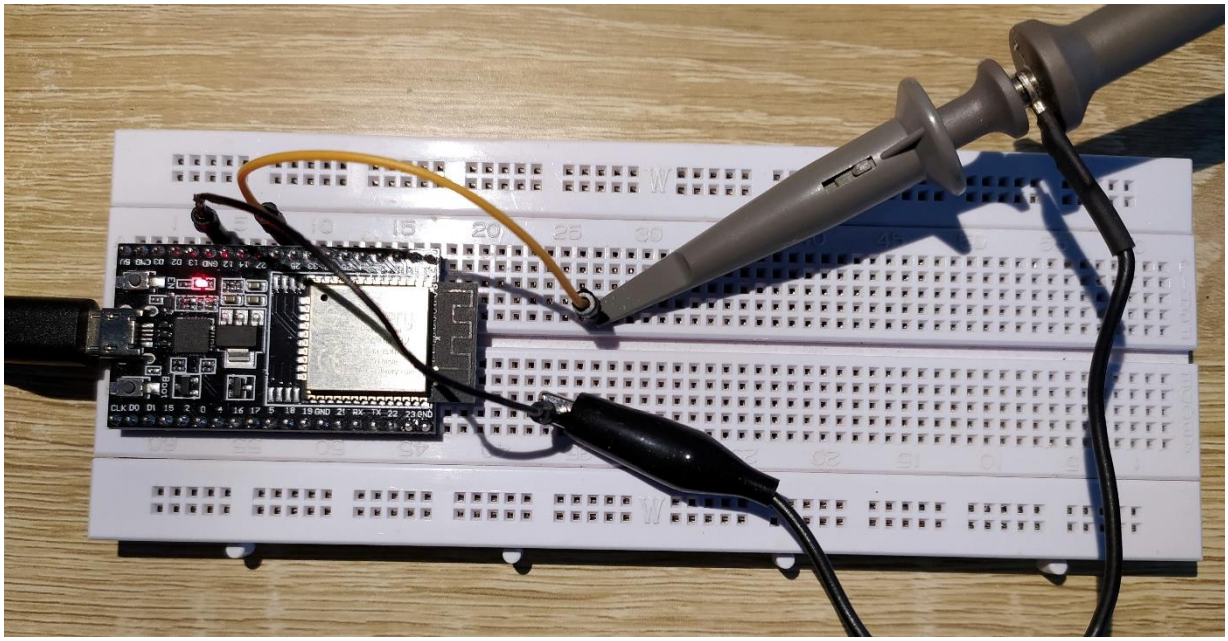


Funktionsgenerator mit dem ESP32 (Teil1)

In diesem Beitrag wollen wir mit einem ESP32 einen Funktionsgenerator bauen, der sich zu 100 Prozent der Hardware des ESP32 bedient. Die Software dient lediglich zur Bedienung. Da die Erzeugung der Wellenformen durch eingebaute Hardware des ESP32 erfolgt, gibt es keine Störungen durch den Programmablauf.

Der Funktionsgenerator liefert Sinus und Rechtecksignale mit einer Frequenz von 20Hz bis 200kHz, sowie Dreiecksignale mit einer Frequenz von 40Hz bis 20kHz. Für Rechteck und Dreieck kann das Tstverhältnis zwischen 0 und 100 % eingestellt werden. Die Ausgangsspannung ist nur positiv zwischen 0 und 3.3 V. Das Signal kann an GPIO26 des ESP32 abgenommen werden. Die Bedienung erfolgt über die serielle Schnittstelle. Im zweiten Teil erhält der Funktionsgenerator ein Display und eine Bedienung über Joy-Stick und natürlich ein Gehäuse aus dem 3D-Drucker.

Benötigte Hardware

Anzahl	Bauteil	Anmerkung
1	ESP32 Dev Kit C V4	

Der Sinusgenerator

Der ESP32 besitzt einen eingebauten Sinusgenerator, der sein Signal an den beiden Digital zu Analog Wandler Ausgängen (GPIO25 und GPIO26) ausgeben kann. Eine Periode kann in bis zu 65536 Schritte unterteilt werden. Als Takt wird der interne 8MHz Takt genutzt. Das heißt, für einen Schritt pro Takt müsste die Frequenz 8.000.000 / 65536 = 122 Hz betragen. Versuche haben allerdings gezeigt, dass die Frequenz bei dieser Einstellung 127 Hz beträgt. Somit ist der interne Takt höher als 8MHz.

Zur Einstellung der Frequenz kann die Schrittweite pro Takt eingestellt werden. Das bedeutet, die Frequenz = 127 * Schrittweite. Somit kann die Frequenz in 127 Hz Schritten eingestellt werden. Da dies für niedrige Frequenzen zu ungenau ist, gibt es noch eine zweite Einstellmöglichkeit. Der Takt kann durch 1 bis 8 geteilt werden. Damit ist die niedrigste Frequenz 127/8 = 15,9 Hz. Die gesamte Frequenzformel lautet also Frequenz = 127 / Vorteiler * Schrittweite. Für kleine Frequenzen sieht das dann so aus.

Schrittweite→ Vorteiler↓	1	2	3	4	5	6	7	8
8	15,9	31,8	47,6	63,5	79,4	95,3	111,1	127,0
7	18,1	36,3	54,4	72,6	90,7	108,9	127,0	145,1
6	21,2	42,3	63,5	84,7	105,8	127,0	148,2	169,3
5	25,4	50,8	76,2	101,6	127,0	152,4	177,8	203,2
4	31,8	63,5	95,3	127,0	158,8	190,5	222,3	254,0
3	42,3	84,7	127,0	169,3	211,7	254,0	296,3	338,7
2	63,5	127,0	190,5	254,0	317,5	381,0	444,5	508,0
1	127,0	254,0	381,0	508,0	635,0	762,0	889,0	1016,0

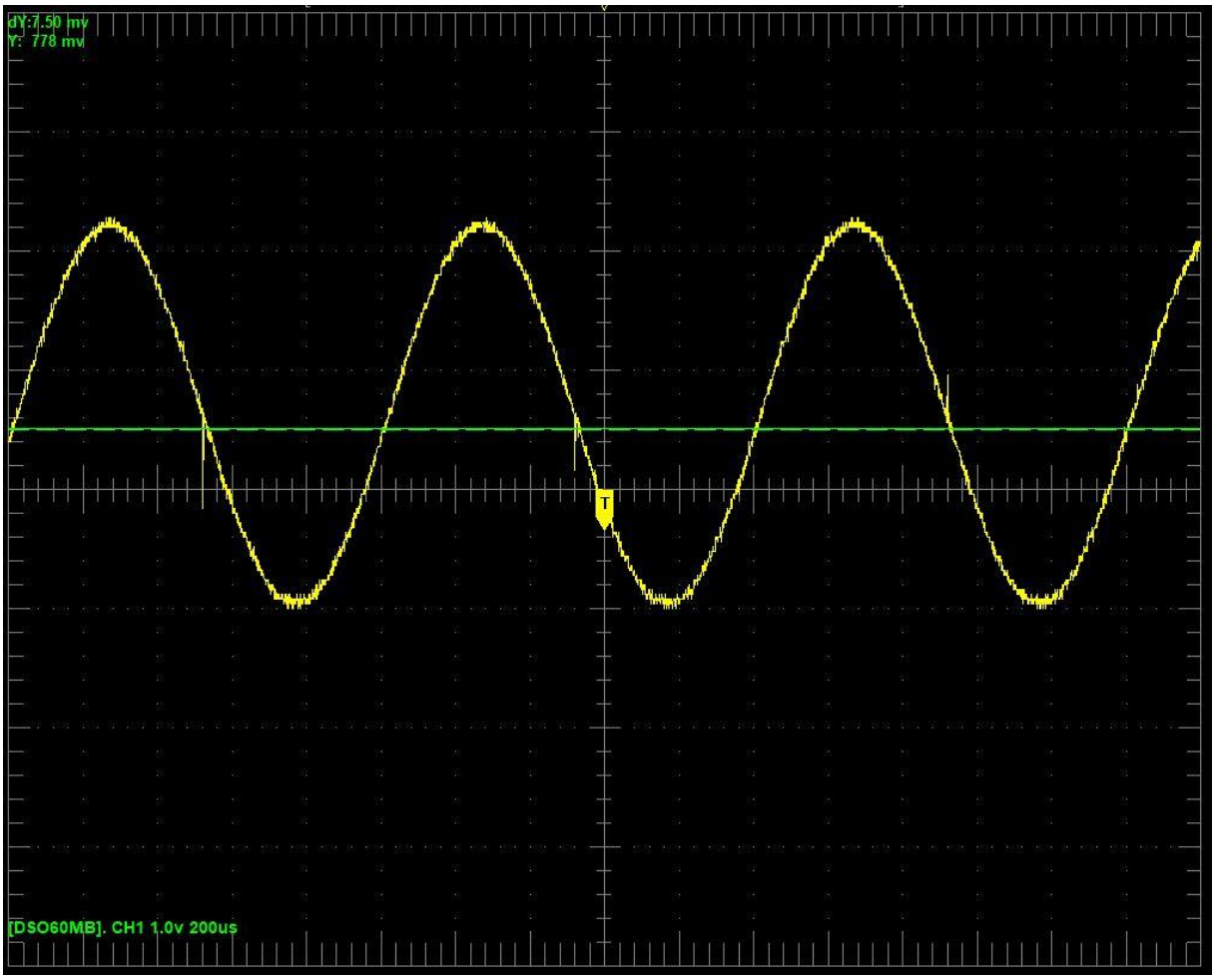
Man sieht, um eine gewünschte Frequenz möglichst gut anzunähern, muss man eine der beiden Variablen, Schrittweite oder Vorteiler, durchprobieren. Da die Schrittweite 65536 Möglichkeiten, der Vorteiler aber nur acht Möglichkeiten hat, ist es naheliegend, den Vorteiler durchzuprobieren. Zur Frequenzeinstellung berechnen wir die Schrittweite für jede der möglichen Vorteiler-Einstellungen und verwenden jene, bei der die geringste Frequenzabweichung auftritt. Damit eine schöne Sinusform erreicht wird, sollte die Schrittweite nicht größer als 1024 gewählt werden. Damit wird eine Sinuswelle aus 64 Schritten zusammengesetzt.

Da es für den Sinusgenerator keine fertige Bibliothek gibt, müssen die entsprechenden Bits in Steuerregistern des ESP32 eingestellt werden. Wer sich im Detail dafür interessiert, wie das funktioniert, der erhält die nötigen Informationen von

ESP32 Technical Reference Manual

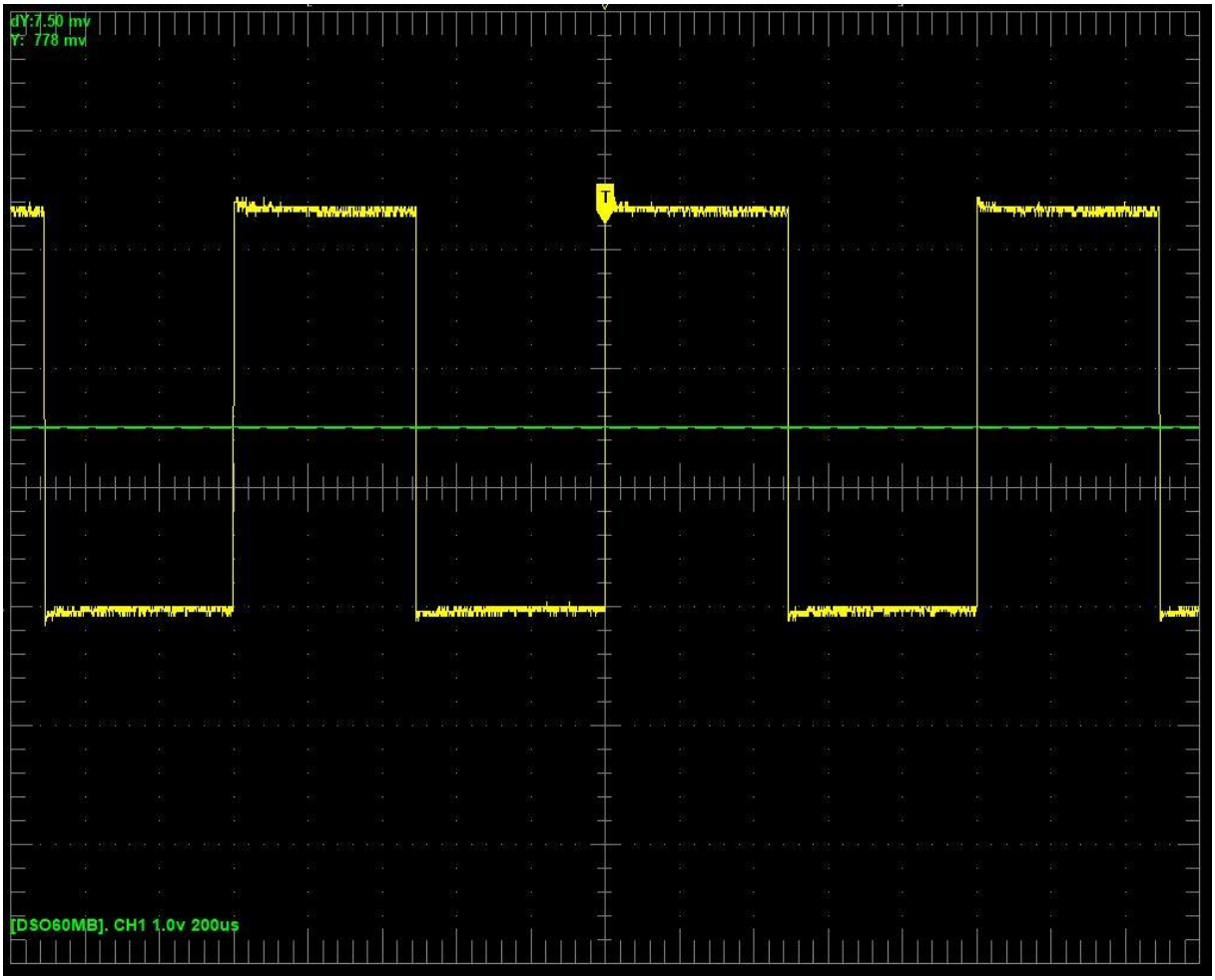
und zur Ansteuerung der Register aus der Arduino IDE

https://github.com/espressif/arduino-esp32/blob/master/tools/sdk/include/soc/soc/soc.h



Der Rechteckgenerator

Der ESP32 besitzt interne Timer, mit denen an einem beliebigen GPIO-Pin Rechtecksignale mit einstellbarem Tstverhältnis erzeugt werden können. Diese Signale sind in erster Linie zur Erzeugung von Pulsbreiten-Modulation gedacht, können aber auch als Rechteckgenerator mit variablem Tstverhältnis genutzt werden. Mit der Funktion ledcAttachPin(26,1) wird GPIO26 als Signalausgang für Timer 1 definiert. Die Funktion ledcSetup(1,frequency,7) setzt die Frequenz für Timer1 und die Auflösung für das Tstverhältnis auf 7 Bit. Die Funktion ledcWrite(1,127.0*ratio/100) setzt das Tstverhältnis von Timer1. 127 ist die maximale Anzahl von Schritten bei 7 Bit. Die Variable ratio enthält das Tstverhältnis in Prozent. Mit ledcDetachPin(26) wird der Anschluss GPIO26 wieder freigegeben.



Der Dreieckgenerator

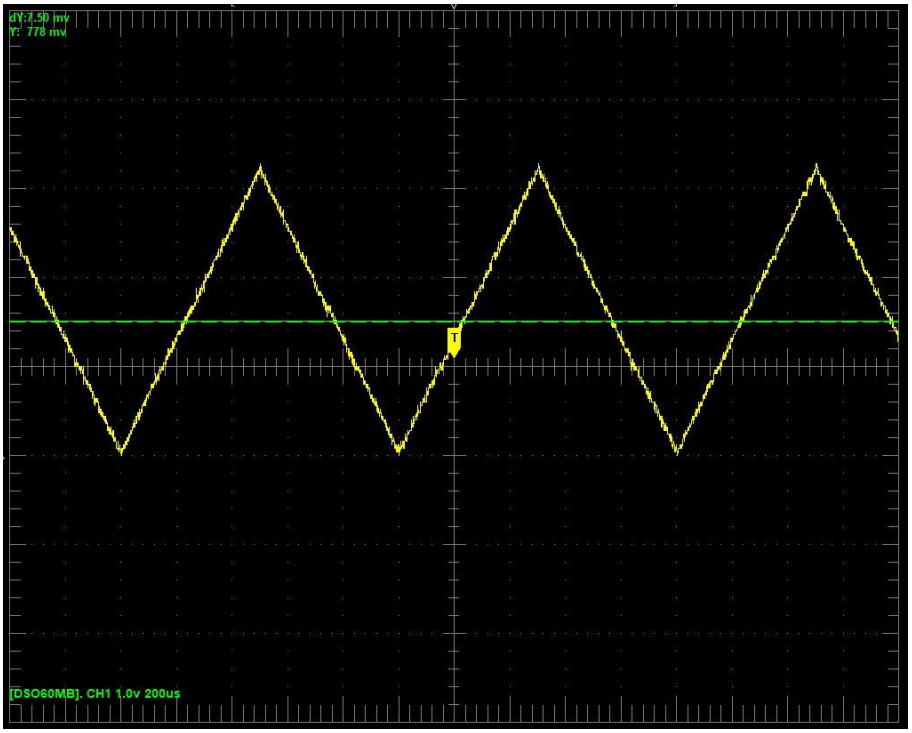
Für den Dreieck-Generator wurde die eingebaute I2S Schnittstelle zweckentfremdet. Eine Betriebsart der I2S Schnittstelle ermöglicht es, Audiodaten zum Beispiel aus einer WAV-Datei an die beiden Analogausgänge GPIO25 und GPIO26 auszugeben. Die Ausgabe erfolgt als Stereo-Signal und zwar der rechte Kanal auf GPIO25 und der linke auf GPO26. Jeder Abtastwert hat 32 Bit. Die höherwertigen 16 Bit enthalten den rechten, und die niederwertigen den linken Kanal. Andere Einstellungen für 1-Kanal Ausgabe und 8Bit sind zwar möglich, funktionieren aber nicht. Zur Ausgabe wird ein FiFo (First in, First out) Buffer verwendet.

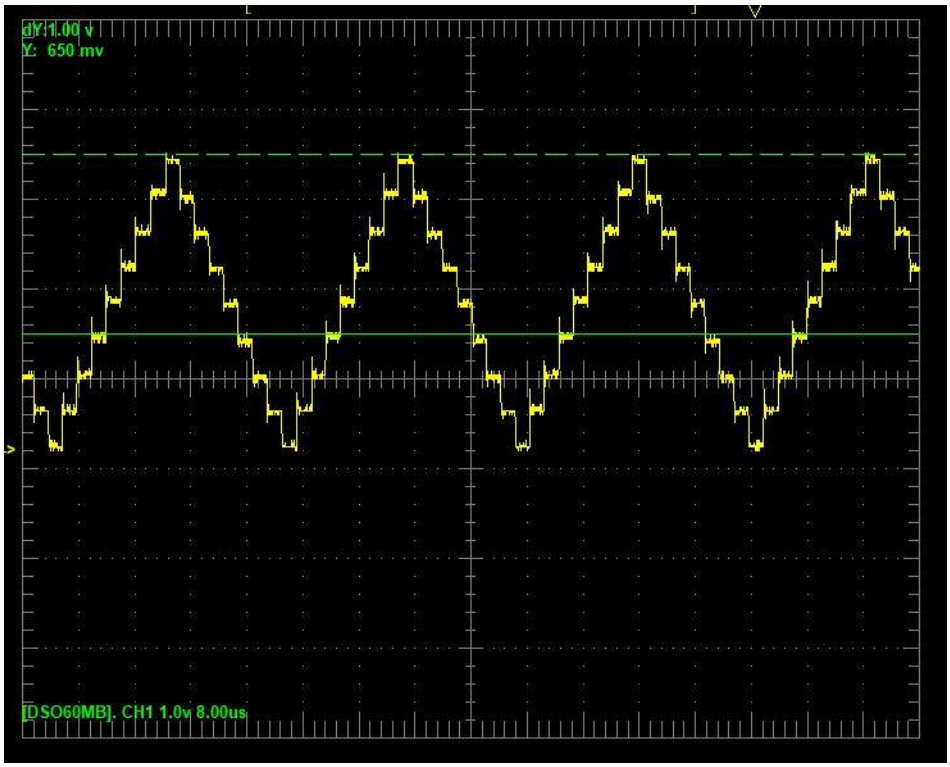
Nun der Trick, den wir benutzen, um damit einen Dreieck-Generator zu realisieren. Wenn der gesamte FiFo-Buffer genau mit einer Periode des Dreiecksignals befüllt wird, und danach kein weiterer Schreibvorgang erfolgt, gibt die I2S Schnittstelle den Inhalt des FiFo-Buffers immer wieder mit der eingestellten Abtastrate aus.

Experimente haben ergeben, dass die Abtastrate zwischen 5,2 kHz und 650kHz sein darf. Wenn wir für eine Periode 128 Abtastwerte nutzen, ergibt das einen Frequenzbereich von 5200/128 = 40,6 Hz bis 5,1 kHz. Für höhere Frequenzen muss die Anzahl der Abtastwerte pro Periode verringert werden. Mit 64 Abtastwerten erhält man 10kHz mit 32 Abtastwerten 20kHz und mit 16 Abtastwerten 40kHz.

Allerdings wird die Kurvenform mit abnehmender Zahl an Abtastwerten immer schlechter. Siehe zweites Bild mit 16 Schritten je Periode.

Das Tstverhältnis kann herangezogen werden, um anstatt eines Dreiecksignals einen Sägezahn zu erstellen. Je nach Tstverhältnis werden die Abtastwerte pro Periode aufgeteilt. Bei einem Tstverhältnis von 20% werden 0.2*128 = 26 Schritte für den Anstieg und 102Schritte für die abfallende Flanke genutzt.





Das Programm

```
/*
 *  Funktionsgenerator für Sinus, Dreieck und Rechteck Signale
 *  Einstellbare Frequenz 20 Hz bis 20 KHz
 *  Für Dreieck und Rechteck einstellbares Tastverhältnis 0 bis 100%
 *  Ausgangsspannung 3.3V Signale nur positiv!
*/

//Bibliotheken zum direkten Zugriff auf Steuerregister des ESP32
#include "soc/rtc_cntl_reg.h"
#include "soc/sens_reg.h"
#include "soc/rtc.h"

//Bibliotheken zur Verwendung des Digital zu Analog Konverters und für den
I2S-Bus
#include "driver/dac.h"
#include "driver/i2s.h"

#define SINFAKT 127.0 //gemessen für Schrittweite = 1 und kein Vorteiler
(8.3MHz)

//Buffer zum Erstellen der Dreieckfunktion
uint32_t buf[128];

//Einstellwerte für Kurvenform, Frequenz und Tstverhältnis
char mode = 'S'; //S=Sinus, R=Rechteck, T=Dreieck
float frequency = 1000; //20 bis 200000 Hz
uint8_t ratio = 50; //Tstverhältnis 0 bis 100%

//Flag Ist wahr, wenn die Initialisierung bereits erfolgte
bool initDone = false;

//Konfiguration für den I2S Bus
i2s_config_t i2s_config = {
     .mode     =     (i2s_mode_t)(I2S_MODE_MASTER     |     I2S_MODE_TX     |
I2S_MODE_DAC_BUILT_IN), //Betriebsart
     .sample_rate = 100000, //Abtastrate
     .bits_per_sample = I2S_BITS_PER_SAMPLE_16BIT, // der DAC verwendet nur
8 Bit des MSB
     .channel_format  =  I2S_CHANNEL_FMT_RIGHT_LEFT,  //  Kanalformat  ESP32
unterstützt nur Stereo
     .communication_format   =   (i2s_comm_format_t)I2S_COMM_FORMAT_I2S_MSB,
//Standard Format für I2S
     .intr_alloc_flags = 0, // Standard Interrupt
     .dma_buf_count = 2, //Anzahl der FIFO Buffer
     .dma_buf_len = 32, //Größe der FIFO Buffer
     .use_apll = 0 //Taktquelle
    };


//Buffer für Dreieck Wellenform füllen
//Parameter up ist die Dauer für den Anstieg in Prozent
//Parameter sz gibt die Buffergröße für eine Periode an
//es werden die Werte für eine Periode in den Buffer geschrieben
void fillBuffer(uint8_t up, uint8_t sz) {
  uint8_t down;  //Zeit für die fallende Flanke in %
  uint32_t sample; //32Bit Datenwort (I2S benötigt zwei Kanäle mit je 16
Bit
  float du,dd,val; //Hilfsvariablen
  down=100-up;
  //Anzahl der Schritte für Anstieg und Abfall berechnen
```

```
  uint16_t stup = round(1.0*sz/100 * up);
  uint16_t stdwn = round(1.0*sz/100*down);
  uint16_t i;
  if ((stup + stdwn) < sz) stup++;//Ausgleich eventueller Rundungsfehler
  //Amplitudenänderung pro Schritt für Anstieg und Abfall
  du = 256.0/stup;
  dd = 256.0/stdwn;
  //füllen des Buffers
  val = 0; //Anstieg beginnt mit 0
  for (i=0; i<stup; i++) {
    sample = val;
    sample = sample << 8; //Byte in das höherwertige Byte verschieben
    buf[i]=sample;
    val = val+du; //Wert erhöhen
  }
  val=255; //Abfallende Flanke beginnt mit Maximalwert
  //Rest wie bei der ansteigenden Flanke
  for (i=0; i<stdwn; i++) {
    sample = val;
    sample = sample << 8;
    buf[i+stup]=sample;
    val = val-dd;
  }
}


//Alle  Ausgänge stoppen
void stopAll(){
    ledcDetachPin(26);
    i2s_driver_uninstall((i2s_port_t)0);
    dac_output_disable(DAC_CHANNEL_2);
    dac_i2s_disable();
    initDone=false;
}

//Kurvenform Rechteck starten
//Pin 26 als Ausgang zuweisen
void startRectangle(){
    ledcAttachPin(26,1 );
    initDone=true;
}

//Frequenz für Rechteck setzen mit entsprechendem Tstverhältnis
void rectangleSetFrequency(double frequency,uint8_t ratio)
{
    ledcSetup(1,frequency,7);  //Wir  nutzen  die  LEDC  Funktion  mit  7  bit
Auflösung
    ledcWrite(1,127.0*ratio/100);     //Berechnung  der  Schrittanzahl  für
Zustand = 1
}


//Dreiecksignal starten
void startTriangle(){
  i2s_set_pin((i2s_port_t)0, NULL); //I2S wird mit dem DAC genutzt
    initDone=true;
}

//Frequenz für Dreieck setzen mit entsprechendem Tstverhältnis
double triangleSetFrequency(double frequency,uint8_t ratio)
{
  int size=64;
  //zuerst wird die geeignete Buffergröße ermittelt
```

```
  //damit die Ausgabe funktionier muss die I2S Abtastrate zwischen
  //5200 und 650000 liegen
  if (frequency<5000) {
    size = 64;
  } else if (frequency<10000) {
    size = 32;
  } else if (frequency<20000) {
    size = 16;
  } else {
    size = 8;
  }
  //Abtastrate muss in einer Periode beide Buffer ausgeben
  uint32_t rate = frequency * 2 * size;
  //Die Abtastrate darf nur innerhalb der Grenzwerte liegen
  if (rate < 5200) rate = 5200;
  if (rate > 650000) rate = 650000;
  //wirklichen Frequenzwert setzen
  frequency = rate / 2 / size;

  //I2S Treiber entfernen
  i2s_driver_uninstall((i2s_port_t)0);
  //Konfiguration anpassen
  i2s_config.sample_rate = rate;
  i2s_config.dma_buf_len = size;
  //und mit der neuen Konfiguration installieren
  i2s_driver_install((i2s_port_t)0, &i2s_config, 0, NULL);
  //Abtastrate einstellen
  i2s_set_sample_rates((i2s_port_t)0, rate);
  //Buffer füllen
  fillBuffer(ratio,size*2);
  //und einmal ausgeben
  i2s_write_bytes((i2s_port_t)0, (const char *)&buf, size*8, 100);
  return frequency;
}

//Sinusausgabe vorbereiten
void startSinus(){
    //Ausgang für Pin26 freigeben
    dac_output_enable(DAC_CHANNEL_2);
    // Sinusgenerator aktivieren
    SET_PERI_REG_MASK(SENS_SAR_DAC_CTRL1_REG, SENS_SW_TONE_EN);
    // Ausgabe auf Kanal 1 starten
    SET_PERI_REG_MASK(SENS_SAR_DAC_CTRL2_REG, SENS_DAC_CW_EN2_M);
    // Vorzeichenbit umkehren
    SET_PERI_REG_BITS(SENS_SAR_DAC_CTRL2_REG,       SENS_DAC_INV2,       2,
SENS_DAC_INV2_S);
    initDone=true;
}

//Frequenz für Sinus setzen
double sinusSetFrequency(double frequency)
{
  //Formel f = s * SINFAKT / v
  //s sind die Schritte pro Taktimpuls
  //v ist der Vorteiler für den 8MHz Takt
  //Es gibt 8 Vorteiler von 1 bis 1/8 um die Kombination Vorteiler und
  //Schrittanzahl zu finden, testen wir alle acht Vorteiler Varianten
  //Die Kombination mit der geringsten Frequenzabweichung wird gewählt

    double f,delta,delta_min = 999999999.0;
    uint16_t divi=0, step=1, s;
    uint8_t clk_8m_div = 0;//0 bis 7
    for (uint8_t div = 1; div<9; div++){
```

```
      s=round(frequency * div/SINFAKT);
      if ((s>0) && ((div == 1) || (s<1024))) {
        f= SINFAKT*s/div;
        /*
        Serial.print(f); Serial.print(" ");
        Serial.print(div); Serial.print(" ");
        Serial.println(s);
        */
        delta = abs(f-frequency);
        if (delta < delta_min) { //Abweichung geringer -> aktuelle Werte
merken
          step = s; divi = div-1; delta_min = delta;
        }
      }
    }
    //wirklichen Frequenzwert setzen
    frequency = SINFAKT * step / (divi+1);
    // Vorteiler einstellen
    REG_SET_FIELD(RTC_CNTL_CLK_CONF_REG, RTC_CNTL_CK8M_DIV_SEL, divi);
    // Schritte pro Taktimpuls einstellen
    SET_PERI_REG_BITS(SENS_SAR_DAC_CTRL1_REG,      SENS_SW_FSTEP,      step,
SENS_SW_FSTEP_S);
    return frequency;
}

//Einstellungsänderungen durchführen
void controlGenerator() {
  switch (mode) {
    case 'S' :
    case 's': if (!initDone) startSinus();
        frequency = sinusSetFrequency(frequency);
        break;
    case 'T' :
    case 't' : if (!initDone) startTriangle();
        frequency = triangleSetFrequency(frequency,ratio);
        break;
    case 'R' :
    case 'r' : if (!initDone) startRectangle();
        rectangleSetFrequency(frequency,ratio);
        break;
  }
}

//Serielle Schnittstelle aktivieren und
//Defaulteinstellungen 1kHz Sinus setzen
void setup()
{
    Serial.begin(115200);
    controlGenerator();
    Serial.print("Kommando M,F,R : ");
}


void loop(){
  //Serielle Schnittstelle abfragen
  if (Serial.available() > 0) {
    //Befehl von der Schnittstelle einlesen
    String inp = Serial.readStringUntil('\n');
    //und zur Kontrolle ausgeben
    Serial.println(inp);
    char cmd = inp[0]; //erstes Zeichen ist das Kommando
    if ((cmd == 'M') || (cmd == 'm')) { //war das Zeichen 'M' wird die
Betriebsart eingestellt
```

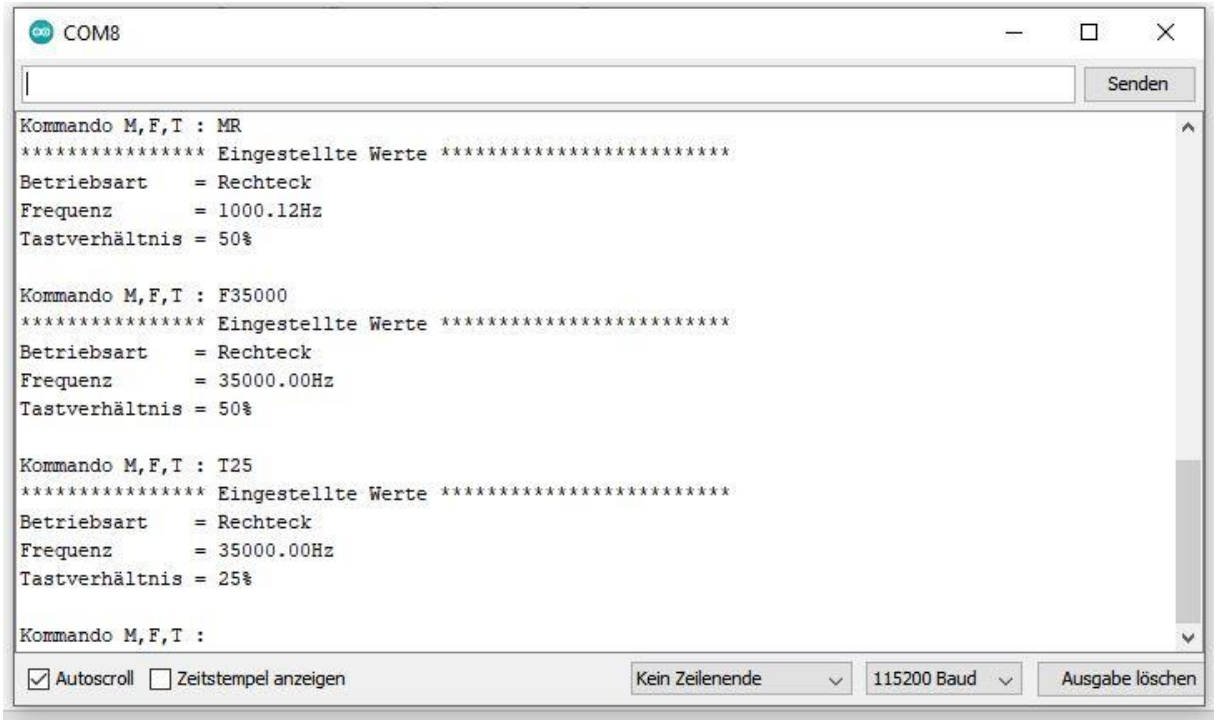
```
      char newMode = inp[1]; //zweites Zeichen ist die Betriebsart
      if (newMode != mode) { //Nur wenn eine Änderung vorliegt, mus was
getan werden
        stopAll();
        mode=newMode;
        controlGenerator();
      }
    } else {
      //bei den anderen Befehlen folgt ein Zahlenwert
      String dat = inp.substring(1);
      //je nach Befehl, werden die Daten geändert
      switch (cmd) {
        case 'F' :
        case 'f' :frequency = dat.toDouble(); break; //Frequenz
        case 'R' :
        case 'r' :ratio = dat.toInt(); break;  //Tstverhältnis
      }
      //Grenzwerte werden überprüft
      if (ratio > 100) ratio = 100;
      if (frequency < 20) frequency = 20;
      if (frequency > 200000) frequency = 200000;
      controlGenerator();
    }
    //aktuelle Werte ausgeben
    String ba;
    switch (mode) {
      case 'S':
      case 's': ba="Sinus"; break;
      case 'T':
      case 't': ba="Dreieck"; break;
      case 'R':
      case 'r': ba="Rechteck"; break;
    }
    Serial.println("***************          Eingestellte          Werte
************************");
    Serial.print("Betriebsart    = "); Serial.println(ba);
    Serial.print("Frequenz               =  ");  Serial.print(frequency);
Serial.println("Hz");
    Serial.print("Tastverhältnis      =      ");      Serial.print(ratio);
Serial.println("%");
    Serial.println();
    Serial.print("Kommando M,F,T : ");
  }
}
```

Bedienung

Die Bedienung erfolgt über die serielle Schnittstelle. Oben im Seriellen-Monitor ist eine Zeile, in die die Befehle eingegeben werden können. Mit dem Button „Senden“, wird der Text an die serielle Schnittstelle gesendet. Folgende Befehle sind möglich:

MS	Sinus
MR	Rechteck
MT	Dreieck
F####	Frequenz in Herz
R##	Tstverhältnis in Prozent

Es können auch Kleinbuchstaben verwendet werden. # steht für Zahleneingabe.


COM8
Senden
Kommando M,F,T : MR
********** Eingestellte Werte **********
Betriebsart = Rechteck
Frequenz = 1000.12Hz
Tastverhältnis = 50%
Kommando M,F,T : F35000
********** Eingestellte Werte **********
Betriebsart = Rechteck
Frequenz = 35000.00Hz
Tastverhältnis = 50%
Kommando M,F,T : T25
********** Eingestellte Werte **********
Betriebsart = Rechteck
Frequenz = 35000.00Hz
Tastverhältnis = 25%
Kommando M,F,T :
Autoscroll
Zeitstempel anzeigen
Kein Zeilenende
115200 Baud
Ausgabe löschen