8bitAVRマイコンによるデジタル信号処理の実験 1

BrightNotes

あわざる

自己紹介

はじめまして. まずはこの冊子を手にとっていただいてありがとうございます.BrightNotes のあわざる[*1]と申します. 私達のサークルのメインの活動は楽曲制作ですがここでは音に関わる電子工作について情報を発信していければと思っています. 私自身は DTM の経験はありませんがいくつかの楽器の経験があります. いずれは自分で制作したツールと楽器を使って何かしらの曲を制作して発表することを目標に活動していきたいと考えています.

導入

今回は実装ができたディレイとピッチ変換, 用いた IC の具体的な操作法をまとめます. 制作に使った AVR マイコンは Atmel 社が製造しているマイクロコントローラで最近流行りの Arduino に搭載されている種類のマイコンです. 制作ではその中の Atmega328P[*2]を用いました. まだ実用性は低いと思っていますが, いずれ自分で楽器を演奏するときに使える作品にしたいと考えています.

[*1] Twitter:@awazaru0524

[*2] まさに Arduino に搭載されているもの

マイコンで行うデジタル信号処理の基本

信号の入力

携帯音楽プレーヤー等から出力される信号はアナログ信号がほとんどでデジタル回路のマイコンでは直接扱うことができません. そこでアナログ信号をマイコンに扱えるデジタル信号に変換してやる必要があります. 多くのマイコンには A/D 変換器が搭載されていてそれを使うことでその変換を行うことができます.A/D 変換器[*3]の使い方はデータシートと世の中の親切な方々にお任せするとして, ここでは実際にプレーヤ等から出力される信号を扱う際に注意することをまとめます. A/D 変換ではあらかじめ基準の電圧を用意して基準電圧に対する入力信号の電圧の割合を数値に変換します. 例えば 8bit[*4]の A/D 変換では 5(V) の基準電圧に対して 3(V) の信号を入力すると

$$255 \times \frac{3}{5} = 153$$

となり,"153" という値でマイコンに入力されます.

200($\mathrm{mV_{pp}}$) など振幅が小さい信号をそのままデジタル値に変更すると荒い変換[*5]になってしまいますので A/D 変換前に信号を十分に増幅してあげる必要があります. 振幅レベルを基準電圧いっぱいにすると細かい変換ができます.iPod などのプレーヤーでは出力振幅レベルは最大で 2($\mathrm{V_{pp}}$) 程度です.A/D 変換入力前に 10 から 20 倍程度の増幅ができるアンプ[*6]を用意すれば音源で iPod を使用した場合でも大抵問題ないと思います.

[*3] Atmega328 では最大 10bit 使えますが今回は 8bit で使用しています

[*4] 最大値 255

[*5] 量子化ビット数と関係します

[*6] 今回はオペアンプを使いました

次に問題になるのは A/D 変換を行う頻度です. つまりはサンプリング周波数です. これは使用したい信号の最大の周波数の 2 倍程度をサンプリング周波数として用意してあげればいいとされています. 例えば CD などでは 44100(Hz) が利用されていて, これは人間の可聴域の周波数が 20 から 20000(Hz) 程度とされていることが関係しているのでしょう. 私も近い 44000(Hz) を使いましたが, いささかオーバーな気もします. というのも実際に音楽などで使用される音で 20000(Hz) 近くを使用する機会はあまりなく, また高すぎるサンプリング周波数はマイコンの処理にも負荷をかけるからです[*7]. もっと低くてもいいと思いますが今回は試していません.

なにはともあれ, こうして変換され, 入力された値はマイコン内部に保存されます. この値を使っていろんな信号処理を行います. つまり電気信号をデータに変換できたことになります.

信号の出力

処理をかけた信号は出力されないと意味がありません. デジタル値として保存されている信号をアナログ値に変換します. マイコンで D/A 変換を行う方法はいくつかあるようですが, 私は PWM とローパスフィルタの二つを利用して D/A 変換を行いました.PWM は Pulse Width Modulation の略で High と Low の 2 通りしか出力できないマイコンで擬似的にその間の値を出力できるようにするものです. これも詳細は他者に譲りますが 8bit の PWM であれば単位時間を等しく 255 分割してその中である一定時間 High を出力し, 残りを Low にすることで平均的には High と Low の中間の電圧を作ることができます. 先ほどの例を利用すると 3(V) の信号を入力したときに”153” という値がマイコンに保存されているのでその値をそのまま PWM 生成の比較用の値として利用すればいいことになります. 最大 10bit

[*7] 割り込み処理の間隔が短くなる

まで利用できる AD 変換で 8bit を採用している理由はこのあたりにあったりします. 具体的には PWM の最大値が 5(V) であれば

$$5 \times \frac{153}{255} = 3.0(V) \tag{1}$$

となります. こうしてデータを電気信号として出力できるようになりました. このまま直接スピーカ等と繋いでも音は聞こえますがあとの事[*8]を考えて PWM のパルスの周波数を取り除いてあげます. 今回は RC のローパスフィルタを使いました. 回路定数を変えながらいくつかのフィルタを試しましたが,PWM のパルスの周波数を十分に取り除くことができるフィルタは実際に必要なオーディオ帯域の信号も周波数によりますが減衰させてしまいます. バランスを考えて値を選ぶことが必要です.

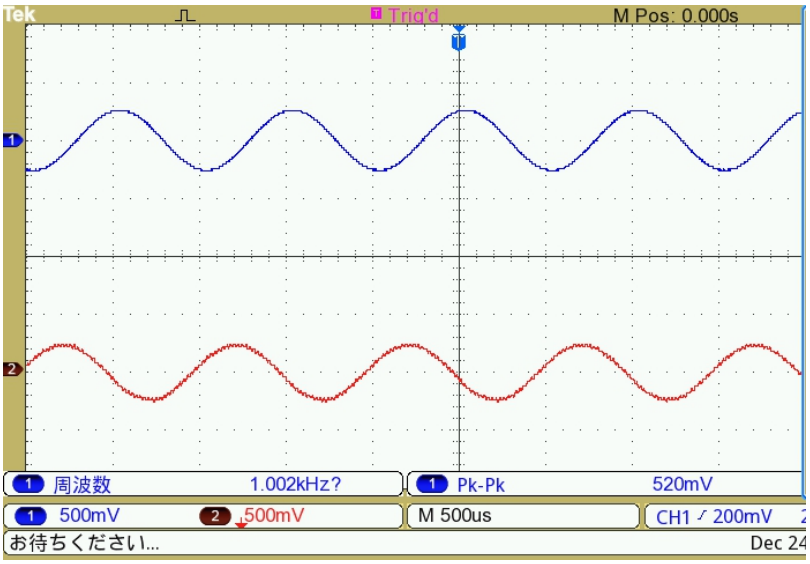

図 1　入力と出力

図 1 では入力と出力の信号の様子を表しています. 上が入力, 下が出力です. 今回は 1(kHz) の正弦波を入力しています. 出力が入力に対して遅れてい

[*8] オーディオアンプに繋ぐ等

ることを確認できます. 具体的な遅延時間はわかっていませんが, 耳で聞いて判断できるレベルではありません.

ディレイ処理

ディレイ処理は入力した信号が後追いで出力されてくる処理です. 入力されたデータを一定時間マイコンの中で保存してから出力すれば, そうでない場合とくらべて遅れて出力されることになります. ディレイ処理で重要なのは入力されたデータを保存するメモリの容量です.1 秒間に 44000 個[*9]の 8bit のデータがマイコンに入力されることになります. つまり 1 秒のディレイをかけるためには常に 44000 個分のデータを保存できる容量が必要になります. 8bit のデータが 1B だとすると,44000B=44kB のデータが保存できるメモリが必要です.Atmega328P の SRAM 容量は 2kB ですので全く足りません. そこで外付け SRAM の 23LC1024[*10]を使用することにしました. これはシリアル通信でマイコンと接続することができる SRAM で今回は SPI という規格で接続しました. 容量も 128kB あり 1 秒のディレイ処理だけでなくそれ以上の時間の処理にも対応できます. A/D 変換で入力されてくるデータを次々に外付け SRAM の方に送り, しばらく立ってから書き込みと同じ速度で読み込みを行っていくことで入力した信号がそうでない場合に比べて後追いのように出力されてくる, という機能ディレイ処理を実装することができます.

図 2 は実際の入出力を表しています. 上が入力, 下が出力の波形です. 入力信号として 1(Hz) の方形波をいれていますが波形は入力も出力も随分様子が違います. コンデンサの影響で波形がかなり変わってしまっていますが, とりあえずディレイの様子を伝えるためには問題なしとして使っています. 次に図 3 では出力が入力に対して明らかに遅れていることがわかると思います.

[*9] サンプリング周波数による

[*10] 詳細は後述

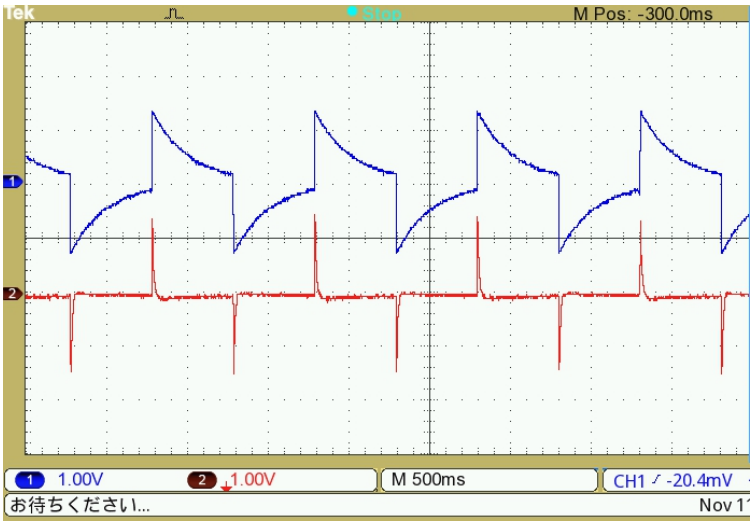

図 2　ディレイ無し

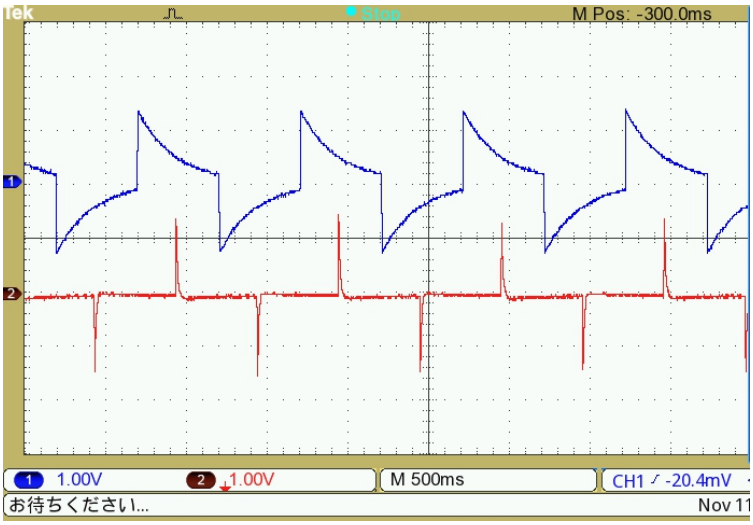

図 3　ディレイ有り

ピッチ変換

ピッチ変換は入力信号のピッチ, つまり音の高さを変えることができます. ポイントは単純に再生速度を変えるわけではないことです. 元の信号とほぼ同じ再生速度で音程を高くしたり低くしたりすることができます. これはある程度の長さのリングバッファ (つまりは配列) を用意し,A/D 変換によって書き込む速度を一定の状態で配列からデータを読み込む速度を変えることで実現できます. ある程度の長さのバッファとは具体的には 20ms 程度のバッファ[*11]です. 20ms というのは日本語に適したバッファの長さのようです [1]. ピッチ変換では Atmega328P 内部の SRAM で十分対応ができます.

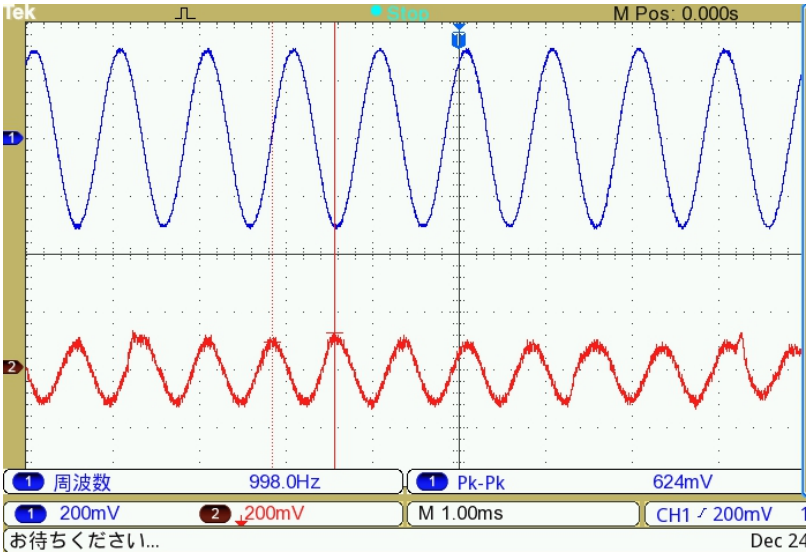

図 4　ピッチ変換の入力と出力

図 4 はピッチを 1.5 倍に変更するようにプログラムしたマイコンに 1(kHz) の信号を入力した様子です. 例によって上が入力, 下が出力になっています. 位相はだいぶずれていますが, 確かに出力が入力の 1.5 倍程度の周波数に

[*11] 今回の例では 880 個の要素の配列

なっていることがわかります. また出力がこれまでに比べてかなり乱れていることがわかると思います. 原因はいくつか考えられますが一つは書き込み位置の座標を読み込み位置の座標が追い抜いたり追い越されたりすることによ発生するノイズです. 図 4 で出力の右端の方に見られる乱れなどはその影響であると思われます. 音質は実際かなり低下するように感じられますが, 音の高さが変わっていることははっきり確認できるレベルです.

ノイズを消す方法もいくつか考えて試してみたのですが, あまりうまくいかず今回の実験ではノイズを取り切ることができませんでした (汗)

まとめ

今回は信号の入出力から, ディレイ処理, ピッチ変換を行いました. どちらの処理もプログラムは非常に単純で 8bit マイコンでも十分使えるレベルです. 今後は 8bit マイコンでリアルタイムにできる処理, できない処理をいろいろ試しながら学んでいきたいと思っています. 同時に今回実装した処理の最適化や改良も行いたいと思っています[*12]. 信号処理を座学で勉強したわけではなく, またマイコン処理も未熟ですのでところどころツッコミどころもあるとは思いますがご容赦ください. 質問等は Twitter もしくはメールでお受けします.

[*12] 特にピッチ変換

23LC1024 について (おまけ)

今回の実験で重要な役割を担った外付け SRAM の 23LC1024 の使い方をまとめます. まずは購入についてですが, 私は RS オンライン[*13]の通販で一個あたり 300 円程度で購入できます.

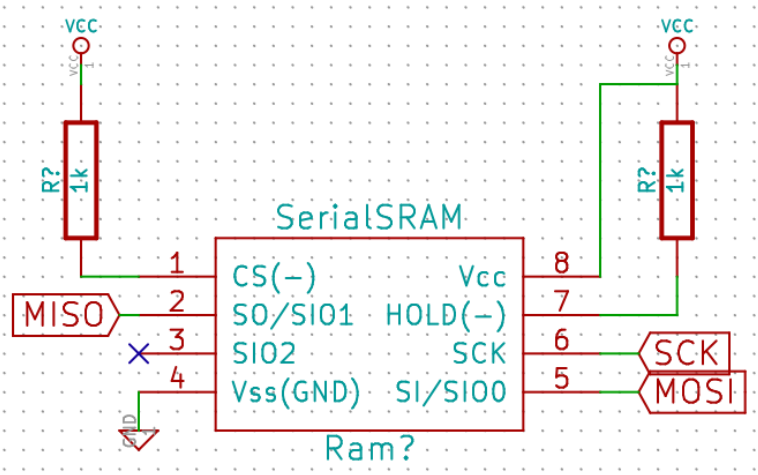

図 5　23LC1024 と Atmega328 の接続 (SPI 通信使用時)

図 5 は SPI 接続で用いる場合の接続方法です. いくつかの使い方があり, 場合によって接続が変わってくるのでまずはデータシートを確認するようにしてください[*14].

実際の使い方

1. 23LC1024 の 1 番ピンの $\overline{\text{CS}}$ ピンを Low にする (起動)
2. 書き込みか読み込みか, 指定する
3. アドレスを指定する

[*13] http://jp.rs-online.com/web/
[*14] 今回の動作モードは正確には Byte モードの SPI 接続です

4. データを書き込む or データを読み込む
5. $\overline{\text{CS}}$ ピンを High にする

2 から 4 の操作を詳しく説明します. 機能の指定で 8bit, アドレスの指定で 24bit, データの受け取り (書き込み) で 8bit を使います. Atmega328P の SPI 通信では一回で最大 8bit のデータの送受信ができるため, 計五回送受信を行うことになります.

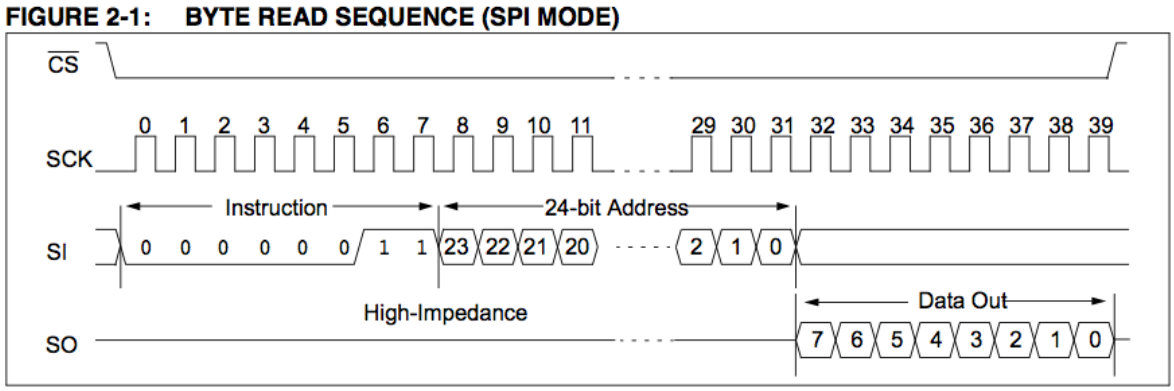

図 6　読み込み時の操作 (データシートより引用)

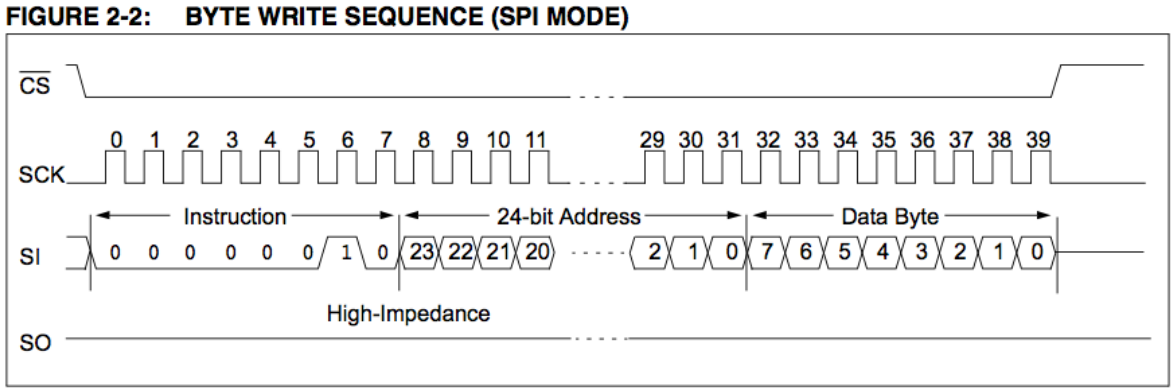

図 7　書き込み時の操作 (データシートより引用)

図 6,7 はデータの送受信のタイミングを示します. 図にもあるように予め

命令として読み込み (0x03), 書き込み (0x02) が決められており, 命令と一致する数字を SPI で送ると動作モードの指定をすることができます．アドレスの指定は後ろから 8bit ずつ行っていきます．23LC1024 ではアドレスは 0x000000 から 0x0FA000 までの間で指定するようにしましょう．次に書き込みの場合は SPI で送信, 読み込みの場合は SPI で受信する[*15]ことでデータのやりとりを行うことができます．書き込み, 読み込みは具体的には以下のように制御します.

```
//SRAM 操作処理 (書き込み)
PORTB=0b00000000;
//Atmega328P の PB2 ピンを SRAM の CS と接続//CS を Low にする
spi_send(0x02);//書き込み動作
spi_send(add_top_w);//0x**----//**を指定
spi_send(add_middle_w);//0x--**--
spi_send(add_low_w);//0x----**
spi_send(ad_buf);
PORTB=0b00000100;//CS を High にする
```

add_ ○ ○_w はそれぞれ 8bit の変数で三つ合わせて一つのアドレスを示します．ad_buf は A/D 変換によって読み込んだ 8bit のデータを保存しておく変数です.

```
//SRAM 操作処理 (読み込み)
PORTB=0b00000000;//CS を Low にする
spi_send(0x03);//読み込み動作
spi_send(add_top_r);//0x**----//**を指定
spi_send(add_middle_r);//0x--**--
spi_send(add_low_r);//0x----**
serial_buf = spi_get();
PORTB=0b00000100;//CS を High にする
```

serial_buf は SPI 通信で送られてくるデータを受ける変数です.

[*15] ダミーデータを送信した後に SPDR レジスタの値を読み込む

SPI の送受信のために以下の様な関数を用意しています.

```
void spi_send(uint8_t spi_data){//8bit データ送信
  uint8_t dummy = 0;
  dummy = SPDR;
  SPDR = spi_data;
  while(!(SPSR&(1<<SPIF)));//転送完了まで待機
  dummy = SPDR;
}


unsigned int spi_get(void){//8bit データ受信
  uint8_t dummy = 0;
  SPDR = dummy;
  while(!(SPSR&(1<<SPIF)));//転送完了まで待機
  return SPDR;
}
```

SPDR はマイコン内蔵のシフトレジスタで SPI 通信によってやりとりされるデータはこのレジスタを経由します.

参考文献

[1] デジタル信号処理の実験
http://elm-chan.org/works/vp/report_j.html
[2] 23LC1024 データシート MICROSHIP 社

タイトル：8bitAVR マイコンによるデジタル信号処理の実験 1
著者：あわざる (Twitter @awazaru0524)
サークル：BrightNotes
発行：2015 年 12 月 31 日 (C89 三日目)
mail: yudai.awashima@gmail.com