

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

Collection of examples and useful hints

Michael Koch

Version from January 28, 2024

Content

1	Integrated Development Environment	3
1.1	IDE Software installation	3
1.2	Assembly code listing	3
1.3	CPU Speed	3
2	Functions, alphabetically sorted	4
2.1	analogReadResolution()	4
2.2	analogReference()	4
2.3	asm volatile("nop;")	4
2.4	delay()	4
2.5	delayMicroseconds()	5
2.6	delayNanoseconds()	5
2.7	digitalRead()	6
2.8	digitalWrite()	7
2.9	digitalWriteFast()	7
2.10	IntervalTimer	8
2.11	pinmode()	9
2.12	random()	9
2.13	Read multiple pins simultaneously	10
2.14	Serial Monitor and Serial Plotter	14
3	Projects	16
3.1	Teensy LC as an accessory for THAT analog computer	16
	3.2 Summary of available software for THAT analog computer	17
	Look-up-table	17
	Look-Up-Table with Derivative	20
	Delay line	24
	Direct digital synthesis	25
	Voltage controlled direct digital synthesis	27
	Triggered signal generator	29
	Band-limited noise generator	32
	Band-limited noise generator with a control voltage input	34
	3.3 Synthesizer with Teensy 4.0	36
	3.4 Synthesizer with Teensy 4.0, V2	38
	3.5 New Synthesizer Board	42
	3.6 RGB LEDs and LED Cubes	48
	4 Hardware	57
	4.1 Teensy 2.0	57
	4.2 Teensy LC	57
	4.3 Teensy 4.0	57
	4.4 Seeed Studio XIAO SAMD21	58
	4.5 PT8211 2-channel audio DAC	60
	4.6 MIDI	61

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

1.2 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\teensyavr.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>

2 Functions, alphabetically sorted

2.1 **analogReadResolution();**

Sets the resolution 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 of 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 at 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...

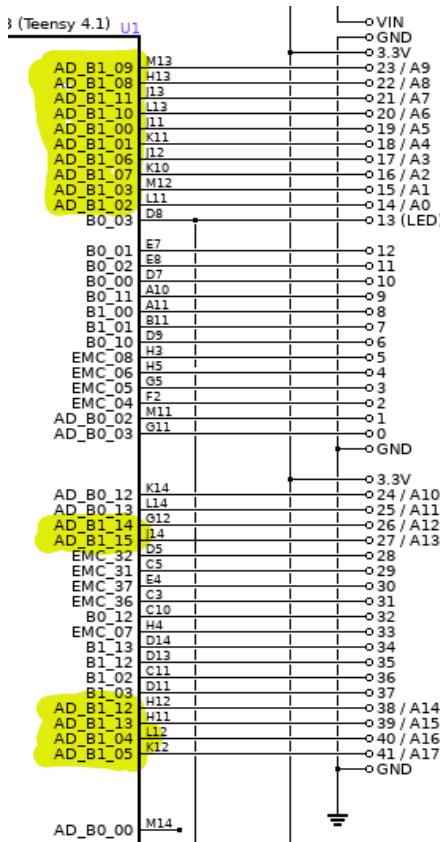
```
uint32_t *port_regs[] = {(uint32_t*)&IMXRT_GPIO6, (uint32_t*)&IMXRT_GPIO7, (uint32_t*)&IMXRT_GPIO8, (uint32_t*)&IMXRT_GPIO9};  
uint8_t pin_numbers[4][32];  
  
void setup() {  
    memset(pin_numbers, 0xff, sizeof(pin_numbers));  
    // put your setup code here, to run once:  
    for (uint8_t 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_regs[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

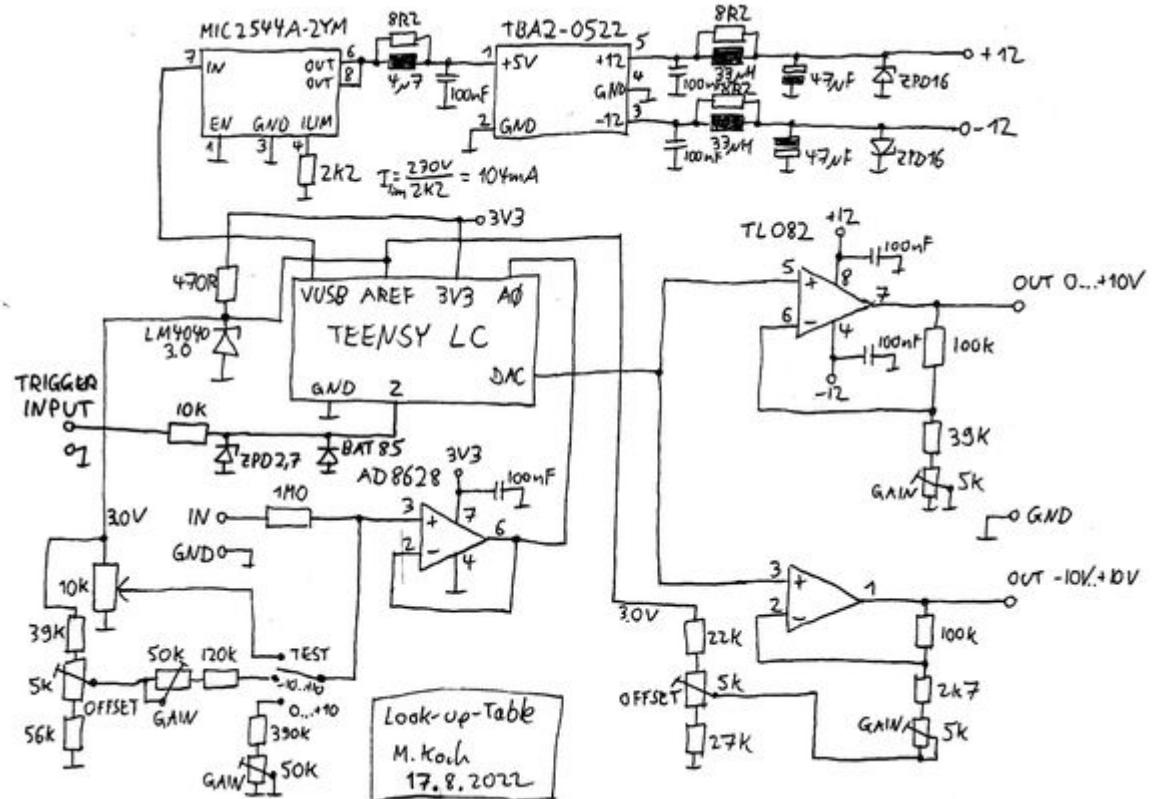
You will see the printout in the lower part of the screen.

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

➤ 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);
}
```

➤ Look-Up-Table with Derivative

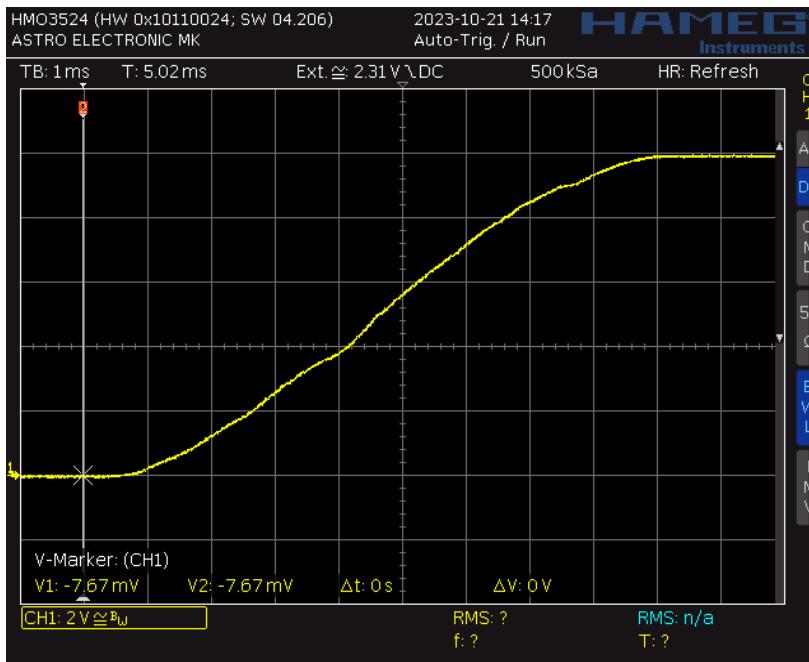
Here a user-defined table is used, which has less entries (351) than the final look-up-table (4096).

In this example the user-defined table contains the height profile of the Eichelkopf 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 or its derivative (which is the slope).

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

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



This is the slope of the Eichelkopf mountain, 50% slope is equivalent 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 Eichelkopf 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, 0.02, 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));
}
```

➤ 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
{}
```

➤ 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  
{  
}
```

➤ 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
{
}

```

➤ 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)               // in one-shot mode, the pointer stays at the last element
        x = 3599;
      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
{
}

```


➤ 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  
{  
}
```

➤ 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 - 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
{
}

```

3.3 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

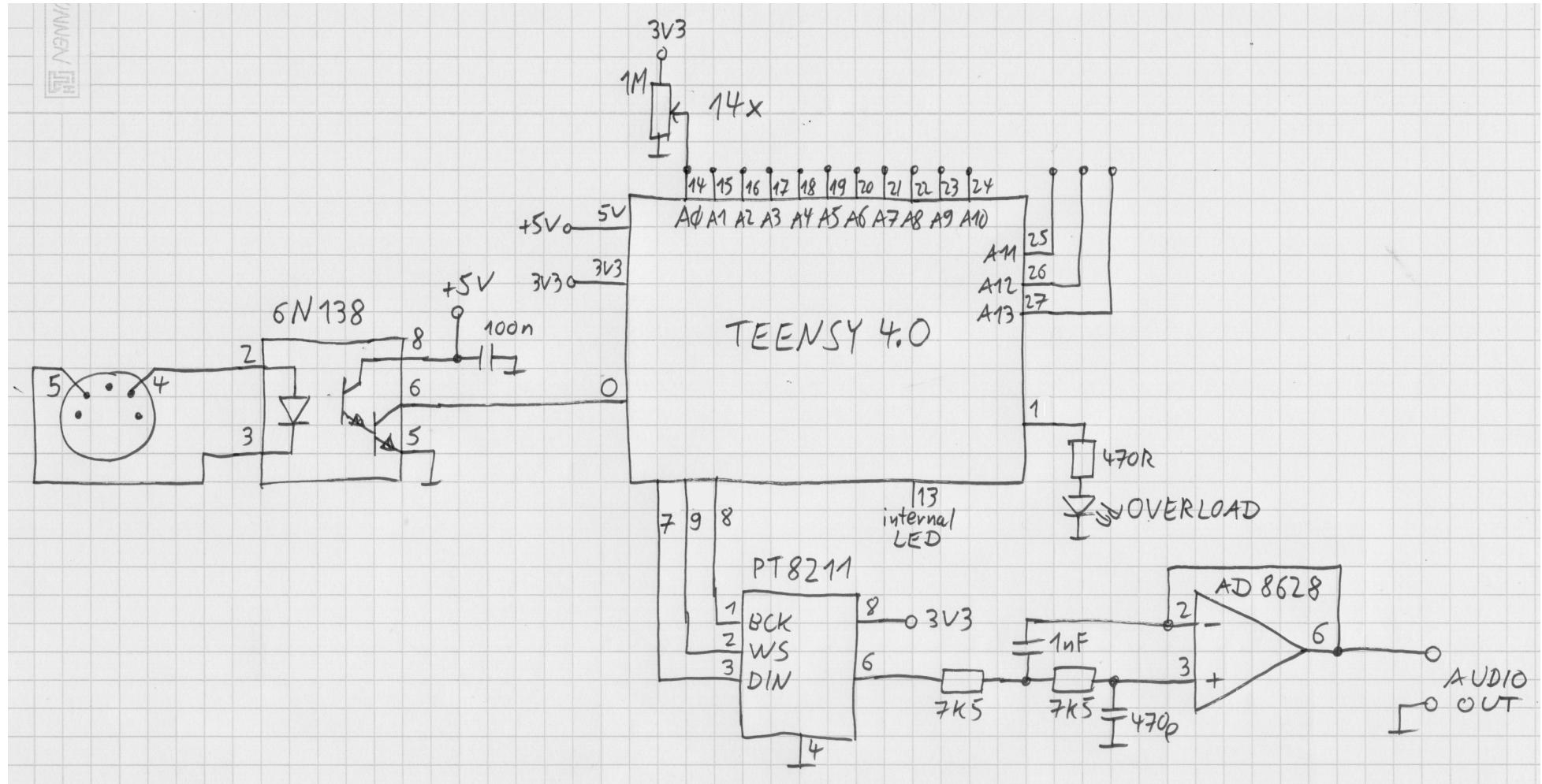
digitalWrite(led, LOW); // switch off the LED
}

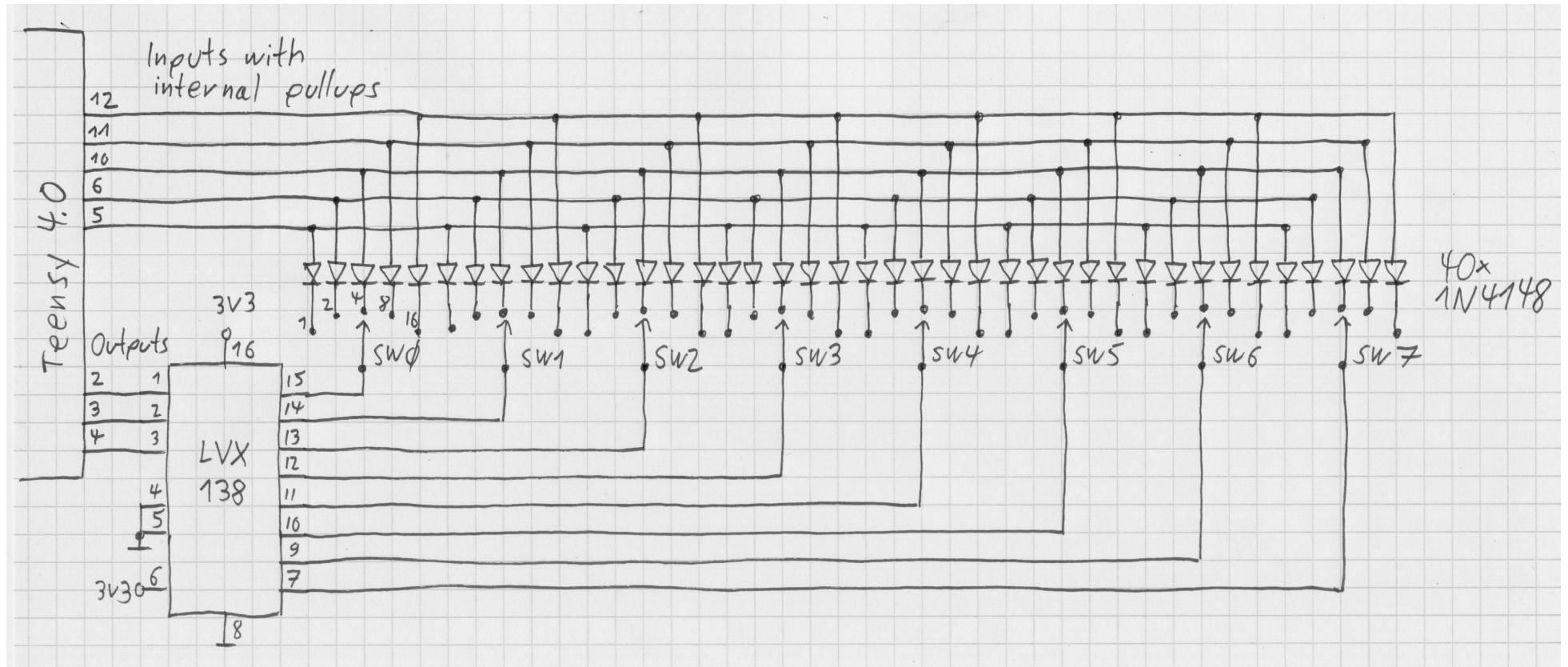
void loop() // nothing to do in the main loop
{
    if(ms >= 2500)
    {
        //Serial.println("Hello World");
        Serial.println(random(4294967295));
        //paused[0] = 0;
        ms = 0;
    }
}

```

}

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=gxdgRHmOulo>

<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

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

T0H: 0.35 μ s +- 0.15 μ s (0.20 μ s ... 0.50 μ s) T0L: 0.80 μ s +- 0.15 μ s (0.65 μ s ... 0.95 μ s) Reset: >50 μ s Low
 T1H: 0.70 μ s +- 0.15 μ s (0.55 μ s ... 0.85 μ s) T1L: 0.60 μ s +- 0.15 μ s (0.45 μ s ... 0.75 μ s)

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

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

T0H: 0.35 μ s +- 0.15 μ s (0.20 μ s ... 0.50 μ s) T0L: 0.90 μ s +- 0.15 μ s (0.75 μ s ... 1.05 μ s) Reset: >50 μ s (ab V4 >280 μ s)
 T1H: 0.90 μ s +- 0.15 μ s (0.75 μ s ... 1.05 μ s) T1L: 0.35 μ s +- 0.15 μ s (0.20 μ s ... 0.50 μ s)

Logic 0		Logic 1		Sum	BPS	Comment
High	Low	High	Low			
0.35 μ s (+0.00 μ s)	0.90 μ s (+0.00 μ s)	0.90 μ s (+0.00 μ s)	0.35 μ s (+0.00 μ s)	1.25 μ s	800kHz	symmetric timing
0.33 μ s (-0.02 μ s)	1.00 μ s (+0.20 μ s)	1.00 μ s (+0.30 μ s)	0.33 μ s (-0.27 μ s)	1.33 μ s	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 1 μ s.

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 μ s.

```
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++)
            PORT->Group[0].OUTSET.reg = (1ul << i) & pinMask;
    }
}
```

```

{
    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?)

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 μ s.

```
// 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 above 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

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 the 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: $0.3 \times VCC$ High-level input voltage: $0.7 \times VCC$ 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>

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/>

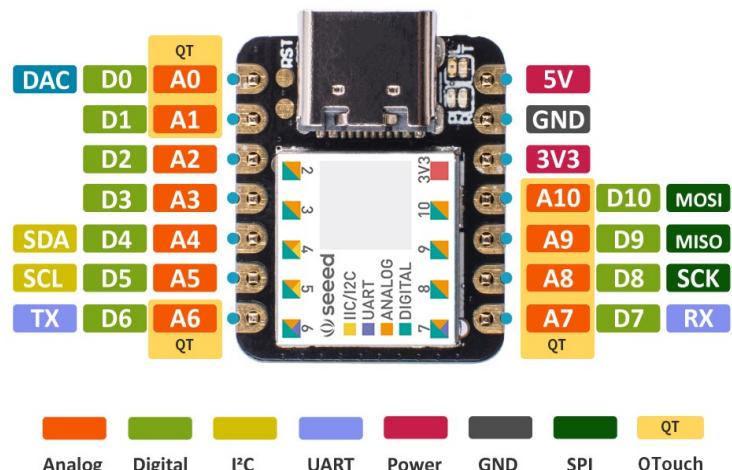
SAMD21 Data sheet: <https://files.seeedstudio.com/wiki/Seeeduino-XIAO/res/ATSAMD21G18A-MU-Datasheet.pdf>

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-Minetype%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 interrupts 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
}
```

4.5 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](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
```

4.6 MIDI

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

Index

1V / octave.....	34	digitalWriteFast.....	60	pinMode(led, OUTPUT).....	17
3.3V to 5V conversion.....	56	digitalWriteFast().....	7	port.....	8
74AHCT1G04.....	54, 56	Direct digital synthesis.....	25	PT8211 2-channel audio DAC.....	60
74AHCT1G125.....	52, 56	Eichelkopf mountain.....	20f.	random.....	32
74HCT1G125.....	56	exp().....	19	random().....	9
74LVC1G125.....	56	exponential function.....	19	randomSeed(13).....	32, 36
AD_B1_xx.....	13	GPIO pin order.....	10	Read multiple pins simultaneously.....	10
analogReadResolution().....	4	GPIO6_PSR.....	10	RGB LED.....	50
analogReadResolution(12).....	17	height profile.....	20	RGB LEDs and LED Cubes.....	48
analogReference().....	4	IDE Software installation.....	3	Seeed XIAL SAMD21.....	8
analogReference(EXTERNAL).....	17	IMXRT_GPIO6.PSR.....	10	Serial Monitor.....	14
analogWrite.....	18	INPUT_DISABLE.....	9	Serial Plotter.....	14
analogWriteResolution(12).....	17	INPUT_PULLDOWN.....	9	Serial.begin.....	14
approximate the exponential function.....	19	INPUT_PULLUP.....	9	Serial.println.....	14
Arduino IDE.....	3	interrupts().....	51	Teensy 4.1.....	13
Arturia MatrixBrute.....	34	IntervalTimer.....	8	Teensy LC.....	16
asm volatile("nop;");.....	4f.	Line driver chips.....	56	Teensyduino.....	3
asm("nop");.....	5	Look-up-table.....	17	THAT analog computer.....	16f.
Assembly code listing.....	3	MIDI.....	46, 61	Timer interrupts.....	59
Band-limited noise generator.....	32	MIDI_CREATE_INSTANCE.....	46	TimerTC3.h.....	59
Band-limited noise generator	34	MIDI.begin.....	46	TimerTCC0.h.....	59
const float curve[number_of_points] =.....	21	myTimer.begin.....	24	Tools --> Serial Monitor.....	14
Control voltage input from synthesizer.....	34	myTimer.priority.....	24	Tools --> Serial Plotter.....	15
CPU Speed.....	3	Nanoseconds delay.....	5	Triggered signal generator.....	29
Delay line.....	24	noInterrupts().....	50	Voltage controlled DDS.....	27
delay().....	4	nop.....	4	Workaround for nanoseconds delay.....	5
delayMicroseconds().....	5	OUTPUT.....	9	WS2812 RGB LED.....	50
delayNanoseconds.....	60	OUTPUT_OPENDRAIN.....	9	*.lst.....	3
delayNanoseconds().....	5, 51	pi = 4 * atan(1.0).....	27	<MIDI.h>.....	46
digitalRead().....	6	pinMask.....	8		
digitalWrite().....	7	pinmode().....	9		