HAGIWO/ハギヲ

モジュラーシンセを始めた人。仕事はレガシーなエンジニア。

フォロー

$8 Sample Drum with Seeed XIAO ESP32C3 - DIY Eurorack Modular Synthesizer

HAGIWO/ハギヲ
2022年9月4日 14:17

Seeed XIAO ESP32C3を使用してモジュラーシンセサイザー のサンプルドラムモジュールを作成したので、その備忘録。

[$8] DIY eurorack modular synth Sample drum with Seeed sturido...

背景

HAGIWO/ハギヲ
モジュラーシンセを始めた人。仕事はレガシーなエンジニア。
フォロー

自作モジュラーシンセの56作品目。

これまで、色んなMCUを使ってモジュールを作ってきた。ATMega、SAMD21、RP2040などなど。

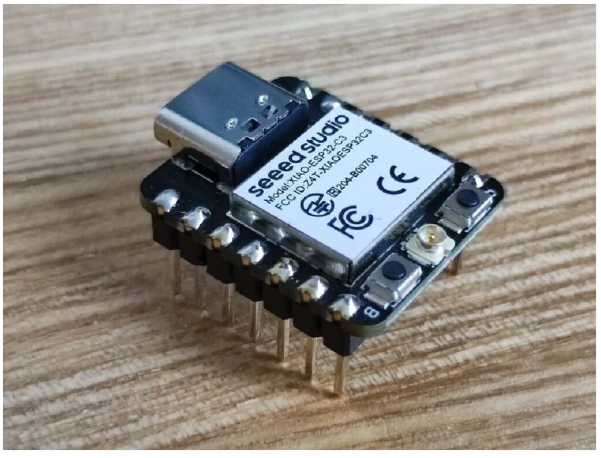
ったモジュールを作ったことがなかったので、ESP32の勉強のため-ルを作ることにした。開発ボードはSeeed XIAO ESP32C3を使うこ

Seeed XIAO ESP32C3

Seeed XIAO ESP32C3は、ESP32を使った開発ボードの中でも小型のため使いやすい。値段は$4.99と安価だ。

Seeed Studio XIAO ESP32C3 - tiny MCU board with Wi-Fi and BLE, battery charge supported, p...
Seeed Studio XIAO ESP32C3 featuring ESP32C3 carries a complet
www.seeedstudio.com

技適マークもついているので、法規を心配する必要もない。



ドラムモジュール

以前、Seeeduino xiaoを使用してドラムモジュールを作ったことがあるが、flash容量は256kbyteだったため、サンプルのビットレートを落としたり、サンプル数にも制限があった。

ESP32C3は4Mbyteのflashがあるため、より大容量のドラムサンプルを保存できると考え、DIYを企画した。

HAGIWO/ハギヲ

モジュラーシンセを始めた人。仕事はレガシーなエンジニア。

フォロー

制作物のスペック

ユーロラック規格 3U 6HPサイズ

電源：45mA (at 5V)

5V単電源で動作可能。

wavデータを再生可能な1 shot drumモジュール。

保存可能sample数：48sample

再生ビットレート：10bit

サンプリングレート：48kHz

最大sample再生時間：0.6sec

SELECT：再生するsampleを選択する。右に回すと次のsample、左に回すと一つ前のsampleを選択する。

PITCH POT ：sampleの再生スピードを調整する

Low pass filter SW：ONするとパッシブローパスフィルタが有効になる。PWM高調波のノイズが低減される。kickなどの低周波数のsample再生時にONすることを想定している。

PITCH CV : sampleの再生スピードを調整する。(0-5Vを想定）

TRIG IN：トリガーを受信するとsampleを再生する。(0-5V想定）

OUT：音声出力

HAGIWO/ハギヲ

モジュラーシンセを始めた人。仕事はレガシーなエンジニア。

フォロー

設定保存機能：SELECTで音色切り替えした5秒後に、選択中のsampleナンバ

：れる。電源をOFF/ONした際は、保存済のsampleナンバーが読み

製作費

総額約1000円

フロントパネル 100円

Seeed Xiao ESP32C3 600円

ロータリーエンコーダ 80円

オペアンプMPC6232 45円

他（汎用部品は下記リンク先参照）

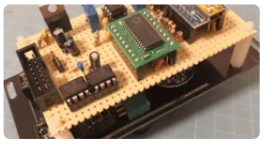
HAGIWO/ハギヲ

モジュラーシンセを始めた人。仕事はレガシーなエンジニア。

フォロー

モジュラーシンセ自作で使う安価な部品一覧

に。適時更新。コストと品質の観点で記載、主観による。aliex クを貼ってもすぐに切れるので、画像で残す。値段は購入当時 は値動きが激しい。品質は私の経験値。よって母数は数個か...



WO/ハギヲ
2021/11/19 16:18

ハードウェア

音声出力回路

音声はD5pinからPWMで出力する。オペアンプとの間にはローパスフィルタを設置しているが、kick等の低音sampleは高調波ノイズが気になる。

その場合はLPF SWでカットオフ周波数を切り替えて、ノイズを低減する。

HAGIWO/ハギヲ
モジュラーシンセを始めた人。仕事はレガシーなエンジニア。
フォロー

トリガー入力回路

0-5Vのトリガー信号が入力した時に、MCUがHigh-Low検出可能かつ、LEDが点灯する回路になっている。

MCUに対する過電圧の保護はLEDを使用している。順方向電圧(Vf）で電圧が[illegible]れる。

[illegible]する負電圧の保護は、ダイオードを使用している。

注意点として、LEDの順方向電圧Vfが、MCUのHigh-Lowのスレッショルドを超える必要がある。今回、Vfが大きい白色LED(Vf=2.2V)を使用したら問題なく動作した。Vfが小さい赤色LEDだと、機能しない可能性がある（未検証）。

ソフトウェア

割り込み

タイマー0を使って、48kHz周期でIRAM_ATTR onTimer()の処理を読み足している。

HAGIWO/ハギヲ

モジュラーシンセを始めた人。仕事はレガシーなエンジニア。

フォロー

```
AM_ATTR onTimer() {
TER_CRITICAL_ISR(&timerMux0) ; // enter critical range
ss
IT_CRITICAL_ISR(&timerMux0) ; // exit critical range
```

```
void setup() {

 timer0 = timerBegin(0, 1666, true);  // timer0, 12.5ns*1666 = 20.83usec(48kHz), count-up
 timerAttachInterrupt(timer0, &onTimer, true); // edge-triggered
 timerAlarmWrite(timer0, 1, true); // 1*20.83usec = 20.83usec, auto-reload
 timerAlarmEnable(timer0); // enable timer0
}
```

音声読み出し

音声のwavデータはflashに格納してある。ESP32C3のflash領域はスペック上は4Mbyteだが、実際に使えるのは3Mbyteだ。また、デフォルト設定では1.3Mbyteしか使えないので、Arduino IDEのToolタブよりPartition Schemeの設定を"Huge APP"に設定してやる必要がある。

HAGIWO/ハギヲ

モジュラーシンセを始めた人。仕事はレガシーなエンジニア。

フォロー

flashへ格納しているデータは16bitだが、音声出力は10bitだ。
flashに格納されている16bit = 8bit + 8bitを読み出して、">>6"で10bitに落としている。

```
sound_out = (((pgm_read_byte(&(smpl[sample_no][(int)i * 2]))) | (pgm_read_byte(&(smpl[sa
```

音声出力

音声出力はDACではなく、PWMを用いている。
PWMによる高調波ノイズを低減するために、PWM周波数は可能な限り大きい値を設定してやる必要がある。

PWMの設定はsetup()で行っている。
タイマー1、39000Hz、分解能10bitで設定し、タイマー1はD5のアウトプット

に割り当てるという処理をしている。

HAGIWO/ハギヲ

モジュラーシンセを始めた人。仕事はレガシーなエンジニア。

フォロー

詳細な検証はしていないが、PWM周波数と分解能は背反関係にある。PWM周波数を上げると、分解能を下げてやる必要がある。逆に、PWM周波数を下げると分解能を上げることができる。

今回は、40000Hz、分解能10bitでは正常に動作しなかったため、周波数を39000Hzに設定している。

```
void setup() {

 pinMode(D5, OUTPUT);//sound_out PWM

 ledcSetup(1, 39000, 10);//PWM frequency and resolution
 ledcAttachPin(D5, 1);//(LED_PIN, LEDC_CHANNEL_0);//timer ch1 , apply D5 output
}
```

PWMの出力にはledcWrite関数を使用する。

```
 ledcWrite(1, sound_out+ 511); //PWM output
```

サンプルデータの作成方法

音声サンプルデータはESP32C3のflashメモリにバイナリデータ保存している。バイナリデータの作成方法は以下の通り。

1.Audacityを使って、サンプルレート48kHz、時間0.60sec～0.61secのサンプルデータを作る。
0.60sec未満だと再生時にノイズが入る懸念がある。また、0.61sec以上だとコンパイルが失敗する。

HAGIWO/ハギヲ

モジュラーシンセを始めた人。仕事はレガシーなエンジニア。

フォロー

2.ファイルの保存をする。ファイル形式は「その他の非圧縮ファイル」で、ヘッダーはRAW、エンコーディングはSigned 16-bit PCMとする。

HAGIWO/ハギヲ

モジュラーシンセを始めた人。仕事はレガシーなエンジニア。

フォロー

3.RAWファイルをPROGMEM形式の配列データに変換して、Arduino IDEからsample.hに配列データを入力する。

「PROGMEM作蔵さん」を使うと便利。

https://hello-world.blog.ss-blog.jp/2016-10-16

HAGIWO/ハギヲ

モジュラーシンセを始めた人。仕事はレガシーなエンジニア。

フォロー

4.以上の作業を48回繰り返す。

「妥協」と「言い訳」

ESP32シリーズは初めて使うMCUなので、慣れてない事も多かった。今回のモジュールの作成では、機能の妥協をしている。
妥協した理由は、コストダウンの為だったり、IC性能の限界だったり、私の技術力が低いためだったりする。

音声出力の分解能が10bitの理由

高い周波数レートを維持するために、PWM分解能を10bitまで下げる必要があったため。格納しているwavデータが16bitなので、6bit捨てていることになる。もったいない。

HAGIWO/ハギヲ

モジュラーシンセを始めた人。仕事はレガシーなエンジニア。

フォロー

サンプリングレートが48kHzの理由

人間の可聴域と、flashメモリの容量を考慮すれば、サンプリングレートは40kHzもあれば十分だと思っている。しかし、フリーダウンロードできるwavサンプルは48kHzのものが多いので、それに合わせて48kHzにした。

[illegible]の音声編集ツールを使ってサンプリングレートは変更可能だが、[illegible]なので妥協した。

PWM出力の理由

外部のDACを使って音声出力ができるのが理想だ。

SPI通信でDACを動かそうと思ったが、高分解能かつ安価なICを見つけられなかったので、この案は却下した。

外部のDACを使った音声出力は、過去のプロジェクトでも出来ていない。単純に、私の経験不足、技術力不足が原因かもしれない。将来チャレンジしたい。

宣伝：オープンソースプロジェクトの支援をお願いします

DIYモジュラーシンセのオープンソースプロジェクトを継続するために、patreonというサービスでパトロンを募集しています。

コーヒー一杯の支援をいただけると嬉しいです。

また、パトロン限定のコンテンツも配信しています。

ソースコード

粗末だが公開する。悪い点があれば指摘を貰えると嬉しい。

ロータリーエンコーダ用に"Encoder.h"ライブラリを使っている。

プロジェクトはinoファイルと"sample.h"の２つのファイルから構成される。

2022/NOV/1追記

ソースコードのバグを修正。ロータリーエンコーダでサンプルを選ぶ際のオーバーフロー修正。

HAGIWO/ハギヲ

モジュラーシンセを始めた人。仕事はレガシーなエンジニア。

フォロー

ル

```
        ENCODER_OPTIMIZE_INTERRUPTS //rotary encoder
#include "sample.h"//sample file
#include <Encoder.h>//rotary encoder
#include <EEPROM.h>

Encoder myEnc(D10, D9);//rotary encoder
float oldPosition  = -999;//rotary encoder
float newPosition = -999;//rotary encoder

float i; //sample play progress
float freq = 1;//sample frequency
bool trig1, old_trig1, done_trig1;
int sound_out;//sound out PWM rate
byte sample_no = 1;//select sample number

long timer = 0;//timer count for eeprom write
bool eeprom_write = 0; //0=no write,1=write

//-------------------------timer interrupt for sound---------------------------------
hw_timer_t *timer0 = NULL;
portMUX_TYPE timerMux0 = portMUX_INITIALIZER_UNLOCKED;
volatile uint8_t ledstat = 0;

void IRAM_ATTR onTimer() {
  portENTER_CRITICAL_ISR(&timerMux0) ; // enter critical range
  if (done_trig1 == 1) {//when trigger in
    i = i + freq;
    if (i >= 28800) {//when sample playd all ,28800 = 48KHz sampling * 0.6sec
      i = 0;
      done_trig1 = 0;
    }
  }
  sound_out = (((pgm_read_byte(&(smpl[sample_no][(int)i * 2]))) | (pgm_read_byte(&(smpl[s
  ledcWrite(1, sound_out+ 511); //PWM output
  portEXIT_CRITICAL_ISR(&timerMux0) ; // exit critical range
}

void setup() {
  EEPROM.begin(1);  //1byte memory space
  EEPROM.get(0, sample_no);//callback saved sample number
  sample_no++;//countermeasure rotary encoder error
  if (sample_no >= 48) {//countermeasure rotary encoder error
    sample_no = 0;
  }

  pinMode(D7, INPUT); //trigger in
  pinMode(D9, INPUT_PULLUP); //rotary encoder
  pinMode(D10, INPUT_PULLUP); //rotary encoder
  pinMode(D5, OUTPUT);//sound_out PWM
  timer = millis();//for eeprom write
  analogReadResolution(10);

  ledcSetup(1, 39000, 10);//PWM frequency and resolution
  ledcAttachPin(D5, 1);//(LED_PIN, LEDC_CHANNEL_0);//timer ch1 , apply D5 output
```

HAGIWO/ハギヲ

モジュラーシンセを始めた人。仕事はレガシーなエンジニア。

フォロー

```
  timer0 = timerBegin(0, 1666, true);  // timer0, 12.5ns*1666 = 20.83usec(48kHz), count-u
  timerAttachInterrupt(timer0, &onTimer, true); // edge-triggered
  timerAlarmWrite(timer0, 1, true); // 1*20.83usec = 20.83usec, auto-reload
  timerAlarmEnable(timer0); // enable timer0
}

void loop() {
      ------------------------trigger----------------------------------
      rig1 = trig1;
       = digitalRead(D7);
      rig1  == 1 && old_trig1 == 0 ) { //detect trigger signal low to high , before samp
      e_trig1 = 1;
       0;


  //-------------------------pitch setting----------------------------------
  freq = analogRead(A3) * 0.002 + analogRead(A0) *  0.002;

  //-------------------------sample change----------------------------------
  newPosition = myEnc.read();
  if ( (newPosition - 3) / 4  > oldPosition / 4) {
    oldPosition = newPosition;
    sample_no = sample_no - 1;
    if (sample_no < 0 || sample_no > 200) {//>200 is overflow countermeasure
      sample_no = 47;
    }
    done_trig1 = 1;//1 shot play when sample changed
    i = 0;
    timer = millis();
    eeprom_write = 1;//eeprom update flug on
  }

  else if ( (newPosition + 3) / 4  < oldPosition / 4 ) {
    oldPosition = newPosition;
    sample_no = sample_no + 1;
    if (sample_no >= 48) {
      sample_no = 0;
    }
    done_trig1 = 1;//1 shot play when sample changed
    i = 0;
    timer = millis();
    eeprom_write = 1;//eeprom update flug on
  }

  //-------------------------save to eeprom----------------------------------
  if (timer + 5000 <= millis() && eeprom_write == 1) {//Memorized 5 seconds after sample
    eeprom_write = 0;
    eeprom_update();
  }
}

void eeprom_update() {
  EEPROM.put(0, sample_no);
  EEPROM.commit();
}
```

sample.h

合計で3Mbyteのバイナリデータの羅列なので、ソースコードの全掲載は省略する。オリジナルのソースコードファイルはpatreonにアップロードする、今後の活動のために支援いただけると嬉しい。

https://www.patreon.com/posts/71493297?pr=true

HAGIWO/ハギヲ

モジュラーシンセを始めた人。仕事はレガシーなエンジニア。

フォロー

```
const static byte smpl[48][58560] PROGMEM = {//58560=48kHz*0.61sec*2byte
                                                e01
                                                x00,0x0d,0x00,0x37,0x00,0xd5,0x00,0x83,0x02,0x25,0x06,0xad,0x0c,0x63,0x17,
                                                ting
                                                ting
  0x73,0x39,0x53,0x3a,0xb2,0x3a,0x83,0x3a,0xac,0x39,0x59,0x38,0x9d,0x36
 }
};
```