Arduino Due で超音波フェーズドアレイ

星 貴之

平成 28 年 4 月 1 日

1. はじめに

超音波フェーズドアレイを実装するには, (1) 高周波 (40 kHz) の信号を (2) 多チャンネルで出力する必要がある. この要件を満たすため, これまでの超音波集束装置 [1] では FPGA を用いていた. FPGA は 1 クロックごとの挙動を記述でき, 50 MHz などのクロックで十分速く動作し, 信号は並列処理され, 数百本の I/O を備えているなど, フェーズドアレイ向きの特徴を備えている. 一方, マイコンはプログラムが CPU で逐次処理され, ひとつの命令に数クロックかかるため遅く, 多くても数十本の I/O しかなく, フェーズドアレイのような多チャンネルを正確に操る用途には使いづらい.

本稿ではマイコンボード Arduino Due [3] (84 MHz クロック搭載) を使って超音波フェーズドアレイの製作に挑戦する. 5 チャンネルの 40 kHz 矩形波を位相制御して出力することを目指す. 信号数が限られているとはいえ, うまくいけば DIY 音響浮揚装置 [2] の焦点距離を可変にするなどに使えるかも知れない.

2. 駆動信号の生成

Arduino Due で 40 kHz 矩形波を出力するにはどうすればよいか. しかも多チャンネル. さらに位相制御.

2·1 tone()

tone 関数を使って超音波距離計を駆動した報告がなされている [4]. しかしこの関数は同時に 1 チャンネルしか出力できないし, Due には実装されていない [5].

2·2 タイマー

タイマーを使って 2 チャンネルの位相制御をした報告もある [6]. しかし我々の目的は 5 チャンネルかそれ以上である. これでは足りない.

2·3 digitalWrite() + delayMicroseconds()

素直に一定時間ごとに出力ピンの値を書き換えるプログラムを書くのはどうか. 可聴域 [7] ならこれでいけるようだが, 超音波帯域では厳しい. 試したところ, 一周期 25 μs (40 kHz) で回る予定のループに数 100 μs かかった. delayMicrosecond 関数は 3μs 未満で動作保証されていない [8]. また digitalWrite 関数も 44 サイクル (524 ns) かかるとのこと [9] (実際には 4 μs かかるという説もある [10]). 出力切り替えは 1 μs 程度で済ませたい. ポート制御の高速化が必要だ.

2·4 高速ポート制御

digitalWrite 関数の中では色々と確認や例外処理が実行されており [11], これらのために処理時間が費やされている. これらを省き, レジスタに直接アクセスすることで高速化を実現する [9] (付録 A 参照). しかしまだチャンネルの数だけ繰り返す手間と時間がかかる.

2·5 同時高速ポート制御

Arduino Due の I/O は, ポート A, B, C, D に組分けされ, 番号が割り振られ, それぞれのポートが 32 bit のレジスタ配列で管理されている. このレジスタ配列に直接アクセスすることで, 複数の出力ピンの値を一括で高速に書き換えることができる [12]. まず使いたいピンが「どのポート」の「何番」かをピン配置表 [13] で調べる. 例えば Arduino Due の 53 番ピンを使いたい場合, これは B.14, すなわち「ポート B」 の「14 番」である. 従ってこれを HIGH にするにはレジスタ配列 REG_PIOB_ODSR の 15 番目のレジスタに 1 を入れる (10 進数で表すと 2^{14} を代入する).

3. 試作

駆動信号を生成する目途が立ったので, 実際に作ってみた (次頁 Figs.1-5).

3·1 開発環境

ウェブサイト [14] を参考に Arduino 開発環境を PC にインストールする (本稿執筆時点での最新版は 1.6.8).

3·2 駆動回路

Arduino Due が出力する信号は 0 〜 3.3 V しかないため, 超音波振動子を駆動するためには信号増幅が必要である. プッシュプル増幅器によって 0 〜 24 V に増幅し, 1 次ハイパスフィルタで DC 成分をカットして −12 〜 12 V を得る. また VIN ピンに電源 9V を供給する. 以下に部品リストを示す.

- 電圧増幅器 L293DD
- 抵抗 100 kΩ
- コンデンサ 1 μF
- 電圧変換器 (24V → 9V) TSR 0.5-2490
- 電源 (24V, 150W) RSP-150-24

3·3 超音波振動子アレイ

超音波振動子 (T4010B4, 日本セラミックス) 100 個を用いた. 10 × 10 に配列し, 一方向は曲面配置で, もう一方向は位相制御で集束させることとした. 焦点位置を中央に固定することで駆動信号は左右対称でよくなり, すなわち 5 チャンネルで足りる.

曲面は曲率半径 (＝焦点距離) 7 cm となるよう設計した. レーザー加工機で厚さ 2 mm のアクリル板を切ってアクリサンデーで接着することにより製作した.

3·4 プログラム

付録 B に今回のプログラムを示す. メインループが回る速度は掛け算, 足し算, 出力の処理時間により決定される. PHASE_DIV, CYCLE_NUM, OFFSET は, 出力波形が 40 kHz, 200 Hz, デューティ比 50% になるようオシロスコープで見ながら調整した.

3·5　動作試験

変調せず超音波を出し続ける (DC 出力) と, 最初の 15 秒程度は「ピーーー」という高周波音が聞こえ, それから「ブブブブ」というノイズが混ざり始めた. DC 出力の超音波焦点を電子秤に照射したところ, 最初は 0.60 g と表示されるものの, ノイズが混ざり始めると 0.30 g に低下した.

200 Hz で変調 (ON/OFF) した場合には時間が経過しても良好に超音波出力しているように見えた. 電子秤は 0.30 g を示した. これは DC 出力が当たる時間が半分になったためである. 焦点に付箋をかざすと, 超音波の放射圧により持ち上げられる様子が観察された.

DC 出力で動作が不安定になる原因は不明. 電圧増幅器の発熱の影響は考えられるが, 200 Hz 変調で安定することの説明がつかない. 謎だ.

4.　おわりに

Arduino による超音波フェーズドアレイは数チャンネルならギリギリできるが, それ以上は難しいかも, という印象. プログラムに最適化, 高速化, 安定化の余地があるかもしれないが, 現時点では分からない.

参考文献

[1] 星貴之: 非接触作用力を発生する小型超音波集束装置の開発, 計測自動制御学会論文集, vol. 50, no. 7, pp. 543-552, 2014.

[2] 星貴之: DIY 音響浮揚装置を作ってみた (第 2 報), エンタテインメントコンピューティング 2015 論文集, pp. 100-106, 札幌, 9 月 25-27 日, 2015.

[3] Arduino Due, https://www.switch-science.com/catalog/1096/.

[4] くらんべりー: arduino で超音波センサーを使ってみる, http://cranberrytree.blogspot.jp/2014/05/arduino.html.

[5] tone, https://www.arduino.cc/en/Reference/Tone.

[6] dejko1: Arduino generating two fast phase offset signals, http://www.rei-labs.net/arduino-generating-two-fast-phase-offset-signals/.

[7] きゃべログ: tone 関数を使わずに原始的な方法でスピーカから音を鳴らす, 簡易シーケンサを作ってみた, http://cabbalog.blogspot.jp/2014/09/squarewave.html.

[8] delayMicroseconds, https://www.arduino.cc/en/Reference/DelayMicroseconds.

[9] macsbug: 32ch Digital Analyzer for Arduino DUE + TFT 5int Touch Display, https://macsbug.wordpress.com/2014/09/.

[10] 栄光学園物理研究部: Arduino で正弦波を出すための小ワザ, http://blog.livedoor.jp/eikophys/archives/51873797.html.

[11] Arduino で遊ぶページ / digitalWrite(), http://garretlab.web.fc2.com/arduino/inside/arduino/wiring_digital.c/digitalWrite.html.

[12] SAM3X8E (Arduino Due) Pin IO registers, http://arduino.stackexchange.com/questions/9117/sam3x8e-arduino-due-pin-io-registers.

[13] Arduino Due ピン配置表, http://www.musashinodenpa.com/arduino/lib/Duepinout.pdf.

[14] Arduino で遊ぶページ / インストール, http://garretlab.web.fc2.com/arduino/introduction/installation/index.html.

Fig.1 アクリサンデーを溝に流し込むと, ちょっと隙間があってもアクリルが溶けてくっついてくれる.

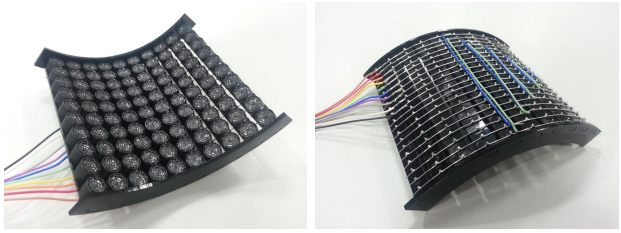

Fig.2 振動子アレイ. 配線は信号 5 本と GND 5 本.

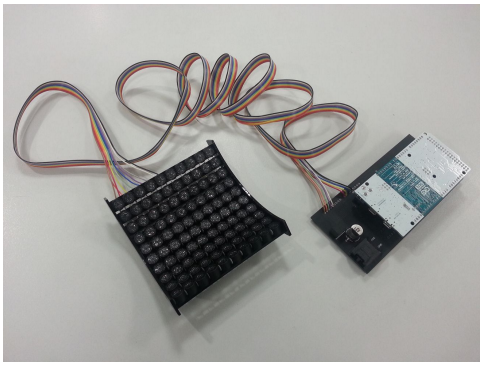

Fig.3 振動子アレイと駆動回路. 他に 24 V 電源.

Fig.4 電子秤に 200 Hz で変調した超音波を照射. このときは調子がよくて 0.33 g を記録.

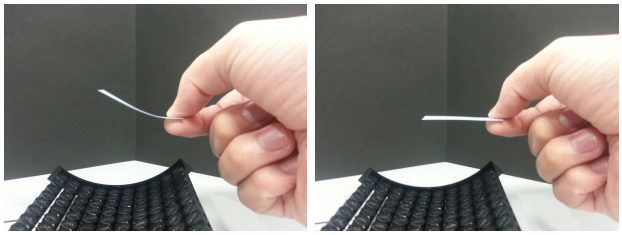

Fig.5 付箋. (左) 超音波焦点, (右) 焦点からずれた位置.

A　高速ポート制御

ウェブサイト [9] より関数定義部分を引用. digitalWrite 関数と digitalRead 関数を高速化した POUT と PIN.

```
//============================================================================
inline void POUT(int pin, boolean val){          // POUT:Port OUTPUT
  if(val) g_APinDescription[pin].pPort ->        // digitalWriteDirect
        PIO_SODR = g_APinDescription[pin].ulPin; //
  else    g_APinDescription[pin].pPort ->        //
        PIO_CODR = g_APinDescription[pin].ulPin; //
}                                                //
//============================================================================
inline int PIN(int pin){                         // PIN:Port INPUT
  return !!(g_APinDescription[pin].pPort ->      // digitalReadDirect
        PIO_PDSR & g_APinDescription[pin].ulPin);//
}                                                //
//============================================================================
```

B　サンプルプログラム

超音波振動子の駆動信号 US とドライバのイネーブル信号 EN を生成するサンプルプログラム.

PHASE_DIV は位相制御の解像度. 超音波の一周期 (25 μs) の間に loop 関数が回る回数に依存. EN が不要であれば, US をポート C にまとめるなどしてもっと早く loop 関数を回せるかも. CYCLE_NUM は定期的に超音波を ON/OFF して 200 Hz の振動刺激を作るため. OFFSET は本来不要だが, 40 kHz 矩形波のデューティ比がなぜかずれるのを調節して 50% にするため.

```
/*--------------------------------*/
/* Arduino Ultrasonic Phased Array */
/*--------------------------------*/
#define F_LEN 7.0     // Focal length 7.0cm
#define W_LEN 0.85    // Wavelength 0.85cm
#define CH_NUM 5      // Number of channels
#define PHASE_DIV 27  // Parameter for driving frequency (40kHz)
#define CYCLE_NUM 125 // Parameter for modulation frequency (200Hz)
#define OFFSET 3      // Parameter for 50% duty ratio

int US[CH_NUM] = {44, 42, 40, 38, 36}; // Ultrasound output ports
// US[0] : 44 : C19 : 2^19 = 524288
// US[1] : 42 : A19 : 2^19 = 524288
// US[2] : 40 : C8  : 2^8  = 256
// US[3] : 38 : C6  : 2^6  = 64
// US[4] : 36 : C4  : 2^4  = 16
int US_NUM[CH_NUM] = {524288, 524288, 256, 64, 16};

int EN[CH_NUM] = {53, 47, 45, 39, 37}; // Driver enable ports
// EN[0] : 53 : B14 : 2^14 = 16384
// EN[1] : 47 : C16 : 2^16 = 65536
// EN[2] : 45 : C18 : 2^18 = 262144
// EN[3] : 39 : C7  : 2^7  = 128
// EN[4] : 37 : C5  : 2^5  = 32
int EN_NUM[CH_NUM] = {16384, 65536, 262144, 128, 32};
```

```
int waveform[CH_NUM][PHASE_DIV]; // Driving signals
int pdiv;                   // [0, (PHASE_DIV-1)]
int cnt;                    // Invert EN at every CYCLE_NUM cycles for vibration
int ch;                     // Number of channels
boolean enable;             // 1: ON / 0: OFF

void setup() {
  // put your setup code here, to run once:
  double d;
  int phase;
  for (ch=0; ch<CH_NUM; ch++){
    pinMode(EN[ch], OUTPUT);
    digitalWrite(EN[ch], HIGH);
  }
  for (ch=0; ch<CH_NUM; ch++){
    pinMode(US[ch], OUTPUT);
    digitalWrite(US[ch], LOW);
  }
  for (ch=0; ch<CH_NUM; ch++){
    d = sqrt( ((double)ch)*((double)ch) + F_LEN*F_LEN ); // Distance
    while( d > W_LEN){
      d -= W_LEN;
    }
    phase = (int)(d * PHASE_DIV / W_LEN); // Calculate phase based on distance
    for (pdiv=0; pdiv<PHASE_DIV; pdiv++){
      if(phase < PHASE_DIV/2){
        if(phase <= pdiv && pdiv < phase + PHASE_DIV/2 + OFFSET){
          waveform[ch][PHASE_DIV -1 -pdiv] = 1;
        }else{
          waveform[ch][PHASE_DIV -1 -pdiv] = 0;
        }
      }else{
        if(phase - PHASE_DIV/2 < pdiv && pdiv <= phase + OFFSET){
          waveform[ch][PHASE_DIV -1 -pdiv] = 0;
        }else{
          waveform[ch][PHASE_DIV -1 -pdiv] = 1;
        }
      }
    }
  }
  cnt = 0;
  enable = false;
}

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

```
  if(cnt < CYCLE_NUM){ // 200Hz modulation
    cnt = cnt + 1;
  }else{
    cnt = 0;
    enable = !enable; // invert
//  enable = true;    // noninvert for debug
  }
 for(pdiv=0; pdiv<PHASE_DIV; pdiv++){ // 40kHz
    if(enable == true){
      REG_PIOA_ODSR = US_NUM[1] * waveform[1][pdiv];
      REG_PIOB_ODSR = EN_NUM[0];
      REG_PIOC_ODSR = EN_NUM[1] + EN_NUM[2] + EN_NUM[3] + EN_NUM[4]
          + US_NUM[0] * waveform[0][pdiv] + US_NUM[2] * waveform[2][pdiv]
          + US_NUM[3] * waveform[3][pdiv] + US_NUM[4] * waveform[4][pdiv];
    }else{
      REG_PIOA_ODSR = 0;
      REG_PIOB_ODSR = 0;
      REG_PIOC_ODSR = 0;
    }
  }
}
```