453, 0X435, 0X427,
 0X427, 0X414, 0X3F1, 0X3D9, 0X3D2, 0X3F4, 0X448, 0X4B9, 0X557, 0X644, 0X767, 0X897, 0X9C2, 0XADF, 0XBEE, 0XCFF,
 0XE12, 0XF0B, 0XFB5, 0XFEF, 0XFC6, 0XF57, 0XEC1, 0XE1A, 0XD66, 0XCA8, 0XBE2, 0XB19, 0XA5A, 0X9B2, 0X923, 0X8A8,
 0X83A, 0X7D7, 0X77F, 0X736, 0X702, 0X6E3, 0X6D5, 0X6D4, 0X6DB, 0X6EC, 0X702, 0X71B, 0X737, 0X751, 0X765, 0X774,
 0X781, 0X78E, 0X79B, 0X7A6, 0X7B1, 0X7BA, 0X7BE, 0X7BA, 0X7B1, 0X7A6, 0X7A0, 0X79E, 0X79E, 0X796, 0X786, 0X76B,
```

```
0X74F, 0X738, 0X725, 0X716, 0X706, 0X6F1, 0X6DC, 0X6CB, 0X6C0, 0X6BC, 0X6C0, 0X6C7, 0X6D1, 0X6DD, 0X6EB, 0X6FA,
0X70A, 0X719, 0X726, 0X733, 0X742, 0X751, 0X761, 0X770, 0X77F, 0X78C, 0X798, 0X7A5, 0X7B1, 0X7BE, 0X7CD, 0X7DE,
0X7F2, 0X808, 0X81F, 0X836, 0X849, 0X857, 0X861, 0X86A, 0X873, 0X87C, 0X886, 0X88D, 0X893, 0X898, 0X89C, 0X8A2,
0X8AA, 0X8B6, 0X8C4, 0X8D2, 0X8E0, 0X8EB, 0X8F3, 0X8F7, 0X8F6, 0X8F0, 0X8E7, 0X8DE, 0X8D8, 0X8D4, 0X8CE, 0X8C6,
0X8B8, 0X8A3, 0X888, 0X86B, 0X84F, 0X839, 0X82A, 0X81D, 0X810, 0X802, 0X7F1, 0X7E2, 0X7D6, 0X7CE, 0X7C8, 0X7C3,
0X7BE, 0X7BA, 0X7B9, 0X7BA, 0X7BE, 0X7C2, 0X7C4, 0X7C5, 0X7C5, 0X7C5, 0X7C4, 0X7C1, 0X7BA, 0X7B2, 0X7A9, 0X7A2,
0X79F, 0X7A2, 0X7A9, 0X7B3, 0X7BF, 0X7CC, 0X7D8, 0X7E3, 0X7ED, 0X7F4, 0X7FA, 0X7FE, 0X801, 0X802, 0X802, 0X7FC,
0X7F5, 0X7EC, 0X7E2, 0X7D9, 0X7D2, 0X7CD, 0X7C9, 0X7C5, 0X7C2, 0X7BF, 0X7BD, 0X7BB, 0X7BB, 0X7BE, 0X7C5, 0X7CC,
0X7D1, 0X7D3, 0X7D6, 0X7D9, 0X7E0, 0X7E4, 0X7E6, 0X7E4, 0X7E3, 0X7E8, 0X7F5, 0X805, 0X80F, 0X810, 0X80A, 0X802,
0X7FC, 0X7FB, 0X7FB, 0X7FB, 0X7FB, 0X7FB, 0X800, 0X80A, 0X817, 0X822, 0X826, 0X823, 0X81D, 0X81A, 0X820, 0X82E,
0X83D, 0X846, 0X845, 0X83E, 0X833, 0X82C, 0X82A, 0X82D, 0X830, 0X831, 0X82C, 0X824, 0X81B, 0X812, 0X80B, 0X807,
0X802, 0X7FD, 0X7F8, 0X7F4, 0X7F0, 0X7ED, 0X7EB, 0X7EA, 0X7EA, 0X7E9, 0X7E9, 0X7E8, 0X7E7, 0X7E7, 0X7EA, 0X7F0,
0X7F4, 0X7F6, 0X7F7, 0X7F6, 0X7F6, 0X7FA, 0X7FD, 0X800, 0X7FF, 0X7FB, 0X7F3, 0X7ED, 0X7EA, 0X7EC, 0X7EE, 0X7EF,
0X7EE, 0X7ED, 0X7EF, 0X7F3, 0X7FA, 0X801, 0X806, 0X80A, 0X80D, 0X80E, 0X810, 0X814, 0X817, 0X819, 0X819, 0X818,
0X817, 0X817, 0X817, 0X816, 0X814, 0X813, 0X811, 0X811, 0X813, 0X813, 0X814, 0X814, 0X815, 0X815, 0X815, 0X815,
0X813, 0X80F, 0X809, 0X803, 0X7FD, 0X7F8, 0X7F5, 0X7F4, 0X7F3, 0X7F3, 0X7F6, 0X7F9, 0X7FF, 0X806, 0X80D, 0X814,
0X819, 0X81C, 0X81E, 0X81E, 0X81D, 0X81B, 0X818, 0X813, 0X80E, 0X807, 0X801, 0X7FB, 0X7F5, 0X7F1, 0X7EE, 0X7EF,
0X7F0, 0X7F1, 0X7F0, 0X7ED, 0X7EB, 0X7E9, 0X7E8, 0X7E8, 0X7E6, 0X7E2, 0X7DE, 0X7DC, 0X7DC, 0X7DF, 0X7E3, 0X7E9,
0X7ED, 0X7EF, 0X7F1, 0X7F2, 0X7F5, 0X7F8, 0X7FB, 0X7FD, 0X7FC, 0X7FA, 0X7F8, 0X7F5, 0X7F3, 0X7F0, 0X7ED, 0X7EB,
0X7EA, 0X7EA, 0X7EC, 0X7EE, 0X7EF, 0X7F0, 0X7F1, 0X7F1, 0X7F2, 0X7F4, 0X7F5, 0X7F7, 0X7F9, 0X7FB, 0X7FF, 0X802,
0X803, 0X805, 0X805, 0X805, 0X805, 0X803, 0X802, 0X800, 0X7FD, 0X7FD, 0X7FE, 0X7FF, 0X801, 0X802, 0X802, 0X801,
0X802, 0X804, 0X808, 0X80D, 0X810, 0X811, 0X812, 0X813, 0X815, 0X817, 0X818, 0X819, 0X818, 0X817, 0X817, 0X816,
0X812, 0X80E, 0X80C, 0X80D, 0X80F, 0X812, 0X813, 0X814, 0X814, 0X814, 0X812, 0X80F, 0X80B, 0X805, 0X802, 0X7FF,
0X7FC, 0X7FC, 0X7FC, 0X7FC, 0X7FC, 0X7FC, 0X7FC, 0X7FB, 0X7FB, 0X7FB, 0X7FC, 0X7FF, 0X7FF, 0X7FF, 0X800, 0X800,
0X800, 0X800, 0X801, 0X802, 0X802, 0X800, 0X7FE, 0X7FD, 0X7FD, 0X800, 0X803, 0X805, 0X805, 0X803, 0X800, 0X7FD,
0X7FB, 0X7FB, 0X7F9, 0X7F7, 0X7F3, 0X7F1, 0X7EF, 0X7ED, 0X7EE, 0X7EF, 0X7F0, 0X7F1, 0X7F1, 0X7F3, 0X7F4, 0X7F6,
0X7F8, 0X7FA, 0X7FC, 0X7FD, 0X7FF, 0X7FE, 0X7FD, 0X7FC, 0X7FD, 0X7FF, 0X802, 0X806, 0X809, 0X80D, 0X80F, 0X811,
0X811, 0X811, 0X811, 0X810, 0X80F, 0X80D, 0X809, 0X807, 0X804, 0X801, 0X7FF, 0X7FC, 0X7F9, 0X7F5, 0X7F0, 0X7EB,
0X7E8, 0X7E7, 0X7E7, 0X7E8, 0X7E8, 0X7E9, 0X7E9, 0X7E9, 0X7E9, 0X7EB, 0X7ED, 0X7EF, 0X7F3, 0X7F8, 0X7FC, 0X801,
0X804, 0X806, 0X807, 0X807, 0X808, 0X80A, 0X80D, 0X810, 0X811, 0X811, 0X80F, 0X80C, 0X80A, 0X809, 0X808, 0X806,
0X805, 0X804, 0X803, 0X802, 0X7FF, 0X7FD, 0X7FD, 0X7FE, 0X7FF, 0X7FE, 0X7FC, 0X7FA, 0X7F9, 0X7F9, 0X7FB, 0X7FD,
```

```
0X7FE, 0X7FE, 0X7FE, 0X7FD, 0X7FC, 0X7FC, 0X7FD, 0X7FF, 0X800, 0X801, 0X802, 0X803, 0X806, 0X808, 0X80A, 0X80A,
0X808, 0X806, 0X807, 0X80A, 0X80D, 0X80F, 0X810, 0X80D, 0X809, 0X805, 0X804, 0X803, 0X803, 0X803, 0X802, 0X801,
0X7FF, 0X7FD, 0X7FB, 0X7FB, 0X7FC, 0X7FE, 0X800, 0X801, 0X800, 0X7FF, 0X7FD, 0X7FB, 0X7FC, 0X7FE, 0X801, 0X802,
0X802, 0X801, 0X7FE, 0X7FB, 0X7F9, 0X7F7, 0X7F5, 0X7F5, 0X7F5, 0X7F4, 0X7F5, 0X7F4, 0X7F4, 0X7F4, 0X7F4, 0X7F5,
0X7F4, 0X7F5, 0X7F4, 0X7F2, 0X7EF, 0X7ED, 0X7EF, 0X7F5, 0X7FE, 0X805, 0X809, 0X808, 0X804, 0X803, 0X805, 0X809,
0X80E, 0X813, 0X814, 0X812, 0X810, 0X810, 0X810, 0X811, 0X813, 0X813, 0X813, 0X813, 0X813, 0X812, 0X812, 0X812,
0X813, 0X811, 0X80E, 0X80C, 0X80C, 0X80B, 0X80B, 0X808, 0X805, 0X802, 0X800, 0X801, 0X805, 0X808, 0X809, 0X80A,
0X80A, 0X809, 0X80A, 0X80D, 0X811, 0X816, 0X81F, 0X82F, 0X845, 0X85D, 0X87B, 0X891, 0X887, 0X857, 0X818, 0X7D3,
0X765, 0X6C3, 0X616, 0X58D, 0X555, 0X58F, 0X627, 0X6CB, 0X726, 0X728, 0X721, 0X772, 0X83B, 0X936, 0X9EF, 0XA0E,
0X9A1, 0X916, 0X8E2, 0X91F, 0X985, 0X9B5, 0X983, 0X90C, 0X898, 0X864, 0X87B, 0X8B2, 0X8CE, 0X8B5, 0X87E, 0X852,
0X844, 0X84F, 0X85B, 0X850, 0X82B, 0X7FC, 0X7D6, 0X7C7, 0X7CB, 0X7D0, 0X7C8, 0X7AE, 0X786, 0X75A, 0X72F, 0X708,
0X6E6, 0X6CA, 0X6AE, 0X694, 0X682, 0X67A, 0X67D, 0X68C, 0X6A2, 0X6B7, 0X6C9, 0X6DA, 0X6F4, 0X71A, 0X74B, 0X784,
0X7C1, 0X7FA, 0X82E, 0X85E, 0X891, 0X8CB, 0X908, 0X93F, 0X970, 0X999, 0X9B8, 0X9C7, 0X9CA, 0X9C7, 0X9CC, 0X9DB,
0X9E6, 0X9E0, 0X9C0, 0X98C, 0X94F, 0X917, 0X8EB, 0X8C7, 0X8A6, 0X881, 0X85C, 0X83A, 0X81D, 0X7FC, 0X7D0, 0X798,
0X75A, 0X722, 0X6FA, 0X6E0, 0X6CE, 0X6BC, 0X6A7, 0X694, 0X687, 0X683, 0X685, 0X689, 0X68F, 0X698, 0X6A5, 0X6B7,
0X6D0, 0X6ED, 0X70C, 0X72C, 0X74E, 0X770, 0X791, 0X7B2, 0X7D1, 0X7F0, 0X80F, 0X82C, 0X848, 0X866, 0X882, 0X89D,
0X8B5, 0X8C7, 0X8D4, 0X8DC, 0X8DD, 0X8DC, 0X8D9, 0X8D4, 0X8CC, 0X8C2, 0X8B6, 0X8A8, 0X897, 0X884, 0X870, 0X85B,
0X844, 0X82D, 0X815, 0X7FE, 0X7E8, 0X7D4, 0X7C3, 0X7B5, 0X7A9, 0X7A1, 0X79C, 0X799, 0X798, 0X79A, 0X79D, 0X7A2,
0X7A9, 0X7B0, 0X7B8, 0X7C2, 0X7CD, 0X7D9, 0X7E4, 0X7EF, 0X7F9, 0X803, 0X80C, 0X813, 0X81A, 0X820, 0X826, 0X82B,
0X830, 0X832, 0X833, 0X834, 0X834, 0X834, 0X836, 0X837, 0X837, 0X836, 0X833, 0X830, 0X82E, 0X82C, 0X82A, 0X828,
0X825, 0X823, 0X821, 0X81E, 0X81C, 0X81A, 0X818, 0X815, 0X812, 0X810, 0X80F, 0X80D, 0X80B, 0X80A, 0X80B, 0X80D,
0X80D, 0X80C, 0X809, 0X808, 0X805, 0X804, 0X801, 0X803, 0X800, 0X7F5, 0X7EC, 0X7EC, 0X7F2, 0X7F6, 0X7F8, 0X7F4,
0X7E8, 0X7DB, 0X7D3, 0X7D1, 0X7D4, 0X7D2, 0X7CE, 0X7CB, 0X7C2, 0X7BB, 0X7BD, 0X7C4, 0X7C8, 0X7CB, 0X7CC, 0X7CD,
0X7CD, 0X7CF, 0X7D3, 0X7D7, 0X7DA, 0X7DD, 0X7E2, 0X7E6, 0X7E9, 0X7EB, 0X7EC, 0X7EF, 0X7F3, 0X7FA, 0X801, 0X807,
0X80A, 0X80A, 0X80A, 0X80C, 0X810, 0X815, 0X819, 0X81C, 0X81D, 0X81B, 0X819, 0X817, 0X816, 0X816, 0X817, 0X816,
0X815, 0X814, 0X812, 0X810, 0X80F, 0X80E, 0X80F, 0X80F, 0X80F, 0X80F, 0X80F, 0X80D, 0X80B, 0X807, 0X803, 0X800,
0X7FF, 0X7FF, 0X7FE, 0X7FD, 0X7FB, 0X7FA, 0X7FA, 0X7FA, 0X7FA, 0X7F8, 0X7F7, 0X7F6, 0X7F4, 0X7F4, 0X7F6, 0X7F9,
0X7FB, 0X7FE, 0X802, 0X805, 0X804, 0X802, 0X802, 0X805, 0X80D, 0X814, 0X818, 0X819, 0X816, 0X811, 0X80E, 0X80D,
0X80E, 0X80F, 0X80F, 0X80E, 0X80C, 0X80B, 0X80B, 0X80C, 0X80D, 0X80D, 0X80C, 0X80B, 0X80C, 0X80F, 0X811, 0X812,
0X814, 0X815, 0X815, 0X813, 0X811, 0X810, 0X80E, 0X80B, 0X808, 0X807, 0X804, 0X800, 0X7FD, 0X7FC, 0X7FC, 0X7FA,
0X7F7, 0X7F3, 0X7F0, 0X7ED, 0X7EB, 0X7EA, 0X7E7, 0X7E5, 0X7E2, 0X7E0, 0X7DE, 0X7DA, 0X7D6, 0X7D3, 0X7D0, 0X7D0,
```

```
0X7D1, 0X7D3, 0X7D7, 0X7DA, 0X7DC, 0X7DF, 0X7E2, 0X7E6, 0X7EB, 0X7F0, 0X7F6, 0X7FA, 0X7FF, 0X802, 0X805, 0X808,
0X80B, 0X810, 0X818, 0X821, 0X82B, 0X833, 0X838, 0X83A, 0X838, 0X836, 0X836, 0X838, 0X838, 0X837, 0X833, 0X82E,
0X828, 0X823, 0X81E, 0X819, 0X814, 0X80F, 0X80B, 0X807, 0X803, 0X7FF, 0X7FC, 0X7F9, 0X7F6, 0X7F3, 0X7F0, 0X7ED,
0X7EB, 0X7EA, 0X7EA, 0X7EA, 0X7EB, 0X7E9, 0X7E6, 0X7E4, 0X7E2, 0X7E1, 0X7E3, 0X7E5, 0X7E7, 0X7E9, 0X7E9, 0X7E7,
0X7E4, 0X7E0, 0X7E0, 0X7E3, 0X7E7, 0X7EB, 0X7EB, 0X7E8, 0X7E4, 0X7E1, 0X7E0, 0X7E4, 0X7E8, 0X7EC, 0X7EE, 0X7EF,
0X7EE, 0X7ED, 0X7EC, 0X7ED, 0X7EE, 0X7F0, 0X7F2, 0X7F4, 0X7F6, 0X7F8, 0X7FB, 0X7FD, 0X800, 0X802, 0X804, 0X807,
0X80B, 0X811, 0X815, 0X819, 0X81A, 0X819, 0X819, 0X81B, 0X81D, 0X81E, 0X81C, 0X819, 0X816, 0X814, 0X813, 0X812,
0X810, 0X80F, 0X80D, 0X80C, 0X80C, 0X80D, 0X80B, 0X807, 0X7FF, 0X7F8, 0X7F2, 0X7F0, 0X7F1, 0X7F3, 0X7F3, 0X7F1,
0X7EF, 0X7ED, 0X7ED, 0X7EF, 0X7F5, 0X7FD, 0X803, 0X80A, 0X80E, 0X812, 0X815, 0X817, 0X817, 0X817, 0X815, 0X813,
0X811, 0X811, 0X813, 0X815, 0X814, 0X812, 0X80F, 0X80E, 0X811, 0X814, 0X817, 0X817, 0X812, 0X80C, 0X807, 0X806,
0X806, 0X807, 0X80A, 0X80D, 0X80E, 0X80A, 0X804, 0X7FD, 0X7FA, 0X7FC, 0X800, 0X803, 0X803, 0X800, 0X7FA, 0X7F7,
0X7F6, 0X7F6, 0X7F6, 0X7F5, 0X7F4, 0X7F4, 0X7F5, 0X7F7, 0X7F9, 0X7F8, 0X7F6, 0X7F3, 0X7F2, 0X7F2, 0X7F2, 0X7F3,
0X7F3, 0X7F5, 0X7F9, 0X7FD, 0X7FD, 0X7F7, 0X7EC, 0X7E1, 0X7DC, 0X7DF, 0X7E7, 0X7F0, 0X7F8, 0X7FA, 0X7F7, 0X7F4,
0X7F1, 0X7F3, 0X7F6, 0X7F8, 0X7F7, 0X7F6, 0X7F3, 0X7F1, 0X7F3, 0X7F7, 0X7FB, 0X7FD, 0X7FC, 0X7F9, 0X7F6, 0X7F5,
0X7F5, 0X7F8, 0X7FB, 0X800, 0X804, 0X808, 0X80B, 0X80F, 0X810, 0X811, 0X813, 0X813, 0X813, 0X812, 0X812, 0X811,
0X810, 0X810, 0X80F, 0X80E, 0X80B, 0X808, 0X804, 0X801, 0X7FF, 0X7FD, 0X7FD, 0X7FC, 0X7FC, 0X7FB, 0X7FA, 0X7F9,
0X7F9, 0X7F8, 0X7F8, 0X7F9, 0X7F9, 0X7F8, 0X7F7, 0X7F6, 0X7F8, 0X7FA, 0X7FD, 0X7FE, 0X7FD, 0X7FB, 0X7FA, 0X7FA,
0X7FC, 0X7FF, 0X803, 0X806, 0X807, 0X805, 0X803, 0X802, 0X802, 0X801, 0X800, 0X7FF, 0X801, 0X803, 0X806, 0X807,
0X807, 0X804, 0X802, 0X7FF, 0X7FC, 0X7FA, 0X7FA, 0X7FA, 0X7FB, 0X7FD, 0X7FE, 0X7FF, 0X801, 0X803, 0X804, 0X804,
0X803, 0X803, 0X803, 0X804, 0X806, 0X809, 0X80C, 0X80E, 0X80F, 0X810, 0X80F, 0X80F, 0X80E, 0X80D, 0X80B, 0X808,
0X805, 0X802, 0X800, 0X7FF, 0X7FD, 0X7FB, 0X7FA, 0X7F9, 0X7F8, 0X7F6, 0X7F5, 0X7F5, 0X7F5, 0X7F5, 0X7F5, 0X7F6,
0X7F7, 0X7F8, 0X7F9, 0X7F9, 0X7FA, 0X7FB, 0X7FB, 0X7FC, 0X7FC, 0X7FC, 0X7FB, 0X7FA, 0X7FB, 0X7FD, 0X7FD, 0X7FE,
0X7FD, 0X7FD, 0X7FE, 0X7FF, 0X802, 0X805, 0X805, 0X804, 0X801, 0X7FE, 0X7FC, 0X7FC, 0X7FF, 0X804, 0X808, 0X80B,
0X80A, 0X808, 0X804, 0X802, 0X803, 0X803, 0X804, 0X804, 0X803, 0X802, 0X800, 0X7FE, 0X7FD, 0X7FD, 0X7FC, 0X7FC,
0X7FD, 0X7FE, 0X7FF, 0X7FF, 0X800, 0X800, 0X800, 0X801, 0X801, 0X801, 0X800, 0X7FF, 0X7FE, 0X7FE, 0X7FD, 0X7FB,
0X7FB, 0X7FA, 0X7F9, 0X7F9, 0X7F8, 0X7F6, 0X7F5, 0X7F4, 0X7F3, 0X7F4, 0X7F5, 0X7F6, 0X7F6, 0X7F7, 0X7F6, 0X7F5,
0X7F5, 0X7F7, 0X7FB, 0X7FE, 0X800, 0X801, 0X800, 0X7FE, 0X7FF, 0X801, 0X803, 0X805, 0X806, 0X806, 0X806, 0X806,
0X807, 0X809, 0X80B, 0X80C, 0X80E, 0X80D, 0X80D, 0X80C, 0X80C, 0X80D, 0X80F, 0X80E, 0X80D, 0X809, 0X805, 0X802,
0X7FF, 0X7FE, 0X7FD, 0X7FD, 0X7FE, 0X7FE, 0X7FD, 0X7FB, 0X7FB, 0X7FB, 0X7FB, 0X7FD, 0X7FF, 0X7FF, 0X7FD, 0X7FB,
0X7F9, 0X7F6, 0X7F4, 0X7F4, 0X7F7, 0X7FC, 0X800, 0X804, 0X805, 0X804, 0X803, 0X800, 0X7FE, 0X7FF, 0X801, 0X805,
0X807, 0X807, 0X805, 0X801, 0X7FD, 0X7FA, 0X7F8, 0X7F7, 0X7F7, 0X7F8, 0X7F8, 0X7FB, 0X7FD, 0X801, 0X804, 0X807,
```

```
 0X808, 0X809, 0X809, 0X809, 0X808, 0X808, 0X807, 0X806, 0X804, 0X802, 0X7FF, 0X7FB, 0X7F7, 0X7F2, 0X7F0, 0X7F2,
 0X7F5, 0X7F7, 0X7F8, 0X7F8, 0X7F7, 0X7F5, 0X7F6, 0X7F8, 0X7FC, 0X800, 0X803, 0X805, 0X805, 0X803, 0X7FF, 0X7FC,
 0X7FB, 0X7FB, 0X7FA, 0X7FA, 0X7FA, 0X7F9, 0X7F9, 0X7FB, 0X7FD, 0X7FF, 0X7FE, 0X7FF, 0X800, 0X803, 0X803, 0X803,
 0X804, 0X804, 0X805, 0X805, 0X805, 0X806, 0X807, 0X807, 0X806, 0X804, 0X802, 0X801, 0X803, 0X804, 0X803, 0X801,
 0X7FF, 0X7FF, 0X800, 0X803, 0X806, 0X807, 0X807, 0X806, 0X806, 0X806, 0X807, 0X807, 0X807, 0X806, 0X804, 0X803,
 0X802, 0X800, 0X7FD, 0X7FA, 0X7F8, 0X7F5, 0X7F2, 0X7F1, 0X7F1, 0X7F1, 0X7F3, 0X7F5, 0X7F6, 0X7F5, 0X7F4, 0X7F3,
 0X7F4, 0X7F6, 0X7F8, 0X7FA, 0X7FD, 0X800, 0X802, 0X803, 0X802, 0X801, 0X7FE, 0X7F8, 0X7F1, 0X7EF, 0X7F1, 0X7F7,
 0X7FF, 0X807, 0X80C, 0X80B, 0X808, 0X806, 0X804, 0X804, 0X805, 0X805, 0X806, 0X807, 0X808, 0X80B, 0X80E, 0X811,
 0X811, 0X811, 0X80F, 0X80D, 0X80D, 0X80D, 0X80E, 0X80F, 0X80D, 0X80B, 0X809, 0X807, 0X804, 0X803, 0X803, 0X802,
 0X800, 0X7FF, 0X800, 0X802, 0X806, 0X806, 0X803, 0X7FE, 0X7F7, 0X7F2, 0X7F2, 0X7F5, 0X7FB, 0X7FF, 0X801, 0X7FF,
 0X7FA, 0X7F4, 0X7EF, 0X7EA, 0X7E6, 0X7E3, 0X7E1, 0X7DF, 0X7DF, 0X7E0, 0X7E2, 0X7E6, 0X7E8, 0X7EA, 0X7ED, 0X7EF,
 0X7F4, 0X7FC, 0X804, 0X80A, 0X80F, 0X812, 0X813, 0X814, 0X815, 0X816, 0X817, 0X818, 0X818, 0X818, 0X816, 0X814,
 0X813, 0X811, 0X810, 0X80E, 0X80B, 0X808, 0X804, 0X7FF, 0X7FA, 0X7F7, 0X7F7, 0X7F9, 0X7FB, 0X7FC, 0X7FC, 0X7F8,
 0X7F4, 0X7EF, 0X7EC, 0X7EA, 0X7E8, 0X7E7, 0X7E7, 0X7E7, 0X7E9, 0X7ED, 0X7EF, 0X7F2, 0X7F5, 0X7FA, 0X800, 0X805,
 0X808, 0X809, 0X807, 0X805, 0X803, 0X804, 0X804, 0X806, 0X808, 0X80A, 0X80B, 0X80B, 0X80B, 0X80C, 0X80C, 0X80D,
 0X80D, 0X80D, 0X80D, 0X80D, 0X80C, 0X80C, 0X80C, 0X80C, 0X80C, 0X80B, 0X80B, 0X80A, 0X809, 0X807, 0X805, 0X803
};

unsigned int ph = 0;
unsigned int dp = 0;
IntervalTimer myTimer1;
IntervalTimer myTimer2;

void setup()    // the setup routine runs once when you press reset
{
  pinMode(led, OUTPUT);        // initialize the digital pin as an output
  analogReference(EXTERNAL);   // 3.0V reference
  analogReadResolution(12);    // set ADC to 12 bits
  analogWriteResolution(12);   // set DAC to 12 bits

  myTimer1.begin(update_frequency, 1000);  // set the frequency update interval to 1000 microseconds (1kHz)
```

```
  myTimer2.begin(timer_interrupt, 8);  // set the sample interval to 8 microseconds (125kHz)
  myTimer2.priority(0);                // high priority for timer interrupt
}

void update_frequency()
{
  digitalWrite(led, HIGH);                 // switch on the LED
  dp = analogRead(A0) * 0.139844 * freq;  // calculate the value that is added to the 32-bit phase accumulator
                                           //  2^32 / 125000 / 4095 / 60 = 0.139844
  digitalWrite(led, LOW);                  // switch off the LED
}

void timer_interrupt()
{
  ph += dp;                               // update the phase accumulator
  analogWrite(A12, waveform[ph >> 21]);   // use the most significant 11 bits as index for waveform array, and write to DAC
}

void loop()   // nothing to do in the main loop
{
}
```

3.3 Synthesizer with Teensy 4.0

```
// Multiple numeric controlled oscillators, realized with Teensy 4.0
// Michael Koch 2023

float samplerate = 48000;    // Sample rate in Hz (for 1kHz)
const int tabledepth = 12;    // length of waveform table = 2 ^ tabledepth
const int nco_number = 100;

float freq = 22;       // Output frequency in Hz

#define DIN 7           // PT8211 data pin
#define BCLK 21         // PT8211 clock pin, max 18Mhz
#define WS 20           // PT8211 channel pin, 0 = right, 1 = left
#define HALFCLKns 28    // half clock cycle duration in ns, minimum is 28ns

short waveform[1 << tabledepth];  // waveform table
unsigned int ph[nco_number];      // phase accumulator
unsigned int dp[nco_number];      // delta phase
int chirp[nco_number];            // chirp

int led = 13;                     // LED output
IntervalTimer myTimer1;


void setup()    // the setup routine runs once when you press reset
{
  pinMode(led, OUTPUT);        // initialize the digital pin as an output
  digitalWrite(DIN,0);         // PT8211 pins
  digitalWrite(BCLK,0);
  digitalWrite(WS,0);
  pinMode(DIN, OUTPUT);
  pinMode(BCLK, OUTPUT);
  pinMode(WS, OUTPUT);

  int length = 1 << tabledepth;
  float two_pi = 8.0 * atan(1.0);    // calculate 2*pi
  for(int x = 0; x < length; x++)    // fill the waveform array
  {
    waveform[x] = (int)(32767 * sin(two_pi * (float)x / length));
  }

  randomSeed(13);
  for(int n=0; n < nco_number; n++)    // Set the frequencies, start phases and chirps
  {
```

```
    dp[n] = (unsigned int)((1.0 + n + 0.05 * random(10)) * freq * 65536.0 * 65536.0 / (float)samplerate);  // calculate the value that is added to the
32-bit phase accumulator
    ph[n] = random();
    chirp[n] = 0;
  }

  myTimer1.begin(timer_interrupt, 1e6 / samplerate);  // set the interrupt interval
  myTimer1.priority(0);                  // high priority for timer interrupt
}

void timer_interrupt()
{
  digitalWrite(led, HIGH);            // switch on the LED (only for checking the sample rate)

  int sum = 0;
  for(int n=0; n < nco_number; n++)
  {
    ph[n] += dp[n];                           // update the phase accumulator
    sum += (int)waveform[ph[n] >> (32 - tabledepth)];   // use the most significant 12 bits as index for waveform array
    dp[n] += chirp[n];
  }
  short value = sum / nco_number;

  digitalWriteFast(WS,1);           // write to PT8211 left channel
  for (int i = 15; i >= 0; i--)
  {
    digitalWriteFast(DIN, (value>>i) & 1);
    delayNanoseconds(HALFCLKns);
    digitalWriteFast(BCLK,1);
    delayNanoseconds(HALFCLKns);
    digitalWriteFast(BCLK,0);
  }
  digitalWriteFast(WS,0);           // both analog outputs are updated

  digitalWrite(led, LOW);           // switch off the LED
}

void loop()   // nothing to do in the main loop
{
}
```

3.4 Synthesizer with Teensy 4.0, V2

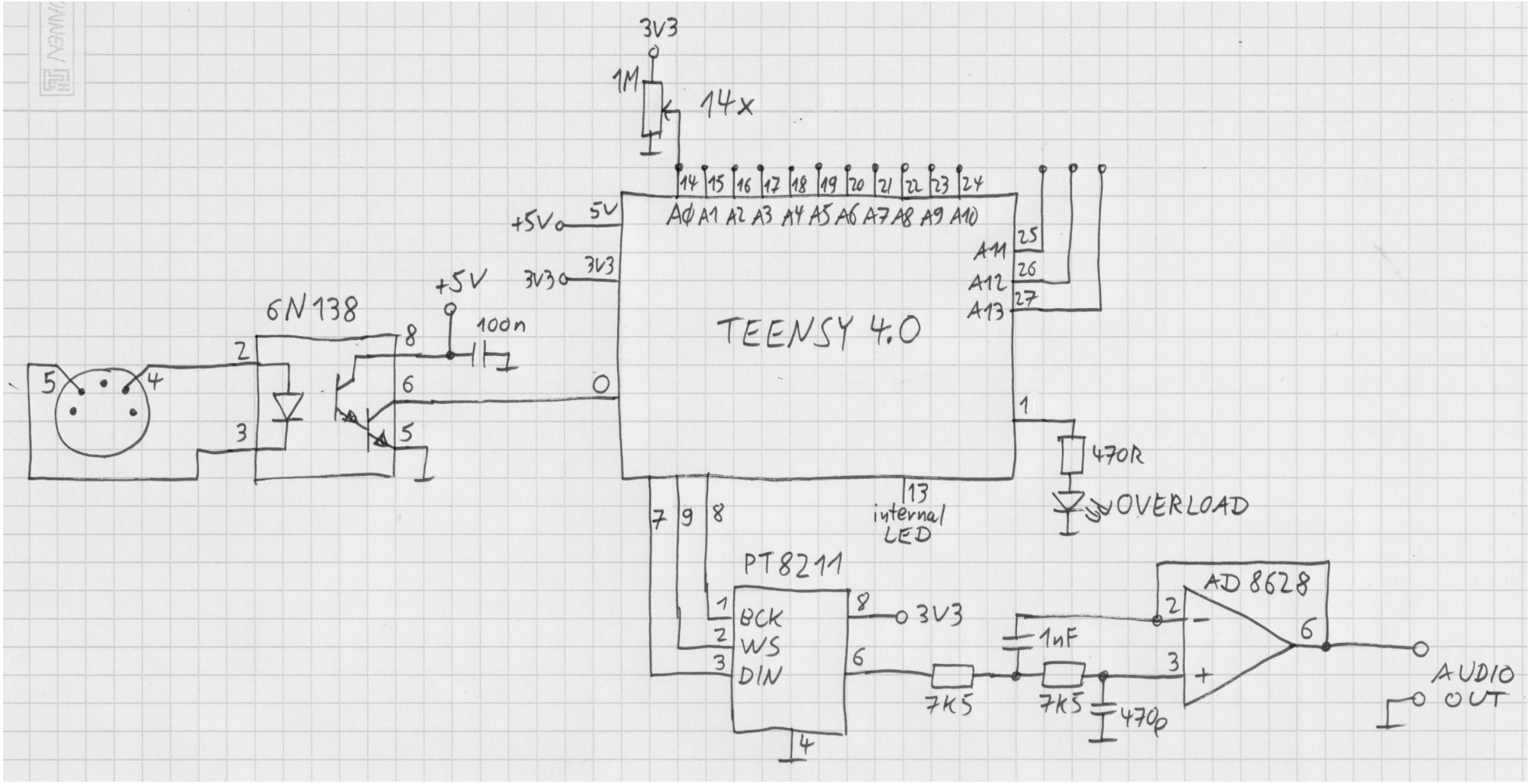
```
// Multiple numeric controlled oscillators, realized with Teensy 4.0
// Michael Koch 2023

float samplerate = 48000;    // Sample rate in Hz (for 1kHz)
const int tabledepth = 12;    // length of waveform table = 2 ^ tabledepth
const int nco_number = 25;
const int normalize = 10;

float freq1 = 800;       // Output frequency in Hz
float freq2 = 1600;       // Output frequency in Hz
float mod_freq = 0.2;      // Modulation frequency
float dopplerfreq = 100;

#define DIN 7           // PT8211 data pin
#define BCLK 21         // PT8211 clock pin, max 18Mhz
#define WS 20           // PT8211 channel pin, 0 = right, 1 = left
#define HALFCLKns 28    // half clock cycle duration in ns, minimum is 28ns

short waveform[1 << tabledepth];  // waveform table
short mod_waveform[1 << tabledepth];  // waveform table
long doppler_waveform[1 << tabledepth];    // doppler table
unsigned long ph[nco_number];      // phase accumulator
unsigned long dp[nco_number];      // delta phase
long chirp[nco_number];            // chirp
unsigned long mod_ph[nco_number];      // phase accumulator
unsigned long mod_dp[nco_number];      // delta phase
unsigned long delaytime[nco_number];       // delay time in DSS cycles
unsigned long counter[nco_number];     // delay counter
unsigned long minfreq;
unsigned long maxfreq;

elapsedMillis ms;

int led = 13;                     // LED output
IntervalTimer myTimer1;

void setup()    // the setup routine runs once when you press reset
{
  pinMode(led, OUTPUT);        // initialize the digital pin as an output
  digitalWrite(DIN,0);         // PT8211 pins
  digitalWrite(BCLK,0);
  digitalWrite(WS,0);
  pinMode(DIN, OUTPUT);
  pinMode(BCLK, OUTPUT);
  pinMode(WS, OUTPUT);
```

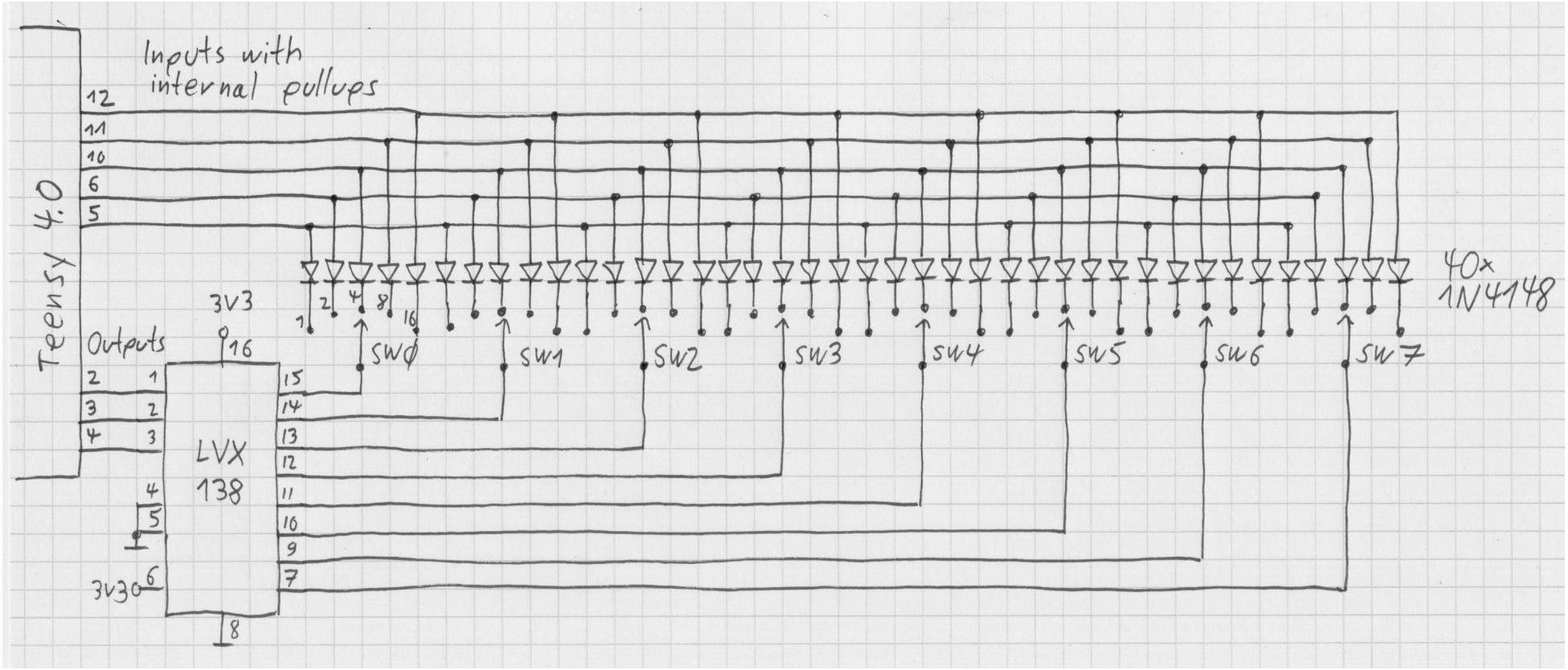
```
  int length = 1 << tabledepth;
  float pi = 4.0 * atan(1.0);        // calculate pi
  float two_pi = 8.0 * atan(1.0);    // calculate 2*pi
  for(int x = 0; x < length; x++)    // fill the waveform array
  {
    waveform[x] = (int)(32767 * sin(two_pi * (float)x / length));   // sine
    //waveform[x] = (int)(65536 * (float)x / length);   // sawtooth
  }

  for(int x = 0; x < length; x++)    // fill the modulation waveform array
  {
    mod_waveform[x] = (int)(4096 * (0.5 - 0.5 * cos(two_pi * (float)x / length)));   // cosine
  }

  for(int x = 0; x < length; x++)    // fill the doppler frequency array
  {
    doppler_waveform[x] = (int)(dopplerfreq * 65536.0 * 65536.0 / (float)samplerate * (cos(pi * (float)x / length)));
  }

  randomSeed(2);
  minfreq = (unsigned int)(freq1 * 65536.0 * 65536.0 / (float)samplerate);
  maxfreq = (unsigned int)(freq2 * 65536.0 * 65536.0 / (float)samplerate);

  for(int n=0; n < nco_number; n++)    // Set the frequencies, start phases and chirps
  {
    dp[n] = minfreq + random(maxfreq - minfreq);  // calculate the value that is added to the 32-bit phase accumulator
    ph[n] = 2 * random(2147483648);
    chirp[n] = 0 * (-1000 + random(2000));
    mod_dp[n] = (unsigned long)(mod_freq * 65536.0 * 65536.0 / (float)samplerate);
    mod_ph[n] = 2* random(2147483648);
    delaytime[n] = random(500000);
    counter[n] = 0;
  }

  //delaytime[0] = 72000;
  //delaytime[1] = 155000;
  //delaytime[2] = 199000;
  //mod_ph[0] = 0;

  myTimer1.begin(timer_interrupt, 1e6 / samplerate);  // set the interrupt interval
  myTimer1.priority(0);                 // high priority for timer interrupt

  Serial.begin(38400);
  ms = 0;
}

void timer_interrupt()
```

```
{
  digitalWrite(led, HIGH);            // switch on the LED (only for checking the sample rate)

  int sum = 0;
  int mod = 0;
  for(int n=0; n < nco_number; n++)
  {
    if (counter[n] == 0)
    {
      if (mod_ph[n] < 0x800000)
      {
        mod_ph[n] += mod_dp[n];
      }
      else
      {
        mod_ph[n] += mod_dp[n];
        if (mod_ph[n] < 0x800000)
          counter[n] = delaytime[n];
      }
    }
    else
    {
      counter[n]--;
    }
    mod = mod_waveform[mod_ph[n] >> (32 - tabledepth)];   // use the most significant 12 bits as index for waveform array

    ph[n] += dp[n] + doppler_waveform[mod_ph[n] >> (32 - tabledepth)];          // update the phase accumulator
    sum += (waveform[ph[n] >> (32 - tabledepth)] * mod) >> 12;   // use the most significant bits as index for waveform array
    dp[n] += chirp[n];
    if((dp[n] <= minfreq) || (dp[n] >= maxfreq))
      chirp[n] = -chirp[n];
  }
  sum = sum / normalize;
  if (sum > 32767) sum = 32767;
  if (sum < -32768) sum = -32768;

  digitalWriteFast(WS,1);           // write to PT8211 left channel
  for (int i = 15; i >= 0; i--)
  {
    digitalWriteFast(DIN, (sum>>i) & 1);
    delayNanoseconds(HALFCLKns);
    digitalWriteFast(BCLK,1);
    delayNanoseconds(HALFCLKns);
    digitalWriteFast(BCLK,0);
  }
  digitalWriteFast(WS,0);           // both analog outputs are updated

  digitalWrite(led, LOW);           // switch off the LED
```

```
}

void loop()   // nothing to do in the main loop
{
  if(ms >= 2500)
  {
    //Serial.println("Hello World");
    Serial.println(random(4294967295));
    //paused[0] = 0;
    ms = 0;
  }
}
```

3.5 New Synthesizer Board





Test for potentiometers and code switches:

```
// Synthesizer board, test for potentiometers and code switches
// Michael Koch 2023

#define DIN 7           // PT8211 data pin
#define BCLK 8          // PT8211 clock pin, max 18Mhz
#define WS 9            // PT8211 channel pin, 0 = right, 1 = left
#define LVX138_A0 2
#define LVX138_A1 3
#define LVX138_A2 4
#define MATRIX_D0 5
```

```
#define MATRIX_D1 6
#define MATRIX_D2 10
#define MATRIX_D3 11
#define MATRIX_D4 12
#define LED 13          // LED on Teensy 4.0
#define OVERFLOW 1      // overflow LED

elapsedMillis ms;

int analog[14];
int codeswitch[8];

void setup()    // the setup routine runs once when you press reset
{
  pinMode(LED, OUTPUT);        // initialize the digital pin as an output
  pinMode(OVERFLOW, OUTPUT);   // initialize the digital pin as an output
  pinMode(DIN, OUTPUT);
  pinMode(BCLK, OUTPUT);
  pinMode(WS, OUTPUT);
  pinMode(LVX138_A0, OUTPUT);
  pinMode(LVX138_A1, OUTPUT);
  pinMode(LVX138_A2, OUTPUT);
  pinMode(MATRIX_D0, INPUT_PULLUP);
  pinMode(MATRIX_D1, INPUT_PULLUP);
  pinMode(MATRIX_D2, INPUT_PULLUP);
  pinMode(MATRIX_D3, INPUT_PULLUP);
  pinMode(MATRIX_D4, INPUT_PULLUP);
  pinMode(A0,INPUT_DISABLE);   // make the input high impedance
  pinMode(A1,INPUT_DISABLE);   // make the input high impedance
  pinMode(A2,INPUT_DISABLE);   // make the input high impedance
  pinMode(A3,INPUT_DISABLE);   // make the input high impedance
  pinMode(A4,INPUT_DISABLE);   // make the input high impedance
  pinMode(A5,INPUT_DISABLE);   // make the input high impedance
  pinMode(A6,INPUT_DISABLE);   // make the input high impedance
  pinMode(A7,INPUT_DISABLE);   // make the input high impedance
  pinMode(A8,INPUT_DISABLE);   // make the input high impedance
  pinMode(A9,INPUT_DISABLE);   // make the input high impedance
  pinMode(A10,INPUT_DISABLE);   // make the input high impedance
  pinMode(A11,INPUT_DISABLE);   // make the input high impedance
  pinMode(A12,INPUT_DISABLE);   // make the input high impedance
  pinMode(A13,INPUT_DISABLE);   // make the input high impedance

  digitalWrite(DIN,0);         // initialize PT8211 pins
  digitalWrite(BCLK,0);
  digitalWrite(WS,0);

  Serial.begin(38400);
  ms = 0;
```

```
}

String int2string(int val, unsigned int digits)   // convert int to string, left-padded with spaces
{
  String s = String(val);
  while (s.length() < digits)
    s = " " + s;
  return(s);
}

void loop()   // nothing to do in the main loop
{
  if(ms >= 250)
  {
    digitalWrite(LED, HIGH);   // switch on the LED
    analog[0] = analogRead(A0);
    analog[1] = analogRead(A1);
    analog[2] = analogRead(A2);
    analog[3] = analogRead(A3);
    analog[4] = analogRead(A4);
    analog[5] = analogRead(A5);
    analog[6] = analogRead(A6);
    analog[7] = analogRead(A7);
    analog[8] = analogRead(A8);
    analog[9] = analogRead(A9);
    analog[10] = analogRead(A10);
    analog[11] = analogRead(A11);
    analog[12] = analogRead(A12);
    analog[13] = analogRead(A13);

    for (int n = 0; n < 8; n++)    // read the 8 code switches
    {
      digitalWriteFast(LVX138_A0, n&1);
      digitalWriteFast(LVX138_A1, n&2);
      digitalWriteFast(LVX138_A2, n&4);
      delayNanoseconds(400);       // because internal pullup is slow
      int code = 0;
      if (digitalRead(MATRIX_D0)) code += 1;
      if (digitalRead(MATRIX_D1)) code += 2;
      if (digitalRead(MATRIX_D2)) code += 4;
      if (digitalRead(MATRIX_D3)) code += 8;
      codeswitch[n] = code;
    }

    for (int n = 0; n < 14; n++)
      Serial.print(int2string(analog[n],4)+" ");
    Serial.print("     ");
    for (int n = 0; n < 8; n++)
```

```
      Serial.print(int2string(codeswitch[n],2)+" ");
    Serial.println();

    delay(125);
    digitalWrite(LED, LOW);   // switch off the LED
    ms = 0;
  }
}
```

Test for MIDI input: (found here, but changed some things: https://www.pjrc.com/teensy/td_libs_MIDI.html)

```
// MIDI Input Test - for use with Teensy or boards where Serial is separate from MIDI
// As MIDI messages arrive, they are printed to the Serial Monitor.

#include <MIDI.h>
#define LED 13          // LED on Teensy 4.0

MIDI_CREATE_INSTANCE(HardwareSerial, Serial1, MIDI);

void setup() {
  pinMode(LED, OUTPUT);        // initialize the digital pin as an output
  MIDI.begin(MIDI_CHANNEL_OMNI);
  Serial.begin(57600);
  Serial.println("MIDI Input Test");
}

unsigned long t=0;

void loop() {
  int type, note, velocity, channel, d1, d2;
  if (MIDI.read()) {                    // Is there a MIDI message incoming ?
    type = MIDI.getType();
    switch (type) {
      case midi::NoteOn:
        note = MIDI.getData1();
        velocity = MIDI.getData2();
        channel = MIDI.getChannel();
        if (velocity > 0) {
          Serial.println(String("Note On:  ch=") + channel + ", note=" + note + ", velocity=" + velocity);
          digitalWrite(LED, HIGH);   // switch on the LED
        } else {
          Serial.println(String("Note Off: ch=") + channel + ", note=" + note);
          digitalWrite(LED, LOW);   // switch off the LED
        }
        break;
      case midi::NoteOff:
        note = MIDI.getData1();
```

```
        velocity = MIDI.getData2();
        channel = MIDI.getChannel();
        Serial.println(String("Note Off: ch=") + channel + ", note=" + note + ", velocity=" + velocity);
        digitalWrite(LED, LOW);   // switch off the LED
        break;
      default:
        d1 = MIDI.getData1();
        d2 = MIDI.getData2();
        Serial.println(String("Message, type=") + type + ", data = " + d1 + " " + d2);
    }
    t = millis();
  }
  if (millis() - t > 10000) {
    t += 10000;
    Serial.println("(inactivity)");
  }
}
```

3.6 RGB LEDs and LED Cubes

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

https://cpldcpu.wordpress.com/2022/01/23/controlling-rgb-leds-with-only-the-powerlines-anatomy-of-a-christmas-light-string/

8x8x8 2-pin RGB: https://de.aliexpress.com/item/1005004131003602.html https://de.aliexpress.com/item/1005005499684157.html

8x8x8 4-pin RGB: https://de.aliexpress.com/item/32888551895.html

4x4x4 2-pin RGB: https://de.aliexpress.com/item/1005004130874725.html

5x5x5 10mm 4-pin RGB: https://de.elv.com/elv-5x5x5-rgb-cube-rgbc555-bausatz-ohne-leds-105043

Ball with 2-pin RGB: https://de.aliexpress.com/item/1005004165948785.html These are RGB-LEDs with automatic color cycling ???

12x12x12 4-pin RGB: https://de.aliexpress.com/item/33010021040.html https://de.aliexpress.com/item/1005003286781904.html

16x16x16 4-pin RGB? https://de.aliexpress.com/item/1005005906423503.html

16x16x16: https://www.ebay.com/itm/134688338387 https://www.youtube.com/watch?v=1eQZz2Dhu6M

50 pieces of 2-pin RGB LEDs: https://de.aliexpress.com/item/1005004188238468.html

https://www.sparkfun.com/products/21209

https://eu.robotshop.com/de/products/rgb-led-owire-2-pin-pth-4mm-concave

https://cdn.sparkfun.com/assets/9/f/1/c/6/CZineLight_0-Wire_Communication_Protocol.pdf

https://github.com/MaltWhiskey/Mega-Cube

https://www.etereshop.com/high-density-smart-3d-led-cube-with-16k-leds/

RGB LEDs with internal chip (Data in, Data out): APA106

WS2812: https://www.elektronik-kompendium.de/sites/praxis/bauteil_ws2812.htm

RGB LEDs from Würth Elektronik:
Type 1312121320437, size 2.2mm x 2.2mm, PLCC6 https://www.we-online.com/components/products/datasheet/1312121320437.pdf
I didn't yet figure out the purpose of the "BI" pin. It's not described in the datasheet.

https://cdn-shop.adafruit.com/datasheets/WS2812.pdf

T0H: 0.35µs +- 0.15µs (0.20µs ... 0.50µs) T0L: 0.80µs +- 0.15µs (0.65µs ... 0.95µs) Reset: >50µs Low

T1H: 0.70µs +- 0.15µs (0.55µs ... 0.85µs) T1L: 0.60µs +- 0.15µs (0.45µs ... 0.75µs)

Logic 0		Logic 1		Sum	BPS	Comment
High	Low	High	Low			
0.35µs (+0.00µs)	0.90µs (+0.10µs)	0.90µs (+0.20µs)	0.35µs (-0.25µs)	1.25µs	800kHz	symmetric timing, out of tolerance
0.40µs (+0.05µs)	0.85µs (+0.05µs)	0.85µs (+0.15µs)	0.40µs (-0.20µs)	1.25µs	800kHz	symmetric timing, out of tolerance
0.45µs (+0.10µs)	0.80µs (+0.00µs)	0.80µs (+0.10µs)	0.45µs (-0.15µs)	1.25µs	800kHz	symmetric timing
0.50µs (+0.15µs)	0.75µs (-0.05µs)	0.75µs (+0.05µs)	0.50µs (-0.10µs)	1.25µs	800kHz	symmetric timing
0.40µs (+0.05µs)	0.80µs (+0.00µs)	0.65µs (-0.05µs)	0.55µs (-0.05µs)	1.20µs	833kHz	minimized tolerances
0.35µs (+0.00µs)	0.75µs (-0.05µs)	0.60µs (-0.10µs)	0.50µs (-0.10µs)	1.10µs	909kHz	
0.30µs (-0.05µs)	0.70µs (-0.10µs)	0.55µs (-0.15µs)	0.45µs (-0.15µs)	1.00µs	1MHz	minimized sum
0.33µs (-0.02µs)	1.00µs (+0.20µs)	1.00µs (+0.30µs)	0.33µs (-0.27µs)	1.33µs	750kHz	SAMD21 SPI 6MHz 2/8 6/8 (or 3MHz 1/4 3/4), out of tolerance
0.33µs (-0.02µs)	0.67µs (+0.13µs)	0.67µs (-0.03µs)	0.33µs (-0.27µs)	1.00µs	1MHz	SAMD21 SPI 3MHz 1/3 2/3, out of tolerance

https://www.mikrocontroller.net/articles/WS2812_Ansteuerung

T0H: 0.35µs +- 0.15µs (0.20µs ... 0.50µs) T0L: 0.90µs +- 0.15µs (0.75µs ... 1.05µs) Reset: >50µs (ab V4 >280µs)

T1H: 0.90µs +- 0.15µs (0.75µs ... 1.05µs) T1L: 0.35µs +- 0.15µs (0.20µs ... 0.50µs)

Logic 0		Logic 1		Sum	BPS	Comment
High	Low	High	Low			
0.35µs (+0.00µs)	0.90µs (+0.00µs)	0.90µs (+0.00µs)	0.35µs (+0.00µs)	1.25µs	800kHz	symmetric timing
0.33µs (-0.02µs)	1.00µs (+0.20µs)	1.00µs (+0.30µs)	0.33µs (-0.27µs)	1.33µs	750kHz	SAMD21 SPI 6MHz 2/8 6/8 or 3MHz 1/4 3/4

Teensy 2.0 is too slow for generating this timing. The shortest possible pulse width seems to be about 1µs.

See also this library: https://www.pjrc.com/teensy/td_libs_OctoWS2811.html

Test program for 8 WS2812 RGB LEDs, with Teensy LC:

```
// RGB-LED Test V1, for Teensy LC
// Michael Koch 2023

int DI = 17;

#define LEDS 8

#define T1H 700
#define T1L 0
#define T0H 0
#define T0L 600

int a[LEDS];

void setup()
{
  pinMode(DI, OUTPUT);  // initialize the digital pin as an output

  a[0] = 0x0000ff00;     // red
  a[1] = 0x0080ff00;     // orange
  a[2] = 0x00ffff00;     // yellow
  a[3] = 0x00ff0000;     // green
  a[4] = 0x00ff0060;     // blue-green
  a[5] = 0x008000ff;     // cyan
  a[6] = 0x000000ff;     // blue
  a[7] = 0x000080ff;     // violet
}

void write_leds()
{
  digitalWriteFast(DI, LOW);
  delayNanoseconds(50000);  // reset     Use at least 70000ns for small 2mm x 2mm RGB LEDs !

  noInterrupts();
  for (int n = 0; n < LEDS; n++)
  {
    int rgb = a[n];
    for (int i = 0; i < 24; i++)
    {
      digitalWriteFast(DI, HIGH);
      if ((rgb <<= 1) & 0x01000000)  // write logic 1
```

```
      {
        delayNanoseconds(T1H);
        digitalWriteFast(DI, LOW);
        delayNanoseconds(T1L);
      }
      else                          // write logic 0
      {
        delayNanoseconds(T0H);
        digitalWriteFast(DI, LOW);
        delayNanoseconds(T0L);
      }
    }
  }
  digitalWriteFast(DI, HIGH);
  interrupts();
}

void loop()
{
  write_leds();
  delay(500);
  int aa = a[0];
  for(int i = 1; i < LEDS; i++)
  a[i-1] = a[i];
  a[LEDS-1] = aa;
}
```

This is a similar test program for Seeed XIAO SAMD21, using fast writing to the output pin. The 3.3V to 5V chip is a non-inverting 74AHCT1G125.

Transmission time for 24 bits (one WS2812 RGB LED) is about 26.7 µs.

```
int DI = D10;

volatile EPortType port = g_APinDescription[DI].ulPort;
volatile uint32_t pin = g_APinDescription[DI].ulPin;
volatile uint32_t pinMask = (1ul << pin);

#define LEDS 40

#define T1H 8
#define T0L 7

int led[LEDS];

void setup() {
  pinMode(DI, OUTPUT);  // initialize the digital pin as an output

  led[0] = 0x0000ff00;     // red
  led[1] = 0x0080ff00;     // orange
  led[2] = 0x00ffff00;     // yellow
  led[3] = 0x00ff0000;     // green
  led[4] = 0x00ff0060;     // blue-green
  led[5] = 0x008000ff;     // cyan
  led[6] = 0x000050ff;     // blue
  led[7] = 0x000080ff;     // violet

  Serial.begin(38400);
  delay(2000);             // give the serial monitor some time to get ready for listening
  while (!Serial) ;        // wait for serial port to connect. Needed for native USB
  Serial.println(port);    // this is 0
  Serial.println(pinMask); // this is 64
}

void write_leds()
{
  PORT->Group[0].OUTCLR.reg = 64;  // set pin low
  delayMicroseconds(50);  // reset

  noInterrupts();
  for (int n = 0; n < LEDS; n++)
  {
```

```
    int rgb = led[n];
    for (int i = 0; i < 24; i++)
    {
      if ((rgb <<= 1) & 0x01000000)            // write logic 1
      {
        PORT->Group[0].OUTSET.reg = 64;  // set pin high
        for (int t = T1H; t != 0; t--)
          asm volatile("nop;");
        PORT->Group[0].OUTCLR.reg = 64;  // set pin low
      }
      else                                      // write logic 0
      {
        PORT->Group[0].OUTSET.reg = 64;  // set pin high
        PORT->Group[0].OUTCLR.reg = 64;  // set pin low
        for (int t = T0L; t != 0; t--)
          asm volatile("nop;");
      }
    }
  }
  interrupts();
}

void loop()
{
  write_leds();
  delay(100);
  int aa = led[0];
  for(int i = 1; i < LEDS; i++)
  led[i-1] = led[i];
  led[LEDS-1] = aa;
}
```

Note: The pulse width may become faster than 200ns sometimes (if the code is in cache?)

For software, see also this example from Seeed: https://wiki.seeedstudio.com/rgb_matrix_for_xiao/

The same test program for Seeed XIAO SAMD21, using the SPI interface and a 74AHCT1G04 inverter for 3.3V to 5V conversion.

Transmission time for 24 bits (one WS2812 RGB LED) is about 54.6 µs.

```
// RGB-LED Test, for Seeed XIAO SAMD21
// Michael Koch 2024

#include <SPI.h>

#define LEDS 20

int led[LEDS];
byte spibuf[24];

void setup()
{
  SPI.begin();
  SPI.setClockDivider(SPI_CLOCK_DIV4);  // 48 MHz / 4 = 12 MHz
  SPI.setDataMode(SPI_MODE0);

  //       0x00GGRRBB
  led[0] = 0x0000ff00;     // red
  led[1] = 0x0080ff00;     // orange
  led[2] = 0x00ffff00;     // yellow
  led[3] = 0x00ff0000;     // green
  led[4] = 0x00ff0060;     // blue-green
  led[5] = 0x008000ff;     // cyan
  led[6] = 0x000040ff;     // blue
  led[7] = 0x000080ff;     // violet
}

void write_leds()
{
  delayMicroseconds(50);  // reset
  for (int n = 0; n < LEDS; n++)
  {
    int rgb = led[n];
    for (int i = 0; i < 24; i++)
    {
      if ((rgb <<= 1) & 0x01000000)    // write logic 1
        spibuf[i] = 0x03;              //  6 bits are set after 1G04 inverter
      else                             // write logic 0
        spibuf[i] = 0x3f;              //  2 bits are set after 1G04 inverter
    }
```

```
    SPI.transfer(spibuf,24);            // send 24 bits for one RGB LED
  }
}

void loop()
{
  write_leds();
  delay(1000);
  int aa = led[0];
  for(int i = 1; i < LEDS; i++)
  led[i-1] = led[i];
  led[LEDS-1] = aa;
}
```

Notes:

It seems not important how long the low impulse at the LED is (unless it's not as long as the reset time). The duration of the high impulse is important.

Measurement of SPI port timing, with SPI.transfer(buffer, length) function as in the avove example:

SPI.setClockDivider	Width of each bit	Time from one byte to next byte, including gap
SPI_CLOCK_DIV2	0.083 µs ???	2.0 µs
SPI_CLOCK_DIV4	0.125 µs	2.28 µs
SPI_CLOCK_DIV8	0.25 µs	3.28 µs
SPI_CLOCK_DIV16	0.5 µs	5.36 µs

Test program for 16x16 matrix of WS2812 LEDs with Teensy 4.0 and AHCT1G125 driver:

```
// 16x16 Array RGB-LED Test, for Teensy 4.0
// Michael Koch 2024

int DI = 23;

#define LEDS 256

#define T1H 900
#define T1L 350
#define T0H 350
#define T0L 900

int a[LEDS];

void setup()
{
  pinMode(DI, OUTPUT);  // initialize the digital pin as an output

  a[0] = 0x00002000;     // red
  a[1] = 0x00102000;     // orange
  a[2] = 0x00202000;     // yellow
  a[3] = 0x00200000;     // green
  a[4] = 0x0020000c;     // blue-green
  a[5] = 0x00100020;     // cyan
  a[6] = 0x00000020;     // blue
  a[7] = 0x00001020;     // violet
}

void clear_leds()
```

```
{
  for(int i = 0; i < LEDS; i++)
    a[i] = 0;
}

void set_led(int x, int y, int color)   // 0,0 is top left
{
  a[255 - x - 16 * y + (y%2) * (2 * x - 15)] = color;
}

void write_leds()
{
  digitalWriteFast(DI, LOW);
  delayMicroseconds(70);  // reset     Use at least 70000ns for small 2mm x 2mm RGB LEDs !

  noInterrupts();
  for (int n = 0; n < LEDS; n++)
  {
    int rgb = a[n];
    for (int i = 0; i < 24; i++)
    {
      digitalWriteFast(DI, HIGH);
      if ((rgb <<= 1) & 0x01000000)  // write logic 1
      {
        delayNanoseconds(T1H);
        digitalWriteFast(DI, LOW);
        delayNanoseconds(T1L);
      }
      else                          // write logic 0
      {
        delayNanoseconds(T0H);
```

```
        digitalWriteFast(DI, LOW);
        delayNanoseconds(T0L);
      }
    }
  }
  digitalWriteFast(DI, HIGH);
  interrupts();
}

void loop()
{
  clear_leds();
  a[0] = 0x00002000;     // red
  a[1] = 0x00102000;     // orange
  a[2] = 0x00202000;     // yellow
  a[3] = 0x00200000;     // green
  a[4] = 0x0020000c;     // blue-green
  a[5] = 0x00100020;     // cyan
  a[6] = 0x00000020;     // blue
  a[7] = 0x00001020;     // violet

  set_led(0,13,0x00000100);
  set_led(1,13,0x00000200);
  set_led(2,13,0x00000400);
  set_led(3,13,0x00000800);
  set_led(4,13,0x00001000);
  set_led(5,13,0x00002000);
  set_led(6,13,0x00004000);
  set_led(7,13,0x00008000);
  set_led(8,13,0x0000ff00);
```

```
  set_led(0,11,0x00000001);
  set_led(1,11,0x00000002);
  set_led(2,11,0x00000004);
  set_led(3,11,0x00000008);
  set_led(4,11,0x00000010);
  set_led(5,11,0x00000020);
  set_led(6,11,0x00000040);
  set_led(7,11,0x00000080);
  set_led(8,11,0x000000ff);

  for(int i = 0; i < 16; i++)
    set_led(i,i,0x00100000);
  write_leds();
  delay(500);
}
```

SOT223-5 Line driver chips for 3.3V to 5V conversion:

74HCT1G125 Low-level input voltage: 0.8 V High-level input voltage: 2.0 V
Suitable for 3.3V to 5V level conversion, but he inputs have clamp diodes, which means a series resistor must be used for the case when the line driver has no supply voltage.
Data sheet: https://assets.nexperia.com/documents/data-sheet/74HC_HCT1G125_Q100.pdf https://docs.rs-online.com/d874/0900766b81200ab8.pdf

74AHCT1G125 Low-level input voltage: 0.8 V High-level input voltage: 2.0 V
The inputs are over-voltage tolerant up to 5.5V, no series resistor is required. Suitable for 3.3V to 5V level conversion.
Data sheet: https://assets.nexperia.com/documents/data-sheet/74AHC_AHCT1G125.pdf

74AHCT1G04 Inverter, Low-level input voltage: 0.8 V High-level input voltage: 2.0 V
The inputs are over-voltage tolerant up to 5.5V, no series resistor is required. Suitable for 3.3V to 5V level conversion.
Data sheet: https://www.farnell.com/datasheets/1919284.pdf https://assets.nexperia.com/documents/data-sheet/74AHC_AHCT1G04_Q100.pdf

74LVC1G125 Low-level input voltage: 0.3 * VCC High-level input voltage: 0.7 × VCC Not suitable for 3.3V to 5V level conversion.
The inputs are over-voltage tolerant up to 5.5V, no series resistor is required.
Data sheet: https://assets.nexperia.com/documents/data-sheet/74LVC1G125.pdf

3.7 Training-Timer with Seeed XIAO SAMD21

```
#include <TimerTC3.h>

int training, pause;
int count;
int ton;

void setup()
{
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(D3, INPUT_PULLUP);   // pullup resistors for mode switch inputs
  pinMode(D4, INPUT_PULLUP);
  pinMode(D5, INPUT_PULLUP);
  pinMode(D6, INPUT_PULLUP);
  pinMode(D7, OUTPUT);

  ton = 350;
  count = -10;

  TimerTc3.initialize(1000000);   // 1 second
  TimerTc3.attachInterrupt(timerIsr);
}

void timerIsr()   // timer interrupt service routine
{
  training = 30 + 10 * digitalRead(D3) + 20 * digitalRead(D4);
  pause = 5 + 5 * digitalRead(D5) + 10 * digitalRead(D6);
```

```
  count++;

  if ((count >= -3) && (count <= -1))   // 3x short beep
  {
    for(int n = 0; n < 100; n++)
    {
      digitalWrite(D7, LOW);
      delayMicroseconds(ton);
      digitalWrite(D7, HIGH);
      delayMicroseconds(ton);
    }
  }

if (count == 0)   // long start beep
  {
    for(int n = 0; n < 1200; n++)
    {
      digitalWrite(D7, LOW);
      delayMicroseconds(ton);
      digitalWrite(D7, HIGH);
      delayMicroseconds(ton);
    }
  }

if (count == training/2)   // short middle beep
  {
    for(int n = 0; n < 100; n++)
    {
      digitalWrite(D7, LOW);
      delayMicroseconds(ton);
      digitalWrite(D7, HIGH);
```

```
      delayMicroseconds(ton);
    }
  }

  if ((count >= training-3) && (count <= training-1))   // 3x short beep
  {
    for(int n = 0; n < 100; n++)
    {
      digitalWrite(D7, LOW);
      delayMicroseconds(ton);
      digitalWrite(D7, HIGH);
      delayMicroseconds(ton);
    }
  }

if (count >= training)   // long end beep
  {
    for(int n = 0; n < 600; n++)
    {
      digitalWrite(D7, LOW);
      delayMicroseconds(2*ton);
      digitalWrite(D7, HIGH);
      delayMicroseconds(2*ton);
    }
    count = -pause;
  }
}

void loop() {  }
```

4 Hardware

4.1 Teensy 2.0

https://www.pjrc.com/store/teensy.html

4.2 Teensy LC

https://www.pjrc.com/store/teensylc.html

4.3 Teensy 4.0

https://www.pjrc.com/store/teensy40.html

Data sheet of CPU chip: https://www.pjrc.com/teensy/IMXRT1060CEC_rev0_1.pdf

Output pins: 3.3V, not 5V tolerant, maximum output current 10mA, recommended output current 4mA

4.4 Seeed Studio XIAO SAMD21

https://wiki.seeedstudio.com/SeeedStudio_XIAO_Series_Introduction/

https://wiki.seeedstudio.com/Seeeduino-XIAO/

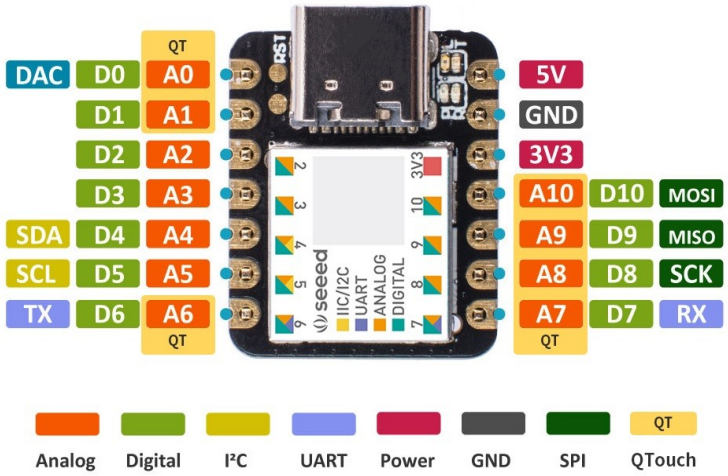
SAMD21 Data sheet: https://files.seeedstudio.com/wiki/Seeeduino-XIAO/res/ATSAMD21G18A-MU-Datasheet.pdf

Schematic diagram: https://files.seeedstudio.com/wiki/Seeeduino-XIAO/res/Seeeduino-XIAO-v1.0-SCH-191112.pdf

Wearable Projects Step by Step: https://files.seeedstudio.com/wiki/Seeeduino-XIAO/res/Seeeduino-XIAO-in-Action-Minitype%EF%BC%86Wearable-Projects-Step-by-Step.pdf

The ADC is 12-bit and the DAC is 10-bit.

The clock frequency is 48 MHz, derived via PLL from a 32.768 kHz crystal.



Timer interrups in Seeed XIAO SAMD21 can be made with TimerTC3 or TimerTCC0:

```
#include <TimerTCC0.h>

void setup()
{
  TimerTcc0.initialize(1000000);
  TimerTcc0.attachInterrupt(timerIsr);
}

void loop()
{
}

void timerIsr()
{
  // your ISR code
}
```

```
#include <TimerTC3.h>

void setup()
{
    TimerTc3.initialize(1000000);
    TimerTc3.attachInterrupt(timerIsr);
}

void loop()
{
}

void timerIsr()
{
  // your ISR code
}
```

It might also be possible to use this library (not yet tested):

```
#include <SAMDTimerInterrupt.h>
```

4.5 Seeed XIAO SAMD21 Debugging

Use another SAMD21 as debugger: https://embeddedcomputing.weebly.com/the-5-programmer-debugger.html

https://wiki.seeedstudio.com/Seeeduino-XIAO-DAPLink/ but I can't find the documentation

https://wiki.seeedstudio.com/Arduino-DAPLink/

It's all quite complicated stuff. I don't understand it. If someone can explain how it works, please let me know.

https://docs.arduino.cc/software/ide-v2/tutorials/ide-v2-debugger/

https://forum.arduino.cc/t/debugging-questions/700058

Installing and configuring the required software for debugging is ultra-complicated stuff. It's best to save all this time and debug with the good old Serial.print() method.

4.6 I2C Port Scanner for Seeed XIAO SAMD21

```
#include <Wire.h>

void setup(void) {
  Serial.begin(115200);
  while (!Serial)
    delay(10);
  Serial.println("I2C Port Scanner");
  Wire.begin();
}

void loop()
{
  byte error, address;
  int nDevices;

  Serial.println("Scanning...");

  nDevices = 0;
  for(address = 1; address < 127; address++ )
  {
    // The i2c_scanner uses the return value of
    // the Write.endTransmisstion to see if
    // a device did acknowledge to the address.
    Wire.beginTransmission(address);
    error = Wire.endTransmission();

    if (error == 0)
    {
      Serial.print("I2C device found at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.print(address,HEX);
      Serial.println("  !");

      nDevices++;
    }
```

```
    else if (error==4)
    {
      Serial.print("Unknow error at address 0x");
      if (address<16)
        Serial.print("0");
      Serial.println(address,HEX);
    }
  }
  if (nDevices == 0)
    Serial.println("No I2C devices found\n");
  else
    Serial.println("done\n");

  delay(5000);           // wait 5 seconds for next scan
}
```

4.7 Test Program for XL9535 Relay Board

```
#include <Wire.h>

void setup(void) {
  Serial.begin(115200);
  while (!Serial)
    delay(10);
  Serial.println("XL9535 Test");
  Wire.begin();
  Wire.beginTransmission(0x20);
  Wire.write(6);    // Configuration register 0
  Wire.write(0);    // all pins are outputs
  Wire.endTransmission();
  Wire.beginTransmission(0x20);
  Wire.write(7);    // Configuration register 1
  Wire.write(0);    // all pins are outputs
  Wire.endTransmission();
}

int i = 0;

void loop()
{
  Wire.beginTransmission(0x20);
  Wire.write(2);     // Output register 0
  Wire.write(i % 256);
  Wire.endTransmission();
  Wire.beginTransmission(0x20);
  Wire.write(3);     // Output register 1
  Wire.write(i / 256);
  Wire.endTransmission();
  if (++i == 65536)
    i = 0;
  delay(50);           // wait 0.05 seconds
}
```

4.8 freeMemory()

What `freeMemory()` is actually reporting is the space between the heap and the stack. It does not report any de-allocated memory that is buried in the heap. Buried heap space is not usable by the stack, and may be fragmented enough that it is not usable for many heap allocations either. The space between the heap and the stack is what you really need to monitor if you are trying to avoid stack crashes.

Found here: https://learn.adafruit.com/memories-of-an-arduino/measuring-free-memory

```
#ifdef __arm__
// should use uinstd.h to define sbrk but Due causes a conflict
extern "C" char* sbrk(int incr);
#else  // __ARM__
extern char *__brkval;
#endif  // __arm__

int freeMemory()
{
  char top;
#ifdef __arm__
  return &top - reinterpret_cast<char*>(sbrk(0));
#elif defined(CORE_TEENSY) || (ARDUINO > 103 && ARDUINO != 151)
  return &top - __brkval;
#else  // __arm__
  return __brkval ? &top - __brkval : &top - __malloc_heap_start;
#endif  // __arm__
}
```

4.9 Seeed XIAO SAMD21 Expansion Board

Expansion board: https://wiki.seeedstudio.com/Seeeduino-XIAO-Expansion-Board/

How to install the OLED library: https://wiki.seeedstudio.com/How_to_install_Arduino_Library/

Library for OLED display: https://github.com/olikraus/U8g2_Arduino

u8g2 Reference Manual: https://github.com/olikraus/u8g2/wiki/u8g2reference

u8x8 Reference Manual: https://github.com/olikraus/u8g2/wiki/u8x8reference

Wiki: https://github.com/olikraus/u8g2/wiki

Schematic diagram: https://files.seeedstudio.com/wiki/Seeeduino-XIAO-Expansion-Board/document/Seeeduino%20XIAO%20Expansion%20board_v1.0_SCH_200824.pdf

Left button: GND to RESET

Right button: GND to D1

U8g2 also includes U8x8 library. Features for U8g2 and U8x8 are:

- **U8g2**
  - **Includes all graphics procedures (line/box/circle draw).**
  - **Supports many fonts. (Almost) no restriction on the font height.**
  - **Requires some memory in the microcontroller to render the display.**
- **U8x8**
  - **Text output only (character) device.**
  - **Only fonts allowed with fixed size per character (8x8 pixel).**
  - **Writes directly to the display. No buffer in the microcontroller required.**

Test program for 128x64 display:

```
#include <U8x8lib.h>

uint8_t graphics[768];  // Memory for 128x48 graphics, leaving 2 lines of text at the bottom

U8X8_SSD1306_128X64_NONAME_HW_I2C u8x8(/* clock=*/ PIN_WIRE_SCL, /* data=*/ PIN_WIRE_SDA, /* reset=*/ U8X8_PIN_NONE);

void setup(void)
{
  u8x8.begin();
  u8x8.setFlipMode(0);  // set number from 0 to 3, the screen word will rotary 180
  u8x8.setFont(u8x8_font_chroma48medium8_r);
  u8x8.setCursor(0, 6);
  u8x8.print("Line 6 .........");
  u8x8.setCursor(0, 7);
  u8x8.print("Line 7 .........");

  for (int x = 0; x < 128; x++)     // plot a sine wave
    setpixel(x,24 - 22*sin((float)x / 128 * 4 * 3.1415));

  printgraphics();
}

void setpixel(int x, int y)
{
  graphics[x + 128 * (y / 8)] |= 1 << (y % 8);
}

void printgraphics()
{
  for (int y = 0; y < 6; y++)
    u8x8.drawTile(0, y, 16, graphics + 128 * y);
}

void loop(void)
{
}
```

SD Library: https://www.arduino.cc/reference/en/libraries/sd/

The SD library supports FAT16 and FAT32 file systems on standard SD cards and SDHC cards. It uses short 8.3 names for files. The file names passed to the SD library functions can include paths separated by forward-slashes, /, e.g. “directory/filename.txt”. Because the working directory is always the root of the SD card, a name refers to the same file whether or not it includes a leading slash (e.g. “/file.txt” is equivalent to “file.txt”).

Example for listing the directory of a SD card: https://docs.arduino.cc/learn/programming/sd-guide/

Test program for reading a file from SD card:

```
#include <SD.h>

File myFile;

void setup(void)
{
  // Open serial communications and wait for port to open:
  Serial.begin(115200);
  while(!Serial);              // Execute after turning on the serial monitor
  delay(500);

  Serial.print("Initializing SD card...");

  pinMode(D2, OUTPUT);          // Modify the pins here to fit the CS pins of the SD card you are using.
  if (!SD.begin(D2))
  {
    Serial.println("initialization failed!");
    return;
  }
  Serial.println("initialization done.");

  // open the file for reading:
  myFile = SD.open("xx.kml");
  if (myFile)
  {
    while (myFile.available())
    {
      Serial.write(myFile.read());
    }
    myFile.close();
  }
  else
  {
    // if the file didn't open, print an error:
    Serial.println("error opening test.txt");
  }
}

void loop(void)
{
}
```

Test program for listing the directory of the SD card:

```
#include <SD.h>

File root;

void setup(void)
{
  // Open serial communications and wait for port to open:
  Serial.begin(115200);
  while(!Serial);              // Execute after turning on the serial monitor
  delay(500);

  Serial.print("Initializing SD card...");

  pinMode(D2, OUTPUT);          // Modify the pins here to fit the CS pins of the SD card you are using.
  if (!SD.begin(D2))
  {
    Serial.println("initialization failed!");
    return;
  }
  Serial.println("initialization done.");

  root = SD.open("/");
  printDirectory(root, 0);
  Serial.println("done!");
}

void loop(void)
{
}

void printDirectory(File dir, int numTabs)
{
  while (true)
  {
    File entry =  dir.openNextFile();
    if (! entry)
    {
       // no more files
       break;
    }
```

```
    for (uint8_t i = 0; i < numTabs; i++)
    {
      Serial.print('\t');
    }

    Serial.print(entry.name());

    if (entry.isDirectory())
    {
      Serial.println("/");
      printDirectory(entry, numTabs + 1);
    }
    else
    {
      // files have sizes, directories do not
      Serial.print("\t\t");
      Serial.println(entry.size(), DEC);
    }
    entry.close();
  }
}
```

Test program for scrolling through all *.KML files with the pushbutton:

```
#include <SD.h>

File root;
const int buttonPin = 1;     // the number of the pushbutton pin

void setup(void)
{
  pinMode(buttonPin, INPUT_PULLUP);

  // Open serial communications and wait for port to open:
  Serial.begin(115200);
  while(!Serial);              // Execute after turning on the serial monitor
  delay(500);

  Serial.print("Initializing SD card...");
```

```
  pinMode(D2, OUTPUT);          // Modify the pins here to fit the CS pins of the SD card you are using.
  if (!SD.begin(D2))
  {
    Serial.println("initialization failed!");
    return;
  }
  Serial.println("initialization done.");

  root = SD.open("/");
}

void loop(void)
{
    String s;
  while (true)
  {
    File entry =  root.openNextFile();
    if (! entry)
    {
      root.rewindDirectory();
      entry =  root.openNextFile();
    }
    s = entry.name();
    if (s.endsWith(".KML"))
    {
      Serial.println(s);
      while (digitalRead(buttonPin) == LOW);   // wait for button release
      delay(100);
      while(digitalRead(buttonPin) == HIGH);   // wait for button press
    }
    entry.close();
  }
}
```

4.10 XIAO e-Ink Expansion Board

https://wiki.seeedstudio.com/XIAO-eInk-Expansion-Board/

2.9" Monochrome e-Ink display: https://www.seeedstudio.com/2-9-Monochrome-ePaper-Display-with-296x128-Pixels-p-5782.html

Demo code can be found here: https://github.com/peterpanstechland/e-ink

The demo program works, but it's quite useless. It's unclear how different text can be shown on the display.

The file "demo.h" contains the bitmap of the image.

4.11 PZEM-0004T-V3 Power Analyzer with Seeed XIAO SAMD21

PZEM-004T AC 100A Digital Ampere / Volt/ Watt measurement module with RS485

Available as a 10A module (with internal shunt) or as a 100A module (with current clamp, either open or closed).

The current clamp is a 1000:1 current transformer and the output load must be 10 Ohm or less.

In this module the isolation is made by optocouplers between the microcontroller and the RS485 output. The microcontroller is on mains voltage level, and it's powered by a capacitor from the mains voltage (seems risky if the mains voltage contains spikes). But you also need 5V supply at the output side, for the optocoupler. It can be modified to 3.3V by changing a resistor.

https://www.aliexpress.com/item/1005004852373322.html

Arduino library: https://github.com/mandulaj/PZEM-004T-v30

Unfortunately, programming is very complicated because a CRC check is required for all data transmissions. You have to rely on the Arduino Library, without understanding how CRC works.

Wiki: https://github.com/olehs/PZEM004T/wiki

This page contains useful instructions how to test the module with a PC software: https://www.nn-digital.com/en/blog/2019/11/04/example-of-the-pzem-004t-v3-v3-0-interfacing-program-using-arduino/
Extract the files to a folder, then start "run.bat" as administrator, which will register a library. Then run "PZEM004T-Master.exe". This test was successful.

The Arduino library mentioned in the above website is not suitable for SAMD21 microcontroller.
A modified library for SAMD21 exists here: https://github.com/zygisjas/PZEM-004T-V30-SAMD21/tree/main
But I didn't succeed to install it in Arduino IDE.

Connections:

Seeed XIAO SAMD21	PZEM-0004T-V3	Notes
GND Pin 13	GND black	
RX Pin 8	TX yellow or white	
TX Pin 7	RX green	It might be required to change a resistor for the optocoupler, if the supply voltage is 3.3V
3V3 Pin 12	VCC red	

The following code is based on the code which I found here: https://github.com/zygisjas/PZEM-004T-V30-SAMD21

But I did heavily modify and simplify it.

```
float voltage, current, power;

bool sendCmd8(uint8_t cmd, uint16_t rAddr, uint16_t val){
    uint8_t sendBuffer[8]; // Send buffer

    sendBuffer[0] = 0xF8;                   // Set slave address
    sendBuffer[1] = cmd;                    // Set command
    sendBuffer[2] = (rAddr >> 8) & 0xFF;    // Set high byte of register address
    sendBuffer[3] = (rAddr) & 0xFF;         // Set low byte =//=
    sendBuffer[4] = (val >> 8) & 0xFF;      // Set high byte of register value
    sendBuffer[5] = (val) & 0xFF;           // Set low byte =//=
    setCRC(sendBuffer, 8);                  // Set CRC of frame
    Serial1.write(sendBuffer, 8); // send frame

    Serial.print("   Written: ");
    for(int i = 0; i < 8; i++)
    {
      if(sendBuffer[i] < 16) Serial.print("0");
      Serial.print(sendBuffer[i], HEX);
      Serial.print(" ");
    }
    return true;
}

bool updateValues()
{
    static uint8_t response[15];

    sendCmd8(0x04, 0x00, 0x05);

    if(recieve(response, 15) != 15)
        return false;

    Serial.print("   Received: ");
    for(int i = 0; i < 15; i++)
    {
      if(response[i] < 16) Serial.print("0");
      Serial.print(response[i], HEX);
      Serial.print(" ");
    }
```

```
    voltage = (float)((uint32_t)response[3] << 8 | (uint32_t)response[4]) / 10.0;
    current = (float)((uint32_t)response[5] << 8 | (uint32_t)response[6] |
                      (uint32_t)response[7] << 24 | (uint32_t)response[8] << 16) / 1000.0;
    power = (float)   ((uint32_t)response[9] << 8 | (uint32_t)response[10] |
                     (uint32_t)response[11] << 24 | (uint32_t)response[12] << 16) / 10.0;
    return true;
}

uint16_t recieve(uint8_t *resp, uint16_t len)
{
    unsigned long startTime = millis(); // Start time for Timeout
    uint8_t index = 0; // Bytes we have read
    while((index < len) && (millis() - startTime < 100))
    {
        if(Serial1.available() > 0)
        {
            uint8_t c = (uint8_t)Serial1.read();
            resp[index++] = c;
        }
    }

    // Check CRC with the number of bytes read
    if(!checkCRC(resp, index))
        return 0;
    return index;
}

bool checkCRC(const uint8_t *buf, uint16_t len)
{
    if(len <= 2) // Sanity check
        return false;
    uint16_t crc = CRC16(buf, len - 2); // Compute CRC of data
    return ((uint16_t)buf[len-2]  | (uint16_t)buf[len-1] << 8) == crc;
}

void setCRC(uint8_t *buf, uint16_t len)
{
    if(len <= 2) // Sanity check
        return;
    uint16_t crc = CRC16(buf, len - 2); // CRC of data
    // Write high and low byte to last two positions
    buf[len - 2] = crc & 0xFF; // Low byte first
```

```
    buf[len - 1] = (crc >> 8) & 0xFF; // High byte second
}

static const uint16_t crcTable[] PROGMEM = {
    0X0000, 0XC0C1, 0XC181, 0X0140, 0XC301, 0X03C0, 0X0280, 0XC241,
    0XC601, 0X06C0, 0X0780, 0XC741, 0X0500, 0XC5C1, 0XC481, 0X0440,
    0XCC01, 0X0CC0, 0X0D80, 0XCD41, 0X0F00, 0XCFC1, 0XCE81, 0X0E40,
    0X0A00, 0XCAC1, 0XCB81, 0X0B40, 0XC901, 0X09C0, 0X0880, 0XC841,
    0XD801, 0X18C0, 0X1980, 0XD941, 0X1B00, 0XDBC1, 0XDA81, 0X1A40,
    0X1E00, 0XDEC1, 0XDF81, 0X1F40, 0XDD01, 0X1DC0, 0X1C80, 0XDC41,
    0X1400, 0XD4C1, 0XD581, 0X1540, 0XD701, 0X17C0, 0X1680, 0XD641,
    0XD201, 0X12C0, 0X1380, 0XD341, 0X1100, 0XD1C1, 0XD081, 0X1040,
    0XF001, 0X30C0, 0X3180, 0XF141, 0X3300, 0XF3C1, 0XF281, 0X3240,
    0X3600, 0XF6C1, 0XF781, 0X3740, 0XF501, 0X35C0, 0X3480, 0XF441,
    0X3C00, 0XFCC1, 0XFD81, 0X3D40, 0XFF01, 0X3FC0, 0X3E80, 0XFE41,
    0XFA01, 0X3AC0, 0X3B80, 0XFB41, 0X3900, 0XF9C1, 0XF881, 0X3840,
    0X2800, 0XE8C1, 0XE981, 0X2940, 0XEB01, 0X2BC0, 0X2A80, 0XEA41,
    0XEE01, 0X2EC0, 0X2F80, 0XEF41, 0X2D00, 0XEDC1, 0XEC81, 0X2C40,
    0XE401, 0X24C0, 0X2580, 0XE541, 0X2700, 0XE7C1, 0XE681, 0X2640,
    0X2200, 0XE2C1, 0XE381, 0X2340, 0XE101, 0X21C0, 0X2080, 0XE041,
    0XA001, 0X60C0, 0X6180, 0XA141, 0X6300, 0XA3C1, 0XA281, 0X6240,
    0X6600, 0XA6C1, 0XA781, 0X6740, 0XA501, 0X65C0, 0X6480, 0XA441,
    0X6C00, 0XACC1, 0XAD81, 0X6D40, 0XAF01, 0X6FC0, 0X6E80, 0XAE41,
    0XAA01, 0X6AC0, 0X6B80, 0XAB41, 0X6900, 0XA9C1, 0XA881, 0X6840,
    0X7800, 0XB8C1, 0XB981, 0X7940, 0XBB01, 0X7BC0, 0X7A80, 0XBA41,
    0XBE01, 0X7EC0, 0X7F80, 0XBF41, 0X7D00, 0XBDC1, 0XBC81, 0X7C40,
    0XB401, 0X74C0, 0X7580, 0XB541, 0X7700, 0XB7C1, 0XB681, 0X7640,
    0X7200, 0XB2C1, 0XB381, 0X7340, 0XB101, 0X71C0, 0X7080, 0XB041,
    0X5000, 0X90C1, 0X9181, 0X5140, 0X9301, 0X53C0, 0X5280, 0X9241,
    0X9601, 0X56C0, 0X5780, 0X9741, 0X5500, 0X95C1, 0X9481, 0X5440,
    0X9C01, 0X5CC0, 0X5D80, 0X9D41, 0X5F00, 0X9FC1, 0X9E81, 0X5E40,
    0X5A00, 0X9AC1, 0X9B81, 0X5B40, 0X9901, 0X59C0, 0X5880, 0X9841,
    0X8801, 0X48C0, 0X4980, 0X8941, 0X4B00, 0X8BC1, 0X8A81, 0X4A40,
    0X4E00, 0X8EC1, 0X8F81, 0X4F40, 0X8D01, 0X4DC0, 0X4C80, 0X8C41,
    0X4400, 0X84C1, 0X8581, 0X4540, 0X8701, 0X47C0, 0X4680, 0X8641,
    0X8201, 0X42C0, 0X4380, 0X8341, 0X4100, 0X81C1, 0X8081, 0X4040
};

uint16_t CRC16(const uint8_t *data, uint16_t len)
{
    uint8_t nTemp; // CRC table index
    uint16_t crc = 0xFFFF; // Default value
```

```
    while (len--)
    {
        nTemp = *data++ ^ crc;
        crc >>= 8;
        crc ^= (uint16_t)pgm_read_word(&crcTable[nTemp]);
    }
    return crc;
}

void setup(void)
{
  Serial1.begin(9600);    // PZEM Module
  Serial.begin(38400);    // USB
  while (!Serial);
  Serial.println("Start");
  delay(100);
}

void loop()
{
  updateValues();
  Serial.println(String("   Data: ") + voltage + "V  " + current + "A  " + power + "W");
  delay(1000);
}
```

In the next chapter is described how the precomputed CRC table can be replaced by CRC computed at runtime.

4.12 JSY1003F Power Analyzer with Seeed XIAO SAMD21

JSY1003F Single Phase Embedded AC Acquisition Module

In this module the isolation is between the mains voltage and the microcontroller.
The microcontroller is powered by an external 3.3V source. Much better solution!

Data sheet: https://www.jsypowermeter.com/uploads/JSY1003F-User-Manual1.pdf

https://www.jsypowermeter.com/jsy1003f-single-phase-ac-ttl-modbus-rtu-embedded-metering-module-product/

https://de.aliexpress.com/item/4000340221120.html

The data protocol is different from the PZEM-0004T-V3 module: Different module adress and different register adresses. The CRC calculation algorithm is the same as in PZEM-0004T-V3 module. I didn't yet find a library for JSY1003F.

The ADC chip is a RN8208G, which has a programmable gain in front of the converter. The datasheet seems to be available only in chinese:
https://www.lcsc.com/datasheet/lcsc_datasheet_2111261730_RENERGY-RN8208G_C2925739.pdf

Pinout from left to right: L nc nc N CT+ CT- GND 3V3 RX TX PF IRQ (CT+ and CT- have no pins, PF and IRQ are not required)

Connections:

Seeed XIAO SAMD21	JSY1003F
GND Pin 13	GND black
RX Pin 8	TX yellow or white
TX Pin 7	RX green
3V3 Pin 12	VCC red

This is the software:

```
float voltage, current, power;

bool sendCmd8(uint8_t cmd, uint16_t rAddr, uint16_t val)
{
  uint8_t sendBuffer[8]; // Send buffer

  sendBuffer[0] = 0x01;                    // Set slave address
  sendBuffer[1] = cmd;                     // Set command
  sendBuffer[2] = (rAddr >> 8) & 0xFF;     // Set high byte of register address
  sendBuffer[3] = (rAddr) & 0xFF;          // Set low byte =//=
  sendBuffer[4] = (val >> 8) & 0xFF;       // Set high byte of register value
  sendBuffer[5] = (val) & 0xFF;            // Set low byte =//=
  setCRC(sendBuffer, 8);                   // Set CRC of frame
  Serial1.write(sendBuffer, 8); // send frame

  Serial.print("   Written: ");
  for(int i = 0; i < 8; i++)
  {
    if(sendBuffer[i] < 16) Serial.print("0");
    Serial.print(sendBuffer[i], HEX);
    Serial.print(" ");
  }

  return true;
}

bool updateValues()
{
  static uint8_t response[11];

  // Read 3 registers starting at 0x00
  sendCmd8(0x03, 0x48, 0x03);

  if(recieve(response, 11) != 11)
    return false;

  Serial.print("   Received: ");
  for(int i = 0; i < 11; i++)
  {
    if(response[i] < 16) Serial.print("0");
    Serial.print(response[i], HEX);
```

```
    Serial.print(" ");
  }

  voltage = (float)((uint32_t)response[3] << 8 | (uint32_t)response[4]) / 100.0;
  current = (float)((uint32_t)response[5] << 8 | (uint32_t)response[6]) / 100.0;
  power = (float)  ((uint32_t)response[7] << 8 | (uint32_t)response[8]);
  return true;
}

uint16_t recieve(uint8_t *resp, uint16_t len)
{
  unsigned long startTime = millis(); // Start time for Timeout
  uint8_t index = 0; // Bytes we have read
  while((index < len) && (millis() - startTime < 100))
  {
    if(Serial1.available() > 0)
    {
      uint8_t c = (uint8_t)Serial1.read();
      resp[index++] = c;
    }
  }

  // Check CRC with the number of bytes read
  if(!checkCRC(resp, index))
    return 0;
  return index;
}

bool checkCRC(const uint8_t *buf, uint16_t len)
{
  if(len <= 2) // Sanity check
    return false;
  uint16_t crc = CRC16(buf, len - 2); // Compute CRC of data
  return ((uint16_t)buf[len-2]  | (uint16_t)buf[len-1] << 8) == crc;
}

void setCRC(uint8_t *buf, uint16_t len)
{
  if(len <= 2) // Sanity check
    return;
  uint16_t crc = CRC16(buf, len - 2); // CRC of data
  // Write high and low byte to last two positions
  buf[len - 2] = crc & 0xFF; // Low byte first
```

```
  buf[len - 1] = (crc >> 8) & 0xFF; // High byte second
}

static const uint16_t crcTable[] PROGMEM = {
  0X0000, 0XC0C1, 0XC181, 0X0140, 0XC301, 0X03C0, 0X0280, 0XC241,
  0XC601, 0X06C0, 0X0780, 0XC741, 0X0500, 0XC5C1, 0XC481, 0X0440,
  0XCC01, 0X0CC0, 0X0D80, 0XCD41, 0X0F00, 0XCFC1, 0XCE81, 0X0E40,
  0X0A00, 0XCAC1, 0XCB81, 0X0B40, 0XC901, 0X09C0, 0X0880, 0XC841,
  0XD801, 0X18C0, 0X1980, 0XD941, 0X1B00, 0XDBC1, 0XDA81, 0X1A40,
  0X1E00, 0XDEC1, 0XDF81, 0X1F40, 0XDD01, 0X1DC0, 0X1C80, 0XDC41,
  0X1400, 0XD4C1, 0XD581, 0X1540, 0XD701, 0X17C0, 0X1680, 0XD641,
  0XD201, 0X12C0, 0X1380, 0XD341, 0X1100, 0XD1C1, 0XD081, 0X1040,
  0XF001, 0X30C0, 0X3180, 0XF141, 0X3300, 0XF3C1, 0XF281, 0X3240,
  0X3600, 0XF6C1, 0XF781, 0X3740, 0XF501, 0X35C0, 0X3480, 0XF441,
  0X3C00, 0XFCC1, 0XFD81, 0X3D40, 0XFF01, 0X3FC0, 0X3E80, 0XFE41,
  0XFA01, 0X3AC0, 0X3B80, 0XFB41, 0X3900, 0XF9C1, 0XF881, 0X3840,
  0X2800, 0XE8C1, 0XE981, 0X2940, 0XEB01, 0X2BC0, 0X2A80, 0XEA41,
  0XEE01, 0X2EC0, 0X2F80, 0XEF41, 0X2D00, 0XEDC1, 0XEC81, 0X2C40,
  0XE401, 0X24C0, 0X2580, 0XE541, 0X2700, 0XE7C1, 0XE681, 0X2640,
  0X2200, 0XE2C1, 0XE381, 0X2340, 0XE101, 0X21C0, 0X2080, 0XE041,
  0XA001, 0X60C0, 0X6180, 0XA141, 0X6300, 0XA3C1, 0XA281, 0X6240,
  0X6600, 0XA6C1, 0XA781, 0X6740, 0XA501, 0X65C0, 0X6480, 0XA441,
  0X6C00, 0XACC1, 0XAD81, 0X6D40, 0XAF01, 0X6FC0, 0X6E80, 0XAE41,
  0XAA01, 0X6AC0, 0X6B80, 0XAB41, 0X6900, 0XA9C1, 0XA881, 0X6840,
  0X7800, 0XB8C1, 0XB981, 0X7940, 0XBB01, 0X7BC0, 0X7A80, 0XBA41,
  0XBE01, 0X7EC0, 0X7F80, 0XBF41, 0X7D00, 0XBDC1, 0XBC81, 0X7C40,
  0XB401, 0X74C0, 0X7580, 0XB541, 0X7700, 0XB7C1, 0XB681, 0X7640,
  0X7200, 0XB2C1, 0XB381, 0X7340, 0XB101, 0X71C0, 0X7080, 0XB041,
  0X5000, 0X90C1, 0X9181, 0X5140, 0X9301, 0X53C0, 0X5280, 0X9241,
  0X9601, 0X56C0, 0X5780, 0X9741, 0X5500, 0X95C1, 0X9481, 0X5440,
  0X9C01, 0X5CC0, 0X5D80, 0X9D41, 0X5F00, 0X9FC1, 0X9E81, 0X5E40,
  0X5A00, 0X9AC1, 0X9B81, 0X5B40, 0X9901, 0X59C0, 0X5880, 0X9841,
  0X8801, 0X48C0, 0X4980, 0X8941, 0X4B00, 0X8BC1, 0X8A81, 0X4A40,
  0X4E00, 0X8EC1, 0X8F81, 0X4F40, 0X8D01, 0X4DC0, 0X4C80, 0X8C41,
  0X4400, 0X84C1, 0X8581, 0X4540, 0X8701, 0X47C0, 0X4680, 0X8641,
  0X8201, 0X42C0, 0X4380, 0X8341, 0X4100, 0X81C1, 0X8081, 0X4040
};

uint16_t CRC16(const uint8_t *data, uint16_t len)
{
  uint8_t nTemp; // CRC table index
  uint16_t crc = 0xFFFF; // Default value
```

```
  while (len--)
  {
    nTemp = *data++ ^ crc;
    crc >>= 8;
    crc ^= (uint16_t)pgm_read_word(&crcTable[nTemp]);
  }
  return crc;
}

void setup(void)
{
  Serial1.begin(9600);    // JSY Module
  Serial.begin(38400);    // USB
  while (!Serial);
  Serial.println("Start");
  delay(100);
}

void loop()
{
  updateValues();
  Serial.println(String("   Data: ") + voltage + "V  " + current + "A  " + power + "W");
  delay(1000);
}
```

This version is without precomputed CRC table:

```
float voltage, current, power;

bool sendCmd8(uint8_t cmd, uint16_t rAddr, uint16_t val)
{
  uint8_t sendBuffer[8]; // Send buffer

  sendBuffer[0] = 0x01;                   // Set slave address
  sendBuffer[1] = cmd;                    // Set command
  sendBuffer[2] = (rAddr >> 8) & 0xFF;    // Set high byte of register address
  sendBuffer[3] = (rAddr) & 0xFF;         // Set low byte =//=
  sendBuffer[4] = (val >> 8) & 0xFF;      // Set high byte of register value
  sendBuffer[5] = (val) & 0xFF;           // Set low byte =//=
  setCRC(sendBuffer, 8);                  // Set CRC of frame
  Serial1.write(sendBuffer, 8); // send frame

  Serial.print("   Written: ");
  for(int i = 0; i < 8; i++)
  {
    if(sendBuffer[i] < 16) Serial.print("0");
    Serial.print(sendBuffer[i], HEX);
    Serial.print(" ");
  }
  return true;
}

bool updateValues()
{
  static uint8_t response[11];

  // Read 3 registers starting at 0x00
  sendCmd8(0x03, 0x48, 0x03);

  if(recieve(response, 11) != 11)
    return false;

  Serial.print("   Received: ");
  for(int i = 0; i < 11; i++)
  {
    if(response[i] < 16) Serial.print("0");
    Serial.print(response[i], HEX);
    Serial.print(" ");
```

```
  }

  voltage = (float)((uint32_t)response[3] << 8 | (uint32_t)response[4]) / 100.0;
  current = (float)((uint32_t)response[5] << 8 | (uint32_t)response[6]) / 100.0;
  power = (float)  ((uint32_t)response[7] << 8 | (uint32_t)response[8]);
  return true;
}

uint16_t recieve(uint8_t *resp, uint16_t len)
{
  unsigned long startTime = millis(); // Start time for Timeout
  uint8_t index = 0; // Bytes we have read
  while((index < len) && (millis() - startTime < 100))
  {
    if(Serial1.available() > 0)
    {
      uint8_t c = (uint8_t)Serial1.read();
      resp[index++] = c;
    }
  }

  // Check CRC with the number of bytes read
  if(!checkCRC(resp, index))
    return 0;
  return index;
}

bool checkCRC(const uint8_t *buf, uint16_t len)
{
  if(len <= 2) // Sanity check
    return false;
  uint16_t crc = CRC16(buf, len - 2); // Compute CRC of data
  return ((uint16_t)buf[len-2] | (uint16_t)buf[len-1] << 8) == crc;
}

void setCRC(uint8_t *buf, uint16_t len)
{
  if(len <= 2) // Sanity check
    return;
  uint16_t crc = CRC16(buf, len - 2); // CRC of data
  // Write high and low byte to last two positions
  buf[len - 2] = crc & 0xFF; // Low byte first
  buf[len - 1] = (crc >> 8) & 0xFF; // High byte second
```

```
}

uint16_t CRC16(const uint8_t *data, uint16_t len)
{
  uint16_t crc = 0xFFFF; // Default value

  while (len--)
  {
    crc = crc ^ *data++;
    for(int i=0; i<8; i++)
    {
      if(crc&1)
      {
        crc = crc >> 1;
        crc = crc ^ 0xa001;
      }
      else
        crc = crc >> 1;
    }
  }
  return crc;
}

void setup(void)
{
  Serial1.begin(9600);    // JSY Module
  Serial.begin(38400);    // USB
  while (!Serial);
  Serial.println("Start");
  delay(100);
}

void loop()
{
  updateValues();
  Serial.println(String("   Data: ") + voltage + "V  " + current + "A  " + power + "W");
  delay(1000);
}
```

4.13 PT8211 2-channel audio DAC

Kit: https://www.pjrc.com/store/pt8211_kit.html

Software: https://forum.pjrc.com/threads/72446-PT8211-with-irregular-clocking-signals-(non-I2S)-from-T4

Data sheet: https://www.pjrc.com/store/pt8211.pdf

The maximum clock frequency is 18.4 MHz.

Write 16-bit values to both channels:

```
digitalWriteFast(WS,0);           // write to right channel
for (int i=15; i>=0; i--)
{
  digitalWriteFast(DIN, (right_value>>i) & 1);
  delayNanoseconds(HALFCLKns);  // 28ns
  digitalWriteFast(BCLK,1);
  delayNanoseconds(HALFCLKns);  // 28ns
  digitalWriteFast(BCLK,0);
}
digitalWriteFast(WS,1);           // write to left channel
for (int i=15; i>=0; i--)
{
  digitalWriteFast(DIN, (left_value>>i) & 1);
  delayNanoseconds(HALFCLKns);  // 28ns
  digitalWriteFast(BCLK,1);
  delayNanoseconds(HALFCLKns);  // 28ns
  digitalWriteFast(BCLK,0);
}
digitalWriteFast(WS,0);           // both analog outputs are updated
```

If you write only to the left channel, then the same signal comes out of both channels. If HALFCLKns is set to the lowest possible value 28, both output channels can be updated in about 1.5us:

```
digitalWriteFast(WS,1);           // write to left channel
for (int i=15; i>=0; i--)
{
  digitalWriteFast(DIN, (left_value>>i) & 1);
  delayNanoseconds(HALFCLKns);  // 28ns
  digitalWriteFast(BCLK,1);
  delayNanoseconds(HALFCLKns);  // 28ns
```

```
  digitalWriteFast(BCLK,0);
}
digitalWriteFast(WS,0);           // both analog outputs are updated
```

4.14 MIDI

MIDI hardware and library is described here: https://www.pjrc.com/teensy/td_libs_MIDI.html

5 Absolute Angle Encoders

I developed this special code in August 2004. It has the following properties:

- **Only one code track on the rotating part (not multiple code tracks as in Gray code)**
- **Multiple sensors are scanning the same code track, these sensors are evenly distributed along the code track**
- **Parallel output signals**
- **Two output signals do never toggle at the same time, like in Gray code**
- **Suitable for applications with very big code track diameter, where multiple code tracks would be impractical or too expensive**
- **Sensors can be optical, magnetic, or mechanical switches**
- **Resolution 2^n is not possible with n sensors, but the resolution of some codes is very close to the theoretical limit, especially if the number of sensors is odd.**
- **The code length (= resolution) is always an even number**

Here is an example of an encoder with 6 sensors and resolution 48 (360° / 48 = 7.5°):

```
Code track: 000011000000000000011111001110000111111111111110
Sensors:    S       S       S       S       S       S

Output states:
000001, 000011, 000111, 001111, 101111, 101011, 001011, 001010, 000010, 000110, 001110, 011110,
011111, 010111, 010110, 010100, 000100, 001100, 011100, 111100, 111110, 101110, 101100, 101000,
001000, 011000, 111000, 111001, 111101, 011101, 011001, 010001, 010000, 110000, 110001, 110011,
111011, 111010, 110010, 100010, 100000, 100001, 100011, 100111, 110111, 110101, 100101, 000101
```

You can see three important properties of the code:

- **Only one bit toggles from one state to the next state.**
- **Only one bit toggles from the last state to the first state.**
- **All states are different from each other.**

How to find a suitable code track?

Step 1: The code is listed in the below table: 1 3 7 15 47 43 11 10

Step 2: Convert the numbers into binary format:

```
 1 = 000001
 3 = 000011
 7 = 000111
15 = 001111
47 = 101111
43 = 101011
11 = 001011
10 = 001010
```

Step 3: Reading the digits column by column gives the 48 bit code track:

```
0.......0.......0.......0.......0.......1.......
.0.......0.......0.......0.......1.......1......
..0.......0.......0.......1.......1.......1.....
...0.......0.......1.......1.......1.......1....
....1.......0.......1.......1.......1.......1...
.....1.......0.......1.......0.......1.......1..
......0.......0.......1.......0.......1.......1.
.......0.......0.......1.......0.......1.......0
000011000000000000011111001110000111111111111110
```

Table with all known codes for up to 8 sensors:

Number of sensors	Code lenght L	360°/L	Code
3	6	60°	1 3
4	8	45°	1 3
5	10	36°	1 3
5	20	18°	1 3 11 10
5	30	12°	1 3 7 15 11 10
6	12	30°	1 3
6	24	15°	1 3 11 10
6	36	10°	1 3 7 15 11 10
6	48	7.5°	1 3 7 15 47 43 11 10
6	60	6°	impossible
7	14		1 3
7	28		1 3 11 10
7	42		1 3 7 15 11 10
7	56		1 3 7 5 21 23 19 18
7	70		1 3 7 5 13 15 11 27 19 18
7	84		1 3 7 5 13 15 11 27 19 83 82 18
7	98		1 3 7 5 13 15 11 27 19 23 87 83 82 18
7	112		1 3 7 5 13 15 11 27 19 23 31 63 47 43 42 34
7	126		1 3 7 5 13 15 11 27 19 23 31 29 61 63 59 43 42 34
8	16	22.5°	1 3
8	32	11.25°	1 3 11 10
8	48	7.5°	1 3 7 15 11 10
8	64	5.625°	1 3 7 5 21 23 19 18
8	80	4.5°	1 3 7 5 13 15 11 27 19 18
8	96	3.75°	1 3 7 5 13 15 11 27 19 83 82 18
8	112		1 3 7 5 13 15 11 27 19 23 87 71 70 66
8	128	2.8125°	1 3 7 5 13 15 11 27 19 23 21 149 145 147 146 18
8	144	2.5°	1 3 7 5 13 15 11 27 19 23 21 29 25 89 91 75 74 66
8	160	2.25°	1 3 7 5 13 15 11 27 19 23 21 29 25 57 59 43 107 75 74 66
8	176		1 3 7 5 13 15 11 27 19 23 21 29 25 57 59 43 47 111 79 75 74 66
8	192	1.875°	1 3 7 5 13 15 11 27 19 23 21 29 25 57 59 43 47 45 109 111 110 106 74 66
8	208		1 3 7 5 13 15 11 27 19 23 21 29 25 57 59 43 47 45 61 63 31 95 87 83 82 18
8	224		1 3 7 5 13 15 11 27 19 23 21 29 25 57 59 43 47 45 61 63 31 95 91 123 115 83 82 18
8	240	1.5°	impossible

Table with all known codes for 9 sensors:

Number of sensors	Code lenght L	360°/L	Code
9	18	20°	1 3
9	36	10°	1 3 11 10
9	54		1 3 7 15 11 10
9	72	5°	1 3 7 5 21 17 19 18
9	90	4°	1 3 7 5 13 9 11 43 35 34
9	108		1 3 7 5 13 9 11 15 47 39 35 34
9	126		1 3 7 5 13 9 11 15 31 23 19 51 35 34
9	144	2.5°	1 3 7 5 13 9 11 15 31 23 19 27 59 43 35 34
9	162		1 3 7 5 13 9 11 15 31 23 19 27 25 57 41 43 35 34
9	180	2°	1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 34
9	198		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 291 290 34
9	216		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 299 291 290 34
9	234		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 47 303 295 291 290 34
9	252		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 47 45 301 293 295 291 290 34
9	270		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 47 45 61 57 59 51 115 83 82 66
9	288	1.25°	1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 47 45 61 57 59 51 55 119 87 83 82 66
9	306		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 47 45 61 57 59 51 55 63 127 95 87 83 82 66
9	324		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 47 45 61 57 59 51 55 63 127 95 79 75 91 83 82 66
9	342		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 47 45 61 57 59 51 55 63 127 95 79 75 91 83 211 210 82 66
9	360	1°	1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 47 45 61 57 59 51 55 63 127 95 79 75 91 83 87 215 211 210 82 66
9	378		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 47 45 61 57 59 51 55 63 127 95 79 75 91 83 87 85 213 215 211 210 82 66
9	396		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 47 45 61 57 59 51 55 63 127 95 79 75 91 123 107 105 109 111 103 119 87 83 82 66
9	414		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 47 45 61 57 59 51 55 63 127 95 79 75 91 123 251 187 155 147 151 183 179 243 115 83 82 66
9	432		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 47 45 61 57 59 51 55 63 127 95 79 75 91 123 251 187 155 147 151 149 181 183 179 243 115 83 82 66
9	450	0.8°	1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 47 45 61 57 59 51 55 63 127 95 79 75 91 123 251 187 155 147 151 149 157 159 191 183 179 243 115 83 82 66
9	468		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 47 45 61 57 59 51 55 63 127 95 79 75 91 123 251 187 155 147 151 149 157 159 191 175 239 111 103 119 87 83 82 66
9	486		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 47 45 61 57 59 51 55 63 127 95 79 75 91 123 251 187 155 147 151 149 157 159 191 175 171 235 107 111 103 119 87 83 82 66
9	504		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 47 45 61 57 59 51 55 63 127 95 79 75 91 123 251 187 155 147 151 149 157 159 191 255 239 111 107 235 171 427 299 315 307 306 290 34

Table with all known codes for 10 sensors:

Number of sensors	Code lenght L	360°/L	Code
10	20	18°	1 3
10	40	9°	1 3 11 10
10	60	6°	1 3 7 15 11 10
10	80	4.5°	1 3 7 5 21 17 19 18
10	100	3.6°	1 3 7 5 13 9 11 43 35 34
10	120	3°	1 3 7 5 13 9 11 15 47 39 35 34
10	140		1 3 7 5 13 9 11 15 31 23 19 51 35 34
10	160	2.25°	1 3 7 5 13 9 11 15 31 23 19 27 59 43 35 34
10	180	2°	1 3 7 5 13 9 11 15 31 23 19 27 25 57 41 43 35 34
10	200	1.8°	1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 34
10	220		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 163 162 34
10	240	1.5°	1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 171 139 138 130
10	260		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 169 137 139 138 130
10	280		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 111 103 102 98 34
10	300	1.2°	1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 115 114 98 34
10	320	1.125°	1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 177 179 147 146 130
10	340		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 187 155 139 138 130
10	360	1°	1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 203 139 138 130
10	380		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 201 203 139 138 130
10	400	0.9°	1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 207 143 139 138 130
10	420		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 199 207 143 139 138 130
10	440		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 243 179 163 162 34
10	460		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 119 247 183 167 163 162 34
10	480	0.75°	1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 119 103 111 107 235 171 139 138 130
10	500	0.72°	1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 119 103 101 109 105 107 235 171 139 138 130
10	520		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 119 103 101 109 105 107 111 239 175 143 139 138 130
10	540		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 119 103 101 109 105 107 111 127 95 223 159 143 139 138 130

10	560		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 119 103 101 109 105 107 111 127 95 93 221 157 153 155 139 138 130
10	580		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 119 103 101 109 105 107 111 127 95 93 85 117 245 181 183 179 178 162 34
10	600	0.6°	1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 119 103 101 109 105 107 111 127 95 93 85 117 125 61 189 157 153 155 139 138 130
10	620		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 119 103 101 109 105 107 111 127 95 93 85 117 125 61 189 157 149 151 147 155 139 138 130
10	640	0.5625°	1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 119 103 101 109 105 107 111 127 95 93 85 117 125 61 189 157 149 151 147 155 159 143 139 138 130
10	660		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 119 103 101 109 105 107 111 127 95 93 85 117 125 61 189 157 149 151 147 155 159 143 175 171 139 138 130
10	680		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 119 103 101 109 105 107 111 127 95 93 85 117 125 61 189 157 149 151 147 155 159 143 175 167 183 179 178 162 34
10	700		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 119 103 101 109 105 107 111 127 95 93 85 117 125 61 189 157 149 151 147 155 159 143 175 167 183 179 187 171 139 138 130
10	720	0.5°	1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 119 103 101 109 105 107 111 127 95 93 85 117 125 61 189 157 149 151 147 155 159 143 175 167 183 179 187 251 235 171 139 138 130
10	740		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 119 103 101 109 105 107 111 127 95 93 85 117 125 61 189 157 149 151 147 155 159 143 175 167 183 179 187 191 255 239 235 171 139 138 130
10	760		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 119 103 101 109 105 107 111 127 95 93 85 117 125 61 189 157 149 151 147 155 159 143 175 167 183 179 187 191 255 223 207 239 235 171 139 138 130
10	780		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 119 103 101 109 105 107 111 127 95 93 85 117 125 61 189 157 149 151 147 155 159 143 175 167 183 179 187 191 255 223 207 205 237 239 235 171 139 138 130
10	800	0.45°	1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 119 103 101 109 105 107 111 127 95 93 85 117 125 61 189 157 149 151 147 155 159 143 175 167 183 179 187 191 255 223 207 205 221 253 237 239 235 171 139 138 130
10	820		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 119 103 101 109 105 107 111 127 95 93 85 117 125 61 189 157 149 151 147 155 159 143 175 167 183 179 187 191 255 223 207 205 221 213 245 253 237 239 235 171 139 138 130
10	840		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 119 103 101 109 105 107 111 127 95 93 85 117 125 61 189 157 149 151 147 155 159 143 175 167 183 179 187 191 255 223 207 205 221 213 215 247 245 253 237 239 235 171 139 138 130
10	860		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 119 103 101 109 105 107 111 127 95 93 85 117 125 61 189 157 149 151 147 155 159 143 175 167 183 179 187 191 255 223 207 205 221 213 215 247 245 253 237 239 235 491 427 171 139 138 130
10	880		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 119 103 101 109 105 107 111 127 95 93 85 117 125 61 189 157 149 151 147 155 159 143 175 167 183 179 187 191 255 223 207 205 221 213 215 247 245 181 437 439 503 511 507 251 235 171 139 138 130

10	900	0.4°	1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 119 103 101 109 105 107 111 127 95 93 85 117 125 61 189 157 149 151 147 155 159 143 175 167 183 179 187 191 255 223 207 205 221 213 215 247 245 181 437 421 423 439 503 511 507 251 235 171 139 138 130
10	920		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 119 103 101 109 105 107 111 127 95 93 85 117 125 61 189 157 149 151 147 155 159 143 175 167 183 179 187 191 255 223 207 205 221 213 215 247 245 181 437 421 423 431 447 511 479 475 219 251 235 171 139 138 130
10	940		1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 119 103 101 109 105 107 111 127 95 93 85 117 125 61 189 157 149 151 147 155 159 143 175 167 183 179 187 191 255 223 207 205 221 213 215 247 245 181 437 421 423 431 447 319 383 351 479 475 219 251 235 171 139 138 130
10	960	0.375°	1 3 7 5 13 9 11 15 31 23 19 27 25 29 21 53 37 39 35 43 41 45 47 63 55 51 49 57 59 123 91 75 73 77 79 71 87 83 115 119 103 101 109 105 107 111 127 95 93 85 117 125 61 189 157 149 151 147 155 159 143 175 167 183 179 187 191 255 223 207 205 221 213 215 247 245 181 437 421 423 431 447 319 383 351 343 471 479 475 219 251 235 171 139 138 130
10	980		impossible

Index