R4年度次世代技術活用人材育成事業
技術修得コース　実習座学（IoT）
第2回

茨城県産業技術イノベーションセンター
IT・マテリアルグループ

目次

【前半】
- 交流電流値算出のアプローチ
- 測定システム構成図
- AD変換の回数
- 割り込み処理
- 処理の時系列
- 使用する電流センサの原理
- 電流センサの特性
- 分圧の法則
- 抵抗ボードと*OFFSET*値
- *OFFSET*値の算出
- 負荷抵抗R_L＝30Ωとすると・・・
- 電流値（実効値）の算出①、②

【後半】
- プログラムの構成
- 実行ファイルの生成
- フローチャート
- データ型の範囲と配列
- プログラムの解説①～③
- 割り込み関数の詳細

【参考資料】
- 実効値とは？
- mcp3208の通信プロトコル
- 回路図①、②
- WiringPiの使用

はじめに

【本研修における目標】
電流センサから取得したアナログ値をデジタル値に変換し、そこから電流値（実効値）を算出する。

【本研修におけるポイント】
・A/D変換器の制御コマンドを理解していること。

・電流センサの仕様を理解すること。

・A/D変換器 - Raspberry Pi間でSPI通信を行う公開ライブラリを活用すること。

・割り込み処理を活用して一定時間ごとにA/D変換を行うこと。

・電流値（実効値）を算出する処理を行うこと。

前半

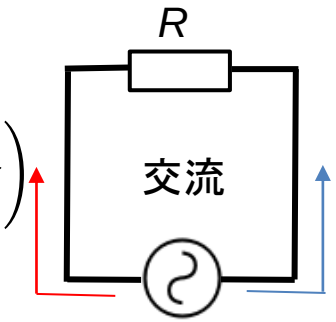
交流電流算出のアプローチ

交流とは？

⇒電圧と電流の向きと大きさが周期的に変化する電気の流れ。（家庭用コンセントなど）

$i = I_o \sin \omega t$

$\left(i = \frac{V_0}{R} \sin \omega t \right)$



$v = V_o \sin \omega t$

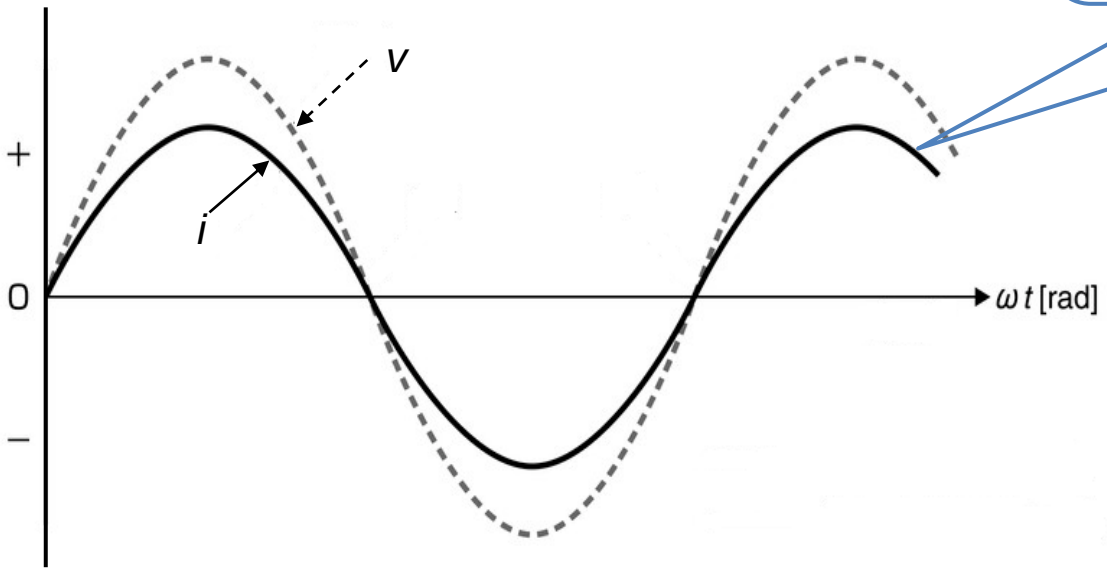


図1　交流波形

この交流電流の値を算出したい！！
ただし、瞬時値ではなく交流の計算で一般的に使用される実効値を算出する。

【アプローチ】

Step1：A/D変換の頻度やタイミングを決める

Step2：電流センサからの出力電圧を取得し、A/D変換を行う

Step3：A/D変換値から電流値（実効値）を算出

測定システム構成図

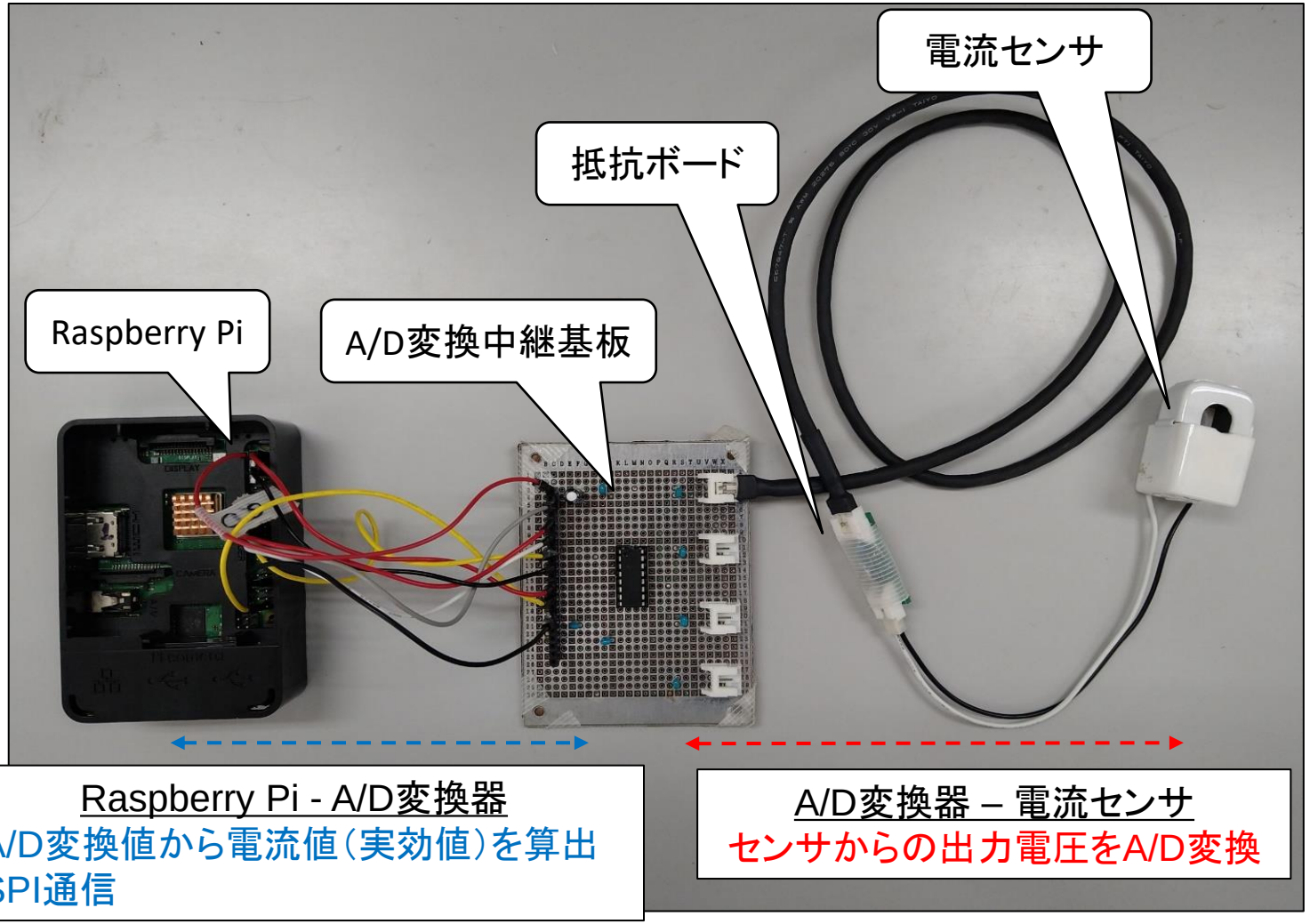


図2　交流電流値見える化システム構成図

A/D変換の回数

本研修では交流電源周波数を50Hz（東日本）で考察する。
周波数とは、電気振動などの現象が単位時間（Hzの場合は1秒）あたりに繰り返される回数。
周波数が50Hzの場合の1周期あたりの時間は式（1）より20m(sec)となる。

$$T = \frac{1}{f} \quad (1)$$

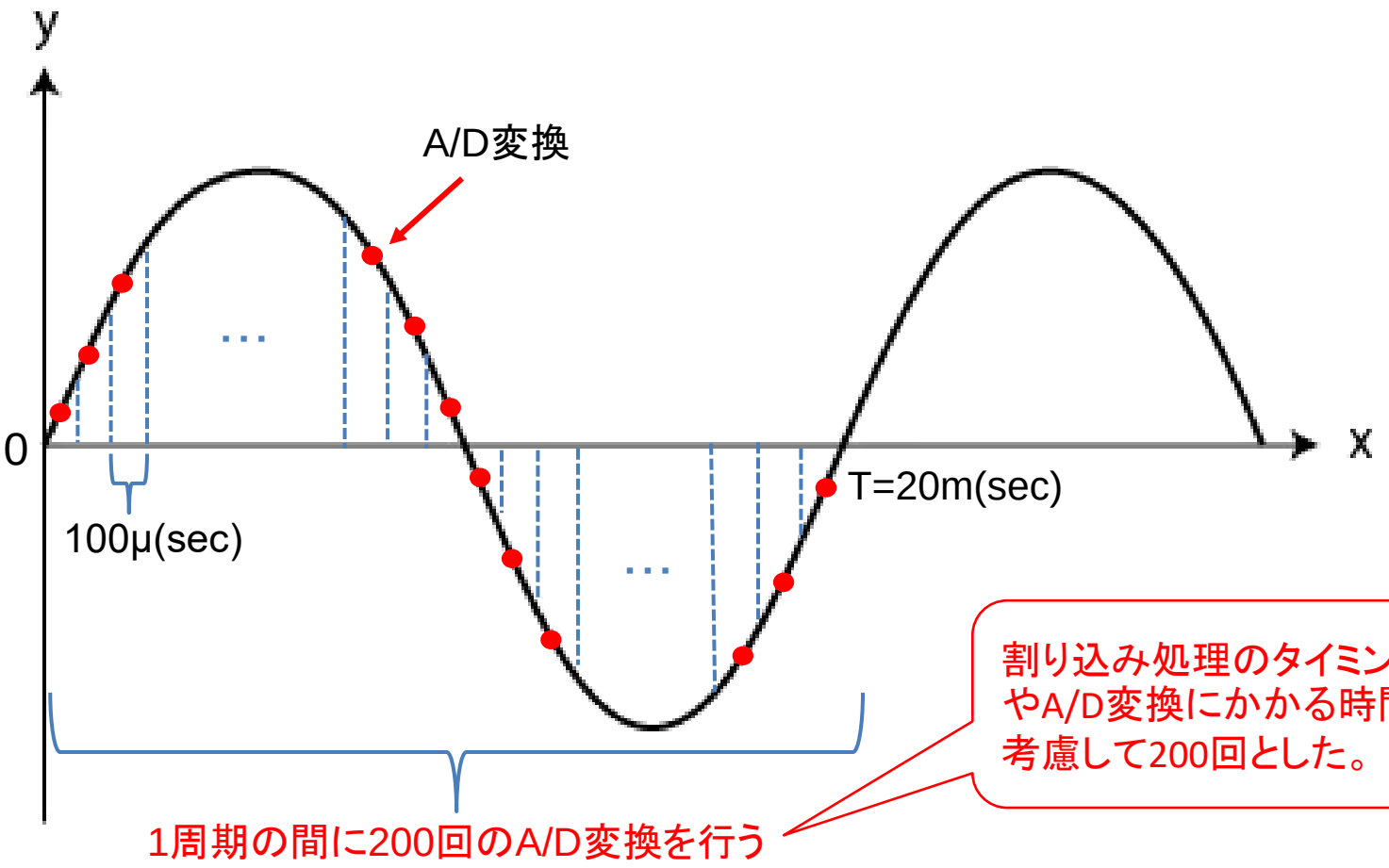


図3　1周期あたりのA/D変換回数

割り込み処理

割り込み処理とは、外部からの割り込み要求を受けたとき、実行しているプログラムを一時中断して他のプログラムを実行すること。

A/D変換を行うタイミングとして100μ(sec)毎に割り込み処理を発生させ、呼び出した関数(alarmWakeup)の中でA/D変換を行っている。

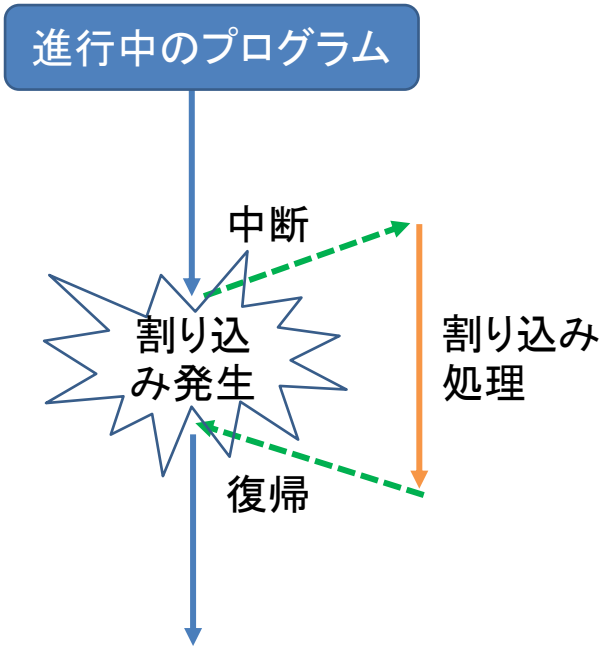


図4　割り込み処理のイメージ

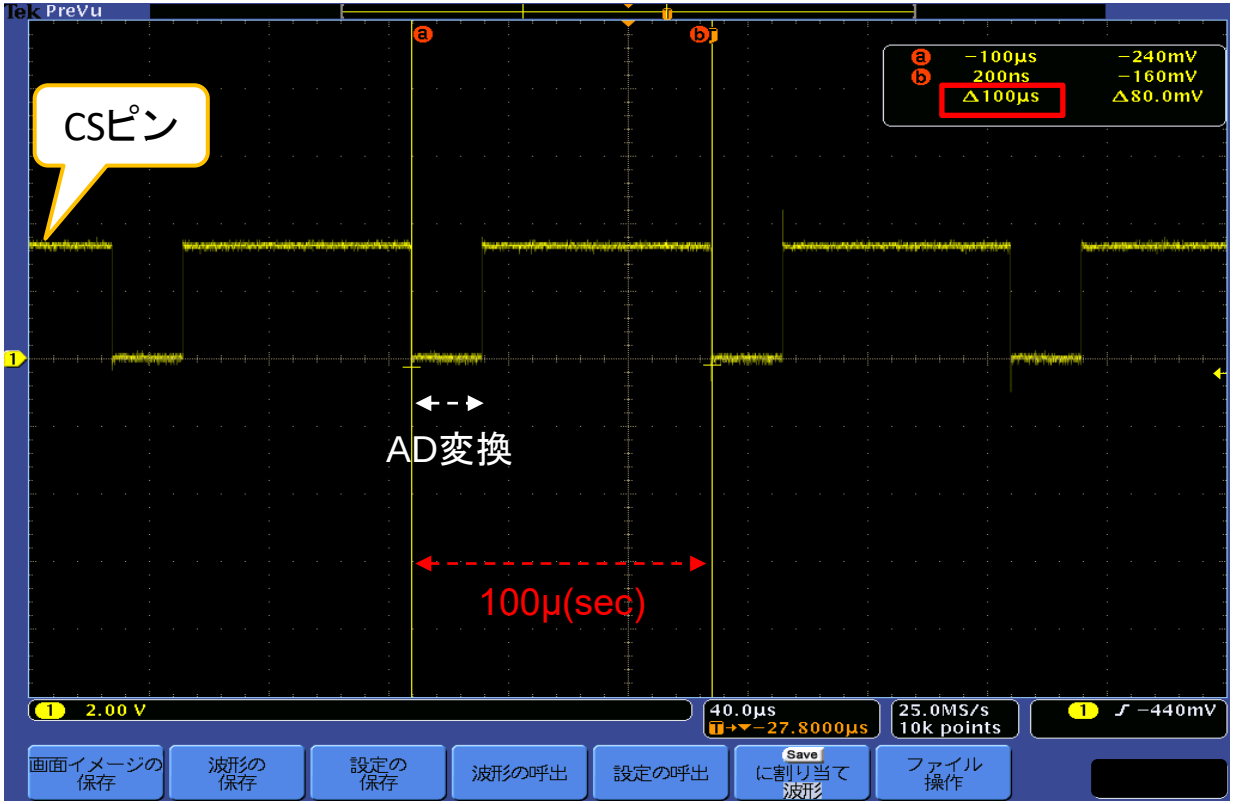


図5　MCP3208のCSピンの波形

処理の時系列

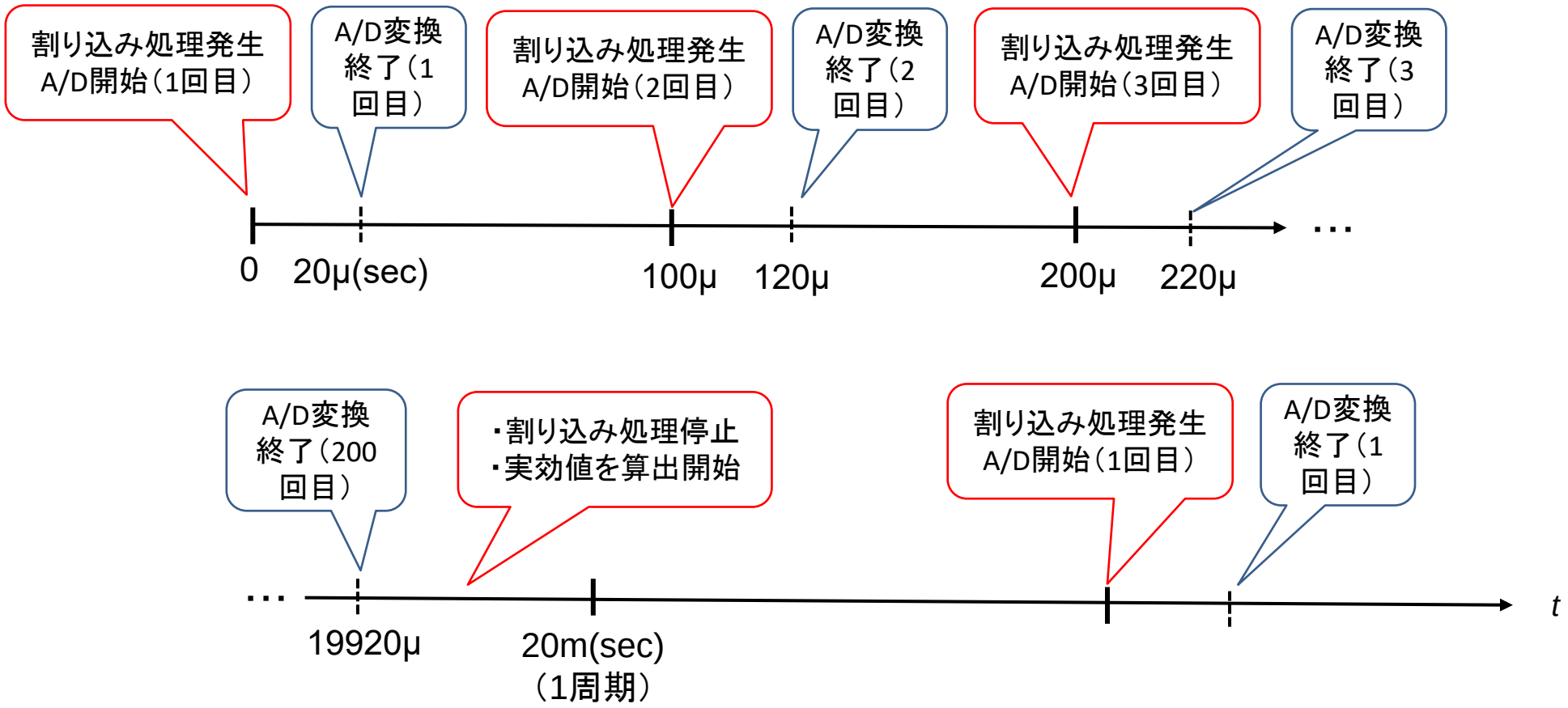


図6　処理の時系列

使用する電流センサの原理

CT方式：測定電流を巻線比に応じた2次電流に変換する原理

【特徴】
- 電源が不要
- 安価
- 交流（AC）のみ測定可能

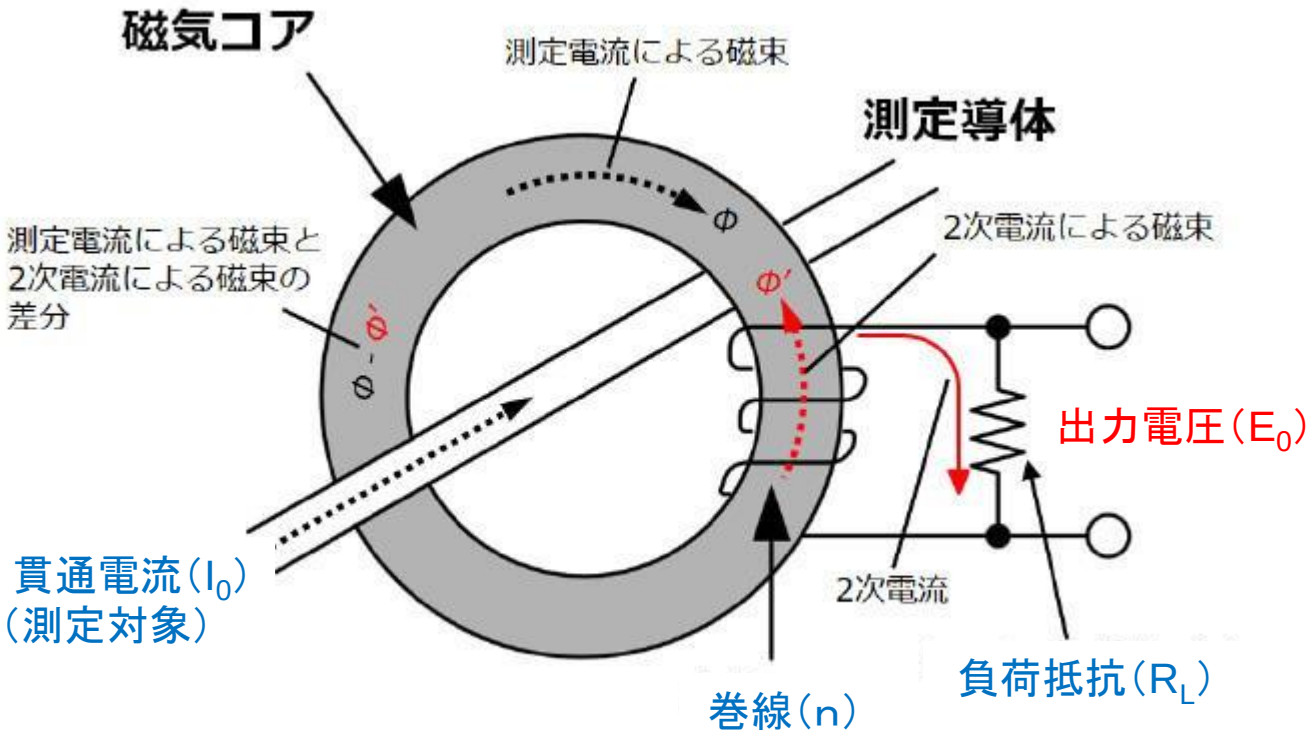


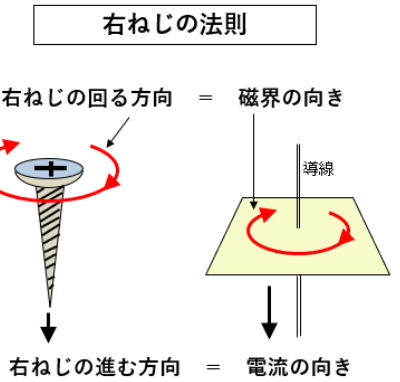
図7　電流センサの原理

【原理】

①1次側の測定電流の交流（AC）により、貫通電流による磁束（Φ）が磁気コアに誘導される。
（※磁束：その場における磁界の強さを、線の束で表したもの。）

②発生した磁束（Φ）を打ち消す向きに誘導起電力が発生し、2次電流が流れる。

③2次電流は負荷抵抗（R_L）に流れ、出力電圧（E_0）が発生する。この電圧は貫通電流（I_0）に比例する。



参考ページ：https://www.hioki.co.jp/jp/products/listUse/?category=39

電流センサの特性

図8　電流センサ

表1　主な仕様

メーカー	U_RD
型式	CTL-10-CLS
適用電流	80Arms (実効値)
公称変流比	3000:1
測定方式	CT方式

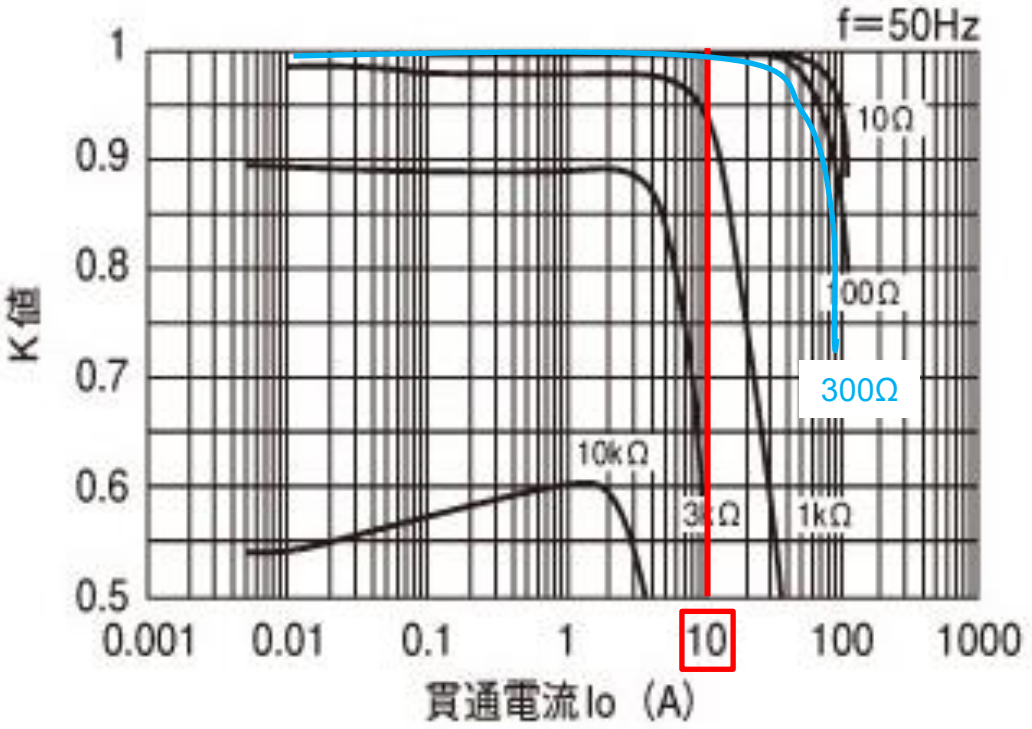


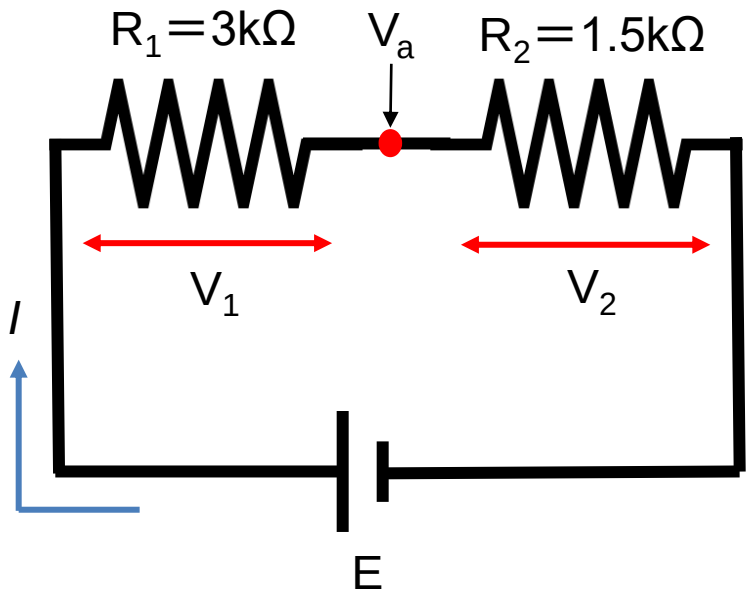
図9　結合係数（K）特性

センサの出力電圧（E_0）、貫通電流（I_0）、結合係数（K）、負荷抵抗（R_L）、結合係数（K）、交流比（n）から式（2）が成り立つ。

$$E_0 = \frac{K \times I_0 \times R_L}{n} \quad (2)$$

本研修ではI_0＝10（A）、K＝1、R_L=300（Ω）、n＝3000として式（2）よりE_0＝±1.0（V）となる。

分圧の法則



(1)回路に流れる電流Iは、

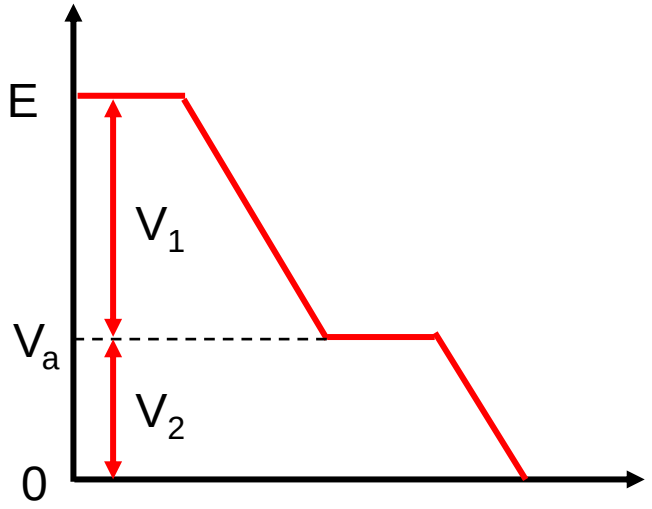
$$I = \frac{E}{R_1 + R_2}$$

(2)抵抗R_1にかかる電圧V_1は、$V_1 = R_1 \times I$ なので

$$V_1 = \frac{R_1}{R_1 + R_2} E$$

(3)抵抗R_2にかかる電圧V_2は、$V_2 = R_2 \times I$ なので

$$V_2 = \frac{R_2}{R_1 + R_2} E$$



抵抗ボードとOFFSET値

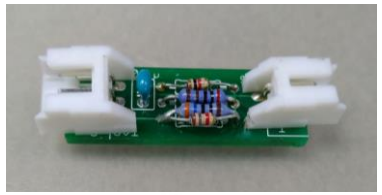


図10　抵抗ボード

V_a=1.67V

【分圧の法則】

$$V_a = \frac{1.5}{3.5 + 1.5} \times 5 = 1.666\cdots[V]$$

図11　回路図

$$V_{OUT} = 1.67 \pm E_0[V]$$

この抵抗ボードは、5Vを1/3分割したOFFSET電圧（1.67V）に電流センサの出力電圧（E_0）を加えた値が、A/D変換器の入力電圧となるように設計されている。（貫通電流が0の場合は常にE_0＝1.67Vが出力）

☆A/D変換器への入力電圧がマイナスにならないように設計している。

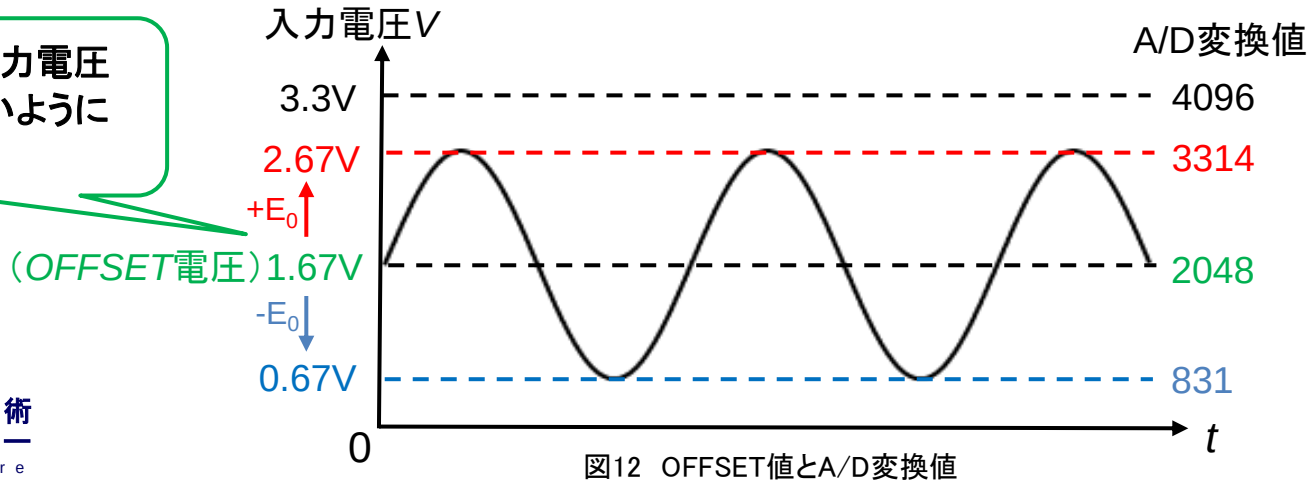


図12　OFFSET値とA/D変換値

OFFSET値の算出

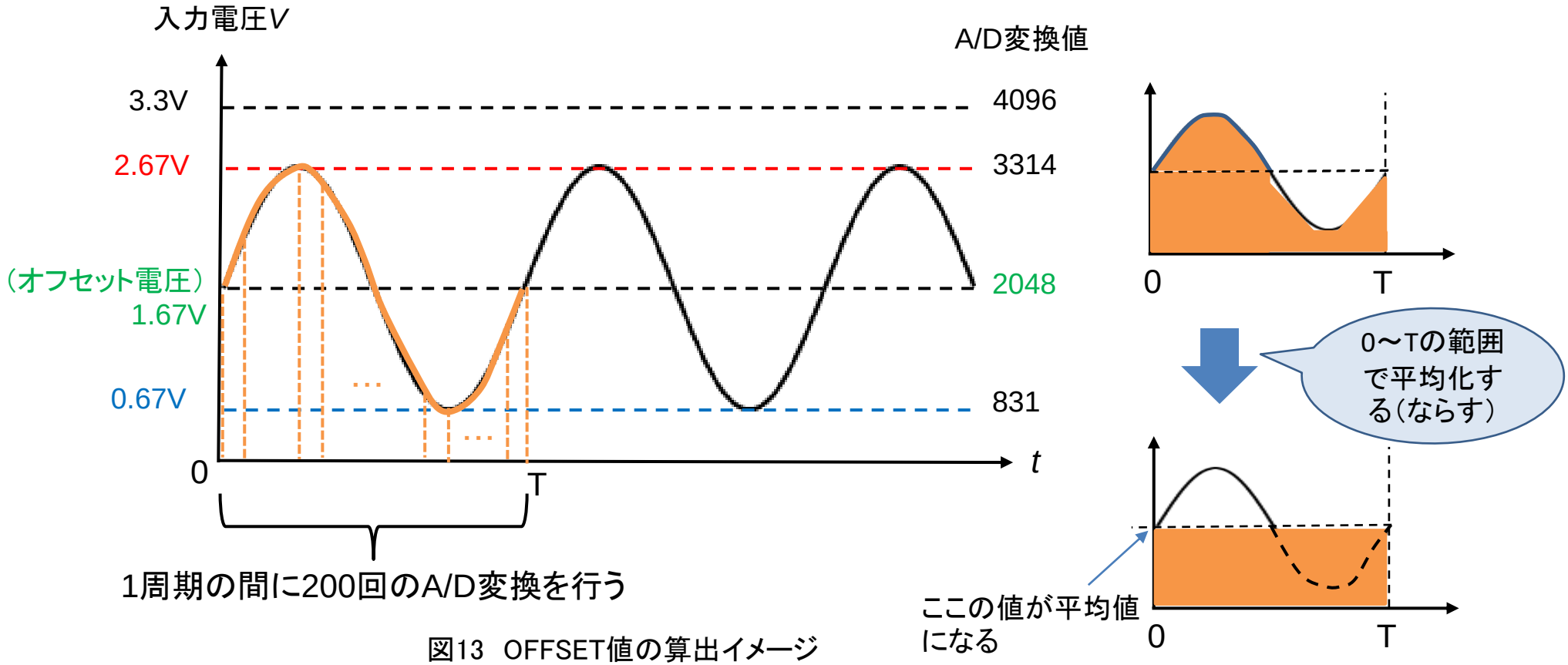


図13　OFFSET値の算出イメージ

☆*OFFSET*値は1周期分のA/D変換値の平均値

$$OFFSET = \frac{2048 + 2051 + \cdots + 3314 + 3308 + \cdots + 838 + 831 + \cdots 2042 + 2048}{200} = 2048$$

負荷抵抗R_L＝30(Ω)とすると･･･

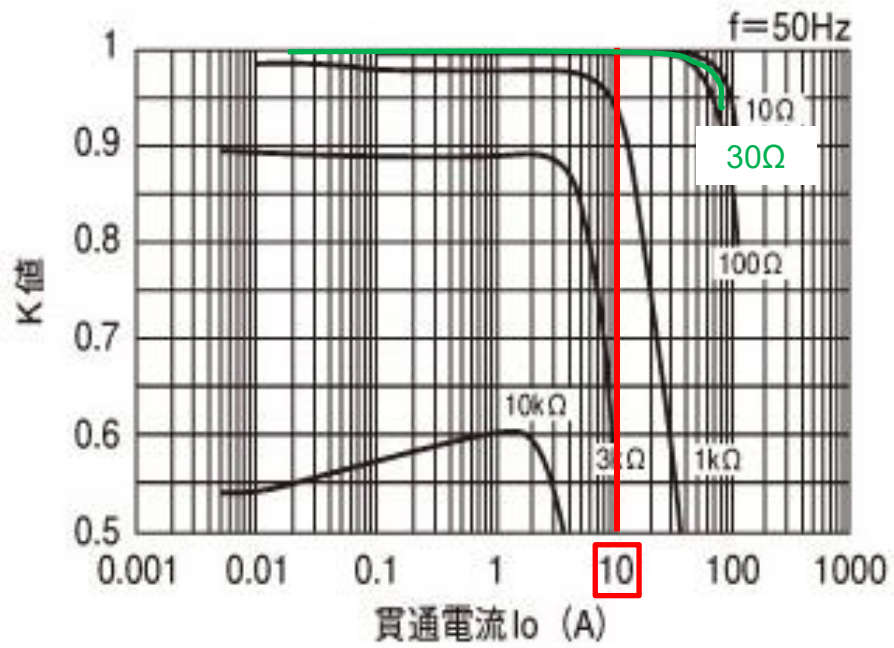
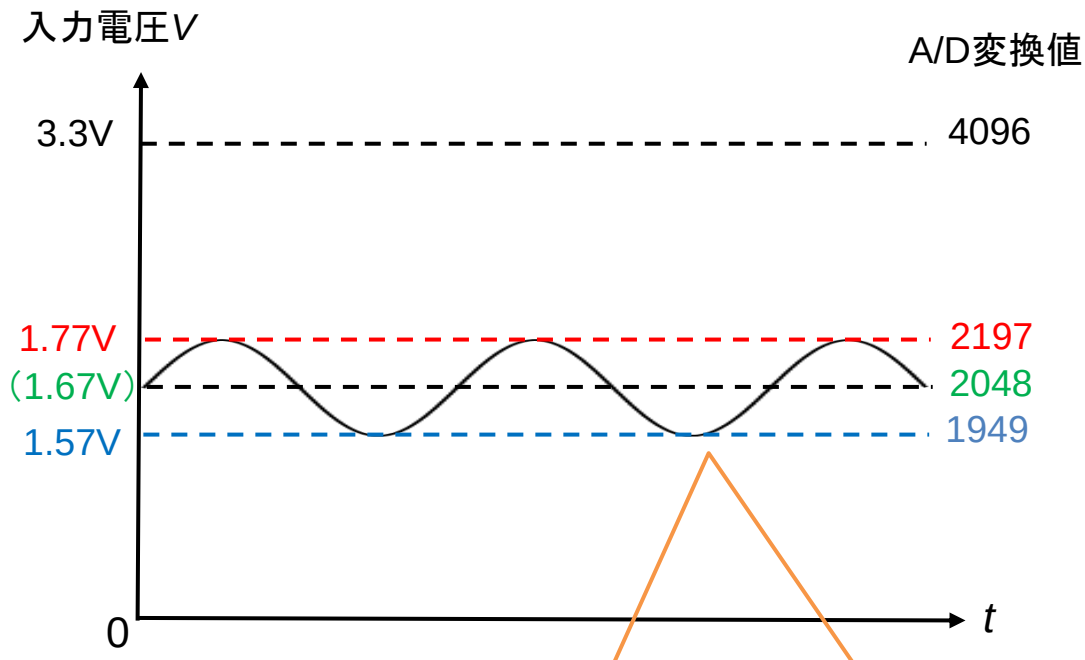


図9　結合係数(K)特性

$$E_0 = \frac{K \times I_0 \times R_L}{n} \quad (2)$$

式(2)よりR_L=10(Ω)とすると
E_0＝±0.1(V)



A/D変換器の入力ピン(ch0)に入力される電圧は1.57～1.77(V)となりA/D変換値は1949～2197となる。これはA/D変換器の12ビットの分解能を有効活用できていない！

入力電圧を可能な限り0～3.3(V)の範囲でフル活用することが精度よくA/D変換するポイント

電流値（実効値）の算出①

センサの貫通電流I_0は式（2）より下記のように求められる。

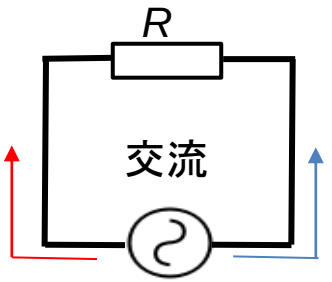
$$I_0 = \frac{E_0 \times n}{K \times R_L} = E_0 \times \frac{n}{R_L} \quad (3)$$

式変形（K=1）

$$E_0 = \frac{K \times \mathrm{I}_0 \times \mathrm{R}_\mathrm{L}}{n} \quad (2)$$

また、出力電圧E_0はある瞬間でのA/D変換値をx、オフセット値を$OFFSET$とすると式（4）が成り立つ。
このときA/D変換値の入力電圧は3.3Vで分解能は4096（12bit）である。

$$E_0 = (x - OFFSET) \times \frac{3.3}{4096} \quad (4)$$



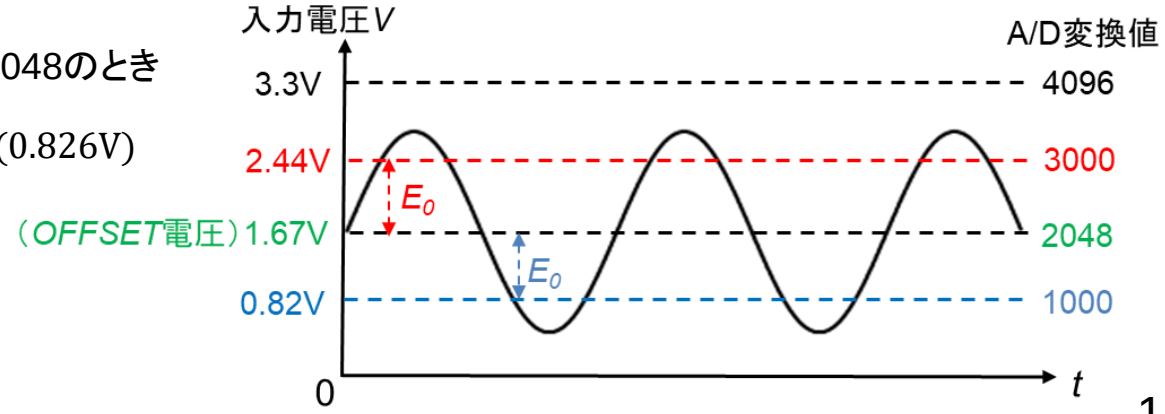
$$v = V_o \sin \omega t$$

（例1）ある瞬間のA/D変換値 x＝3000、OFFSET=2048のとき

$$E_0 = (3000 - 2048) \times \frac{3.3}{4096} = +0.767[\mathrm{V}]\ (2.447\mathrm{V})$$

（例2）ある瞬間のA/D変換値 x＝1000、OFFSET=2048のとき

$$E_0 = (1000 - 2048) \times \frac{3.3}{4096} = -0.844[\mathrm{V}]\ (0.826\mathrm{V})$$



電流値（実効値）の算出②

式（3）、（4）よりある瞬間での電流値（瞬時値）i は式（5）になる。

$$i = (x - OFFSET) \times \frac{3.3}{4096} \times \frac{n}{R_f} \quad (5)$$

一般的に交流の大きさを表す値で実効値があり、瞬時値 i の2乗を1周期にわたって平均したものの平方根の値となっている。実効値をI_Eとすると式（6）になる。

$$I_E = \sqrt{\frac{1}{T}\int_0^T i^2 dt} \quad (6)$$

式（6）より1周期での実効値を算出することができる。実際のプログラムでは上述の処理を繰り返し実行して連続で実効値を算出している。

後半

プログラムの構成

本研修で使用するプログラムファイルは下記の3つ。

mcp3208SpiTest.cpp
　⇒電流値の算出を行うプログラムが記載されている。(←ここを解説！)
mcp3208Spi.h
　⇒A/D変換器と接続するデバイスがSPI通信を行うためのクラスの定義がまとめられている。
mcp3208Spi.cpp
　⇒mcp3208Spi.hで定義されているクラスの詳細が記載されている。

〇コンパイルとは？
高級言語で書いたプログラムをコンピュータが実行可能な形式のプログラムに翻訳すること。
※高級言語：人間が理解しやすい命令や構文規則を備えたプログラミング言語の総称

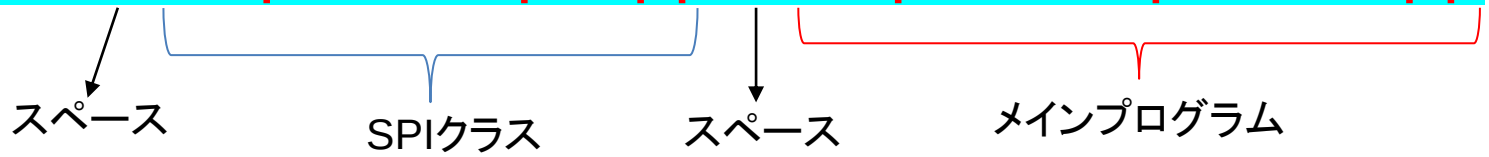
実行ファイルの生成

○実際にコンパイルしてみる

Raspberry Piにはgcc(GNU Compiler Collection)が標準で入っていて、その中のC++のコンパイラであるg++が使用できる、
※gcc:GNU(グニュー)プロジェクトが開発及び配布している、様々なプログラミング言語のコンパイラ集。

○実行ファイルを作成するコマンド

```
~$ g++ ␣ mcp3208Spi.cpp ␣ mcp3208SpiTest.cpp
```



○コンパイルが成功すると実行ファイル「a.out」が生成される。実行する際は下記のコマンドを入力。

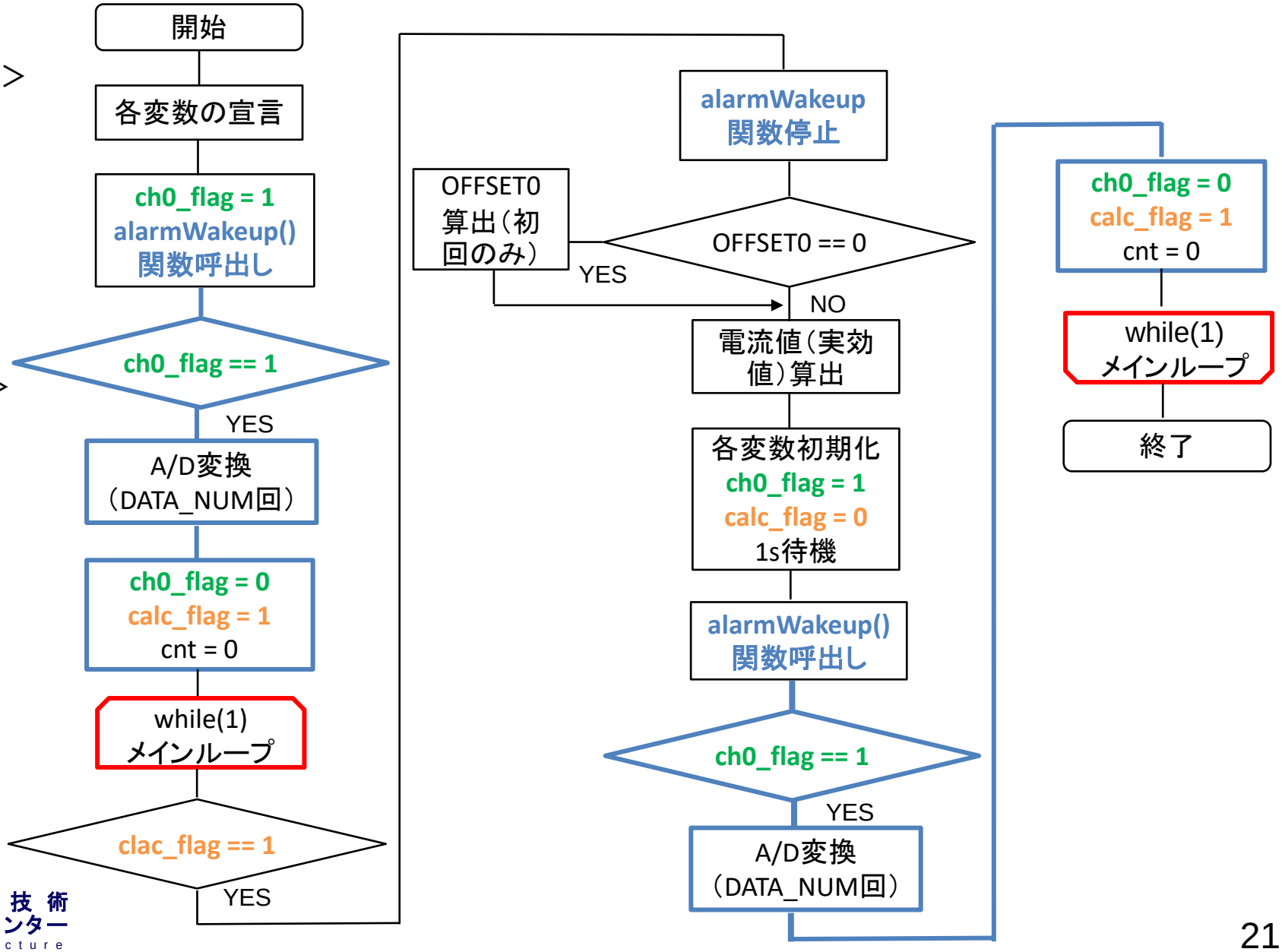
```
-$ ./a.out
```

フローチャート

＜フラグの名称と説明＞
ch0_flag
1: A/D変換開始
0: A/D変換停止

calc_flag
1: 実効値算出を開始
0: 実効値算出を終了

＜割り込み処理関数＞
alarmWakeup()
A/D変換を実施



データ型の範囲と配列

〇データ型・・・C言語やC++では扱う変数の値の範囲や種類に応じてデータの型を指定する。

種類名	バイト	値の範囲	その他
int	4（32ビット）	-2,147,483,648～ 2,147,483,647	整数型
unsigned int	4（32ビット）	0～4,294,697,295	整数型
float	4（32ビット）	1.175494e-38～ 3.402823e+38	小数点型
char	1（8ビット）	-128～127	文字型
unsigned char	1（8ビット）	0～255	文字型

〇配列・・・同じ型のデータをひとまとまりにして操作しやすくするためのもの。

書式⇒ データ型 ␣ 配列名[要素数];

unsigned char data_s1;
unsigned char data_s2;
unsigned char data_s3;

⇨ unsigned char data_s[3];

data_s[1] 〇〇〇〇 〇〇〇〇
data_s[2] 〇〇〇〇 〇〇〇〇
data_s[3] 〇〇〇〇 〇〇〇〇

配列を使って書くと・・・

要素数分の8ビットの箱ができるイメージ

プログラムの解説①

```
1  #include "mcp3208Spi.h"
2  #include <signal.h>
3  #include <math.h>
4
5  #define DATA_NUM 600                  //データの計測点数
6  //#define n 3000                      //センサ交流比
7  //#define R 300                       //中継基板抵抗値
8  #define X 10.0                        //センサ交流比/中継基板抵抗値
9  #define scale_factor 3.3/4096.0       //スケールファクター
10
11 void alarmWakeup(int sig_num);        //割り込み関数宣言
12
13 int ch0_flag = 0;                     //ch0制御フラグ
14 int calc_flag = 0;                    //計算フラグ
15 int i = 0;                            //AD変換回数のカウント変数【メイン処理用】
16 int cnt = 0;                          //AD変換回数のカウント変数【割り込み関数用】
17 float OFFSET0 = 0.0;                  //ch0のオフセット値格納変数
18 float sum = 0.0;                      //AD変換値の加算値格納変数
19 float cur = 0.0;                      //電流値(瞬時値)格納変数
20 float sum_cur = 0.0;                  //瞬時値2乗の加算値格納変数
21 float Ie = 0.0;                       //電流値(実効値)格納変数
22 int a2dVal0[DATA_NUM];                //ch0のAD変換値格納配列
23 unsigned char data_s[3];              //AD変換のch制御ビット格納配列
24
25 mcp3208Spi a2d("/dev/spidev0.0", SPI_MODE_0, 2000000, 8);
26
27 using namespace std;
28
```

①ライブラリの読み込み
mcp3208Spi.h
⇒ SPI通信で必要。
signal.h
⇒ 割り込み関数で必要。
math.h
⇒ 平方根などの計算で必要。

②各変数の宣言

③SPI通信を行う際の設定

SPIのファイル	/dev/spidev0.0を指定
SPI通信モード	モード0を指定
SPIクロック周波数	2MHzを指定
ビット数	8ビットを指定

プログラムの解説②

```
29
30  int main(void)       //メイン関数
31  {
32      signal(SIGALRM, alarmWakeup);
33      ch0_flag = 1;
34      ualarm(1, 100);      //void alarmWakeup関数を開始
35
36      while(1)             //メインループ
37      {
38          if(calc_flag == 1)
39          {
40              ualarm(0,0);              //void alarmWakeup関数を停止
41              if(OFFSET0 == 0.0)        //オフセット値計算(初回のみ)
42              {
43                  for(i=0;i<DATA_NUM;i++)
44                  {
45                      sum += (float)a2dVal0[i];
46                  }
47                  usleep(100);
48                  OFFSET0 = sum /(float)DATA_NUM;
49                  cout << "OFFSET0: " << OFFSET0 << endl;
50
51              }
52              for(i=0; i<DATA_NUM; i++)
53              {
54                  cur = ((float)a2dVal0[i] - OFFSET0)*(scale_factor)*X;
55                  sum_cur += cur*cur;
56              }
57
58              Ie = sqrt(sum_cur/(float)DATA_NUM);      //実効値算出
59
60              cout << "Ie: " << Ie <<"[A]"<<endl;
61              cout << "-------------------------" << endl;
62
63              sum = 0.0;
64              cur = 0.0;
65              sum_cur = 0.0;
66              Ie = 0.0;
67              cnt=0;
68              ch0_flag=1;
69              calc_flag=0;
70
71              sleep(1);
72              ualarm(1,100);      //void alarmWakeup関数を開始
73          }
74      }
75      return 0;
76  }
77
```

④ualarm()関数によって1μs後に割り込み関数alarmWakeup()が呼び出され、その後100μs毎に呼び出される。(A/D変換処理に移行)

⑤alarmWakeup()関数の呼び出し停止。

⑥OFFSET値の算出を行う。(設定した周期分のA/D変換値の平均値を算出)
OFFSET値算出はプログラム起動後の1回のみ。

⑦実効値の算出を行う。
電流の瞬時値を2乗し、設定した周期分加算する。
加算した値を設定した周期で割ってsqrt(平方根)の値をとる。

⑧各変数の初期化と各フラグの設定後に1s待機。

⑨alarmWakeup()関数を再度呼び出し。
(A/D変換処理に移行)

割り込み関数の詳細

【関数①】

書式： signal(シグナルの種類、扱う関数)

概要：送るシグナルの種類とそれに紐づく関数を宣言する。

使用例： signal(SIGALRM、alarmWakeup)

SIGALRMは関数②のualarm()と同時に使用するシグナル。
ualarm()で指定したμ秒の経過後にSIGALRMを発生する。また、SIGALRMが発生するとalarmWakeup関数が呼び出される。
（SIGALRMは標準シグナルNo.で決められており”14”である。）

【関数②】

書式： ualarm(usecond_t usecs、usecond_t interval)

概要：指定したμ秒後と指定した間隔でシグナルを送る。

使用例： ualarm(1、100)

ularam関数の呼び出された後、1μ秒後にシグナルを送り、100μ秒間隔でシグナルを送り続ける。
※ualarm(0、0)とするとシグナル送信を停止する。

プログラムの解説③

```
83  void alarmWakeup(int sig_num)
84  {
85      if(ch0_flag == 1)        //ch0のAD変換処理
86      {
87          if(cnt<DATA_NUM)
88          {
89              if(sig_num == SIGALRM)
90              {
91                  data_s[0] = 0b00000110;
92                  data_s[1] = 0b00000000;
93                  //data_s[2] = 0b00000000;
94
95                  a2d.spiWriteRead(data_s, sizeof(data_s) );
96
97                  a2dVal0[cnt] = 0;
98                  a2dVal0[cnt] = (data_s[1]<< 8) & 0b1111100000000;
99                  a2dVal0[cnt] |=  (data_s[2]);
100             }
101             cnt++;
102
103             if(cnt==DATA_NUM)
104             {
105                 ch0_flag = 0;
106                 calc_flag = 1;
107                 cnt=0;
108             }
109         }
110     }
111 }
112
```

⑩Raspberry Pi⇒A/D変換器への制御コマンド（D_{IN}）

data_s[0]　　data_s[1]

0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0

開始 SI D2 D1 D0

⑪spiWriteRead関数を呼び出してA/D変換器に制御コマンドを送信＆A/D変換結果を取得

⑫a2dVal0[ ]はint型なので16ビット

0	0	0	0	0	0	0	0	×	×	×	0	B11	B10	B9	B8

8ビット左シフト（<<8）

×	×	×	0	B11	B10	B9	B8	0	0	0	0	0	0	0	0

論理積（＆）12ビット分

0	0	0	0	B11	B10	B9	B8	0	0	0	0	0	0	0	0

⑬data_s[2]と③の結果について論理和（｜）をとる

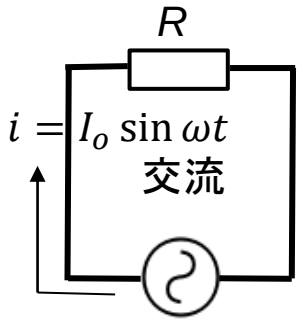
0	0	0	0	B11	B10	B9	B8	B7	B6	B5	B4	B3	B2	B1	B0

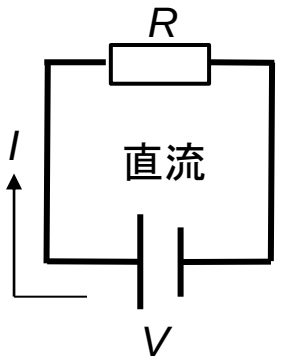
A/D変換値（12ビット）

参考資料

実効値とは？

実効値･･･交流電圧または電流の表現方法の一種。





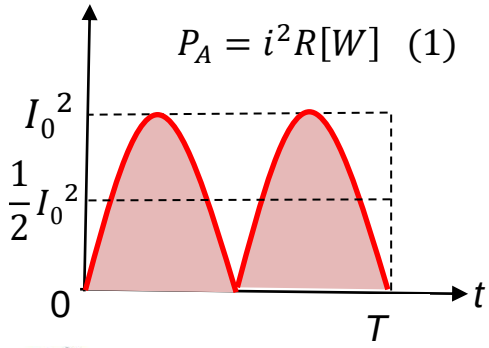
電球で考えると･･･
DCとACが同じ明るさで光るときのACの大きさをDCの大きさで表すイメージ。

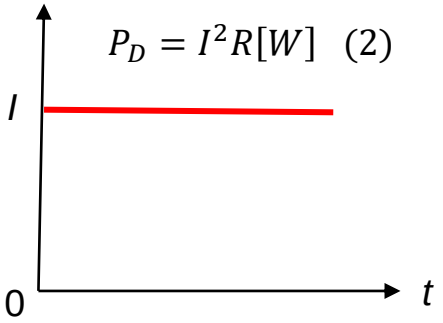
「ある電気抵抗に交流電圧を加えた場合の一周期における平均電力」

＝

「同じ抵抗に直流電圧を加えた場合の電力」

⇒この交流の電圧と電流の実効値はそれぞれ、その直流の電圧と電流と同じ値であると定義される。





$P_A = (i^2\text{の1周期の平均値}) \times R$ (3)

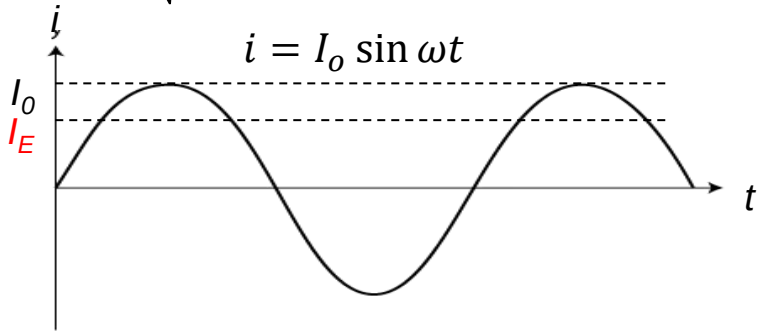
実効値はP_D=P_Aなので式(1)、式(3)より

$I^2R = (i^2\text{の1周期の平均値}) \times R$

$I^2 = (i^2\text{の1周期の平均値})$

$I_E = \sqrt{(i^2\text{の1周期の平均値})}$

$I_E = \sqrt{\frac{1}{2}{I_0}^2} \quad \therefore I_E = \frac{I_0}{\sqrt{2}}$



茨城県産業技術イノベーションセンター
Ibaraki Prefecture

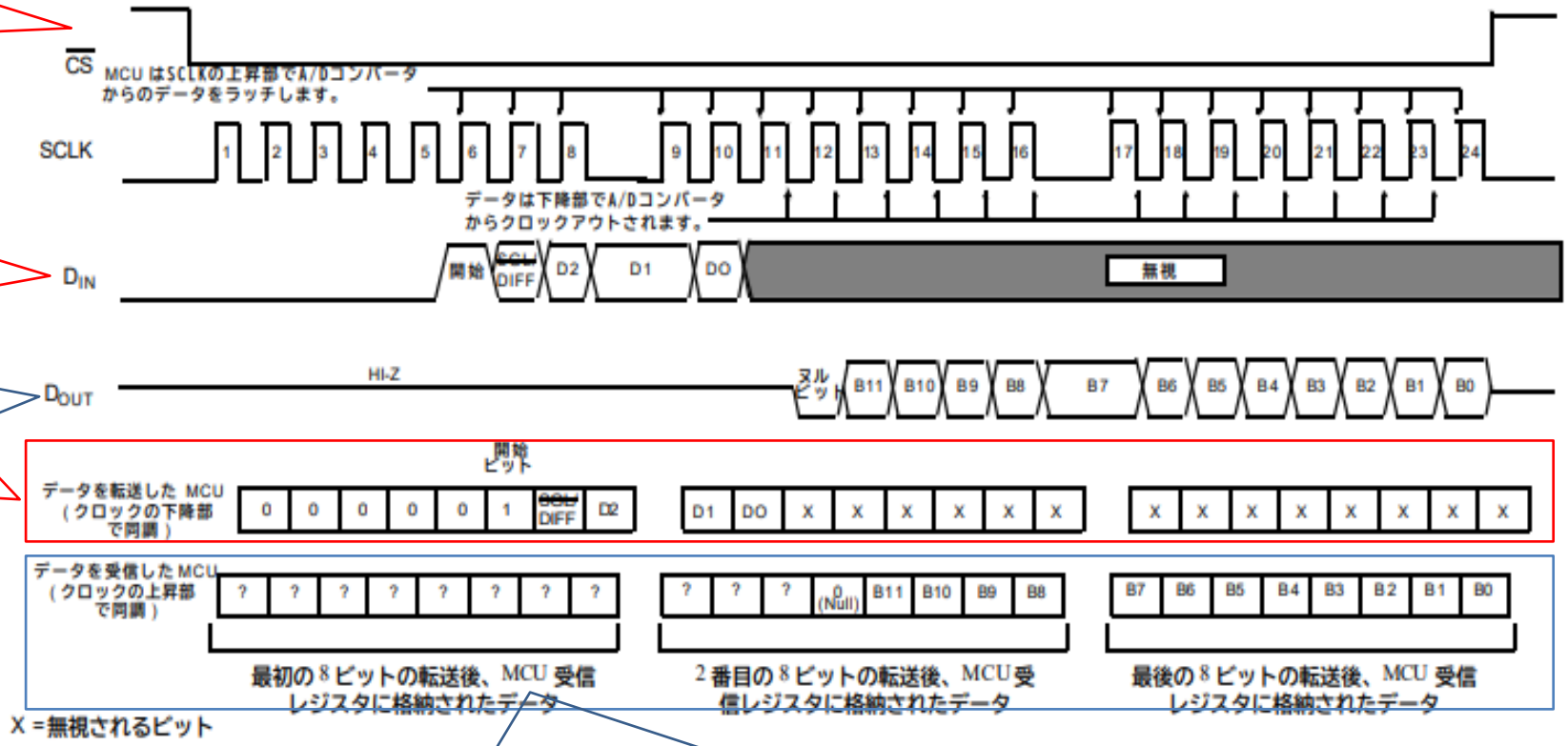
mcp3208の通信プロトコル

CSピンがHigh→Low
に通信開始

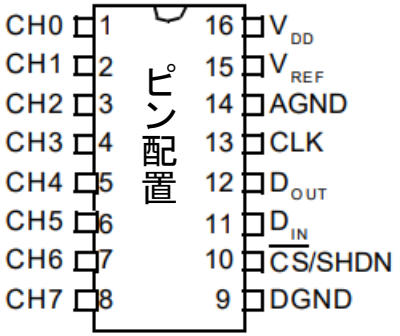
Raspberry Pi ⇒
A/D変換器

A/D変換器 ⇒
Raspberry Pi

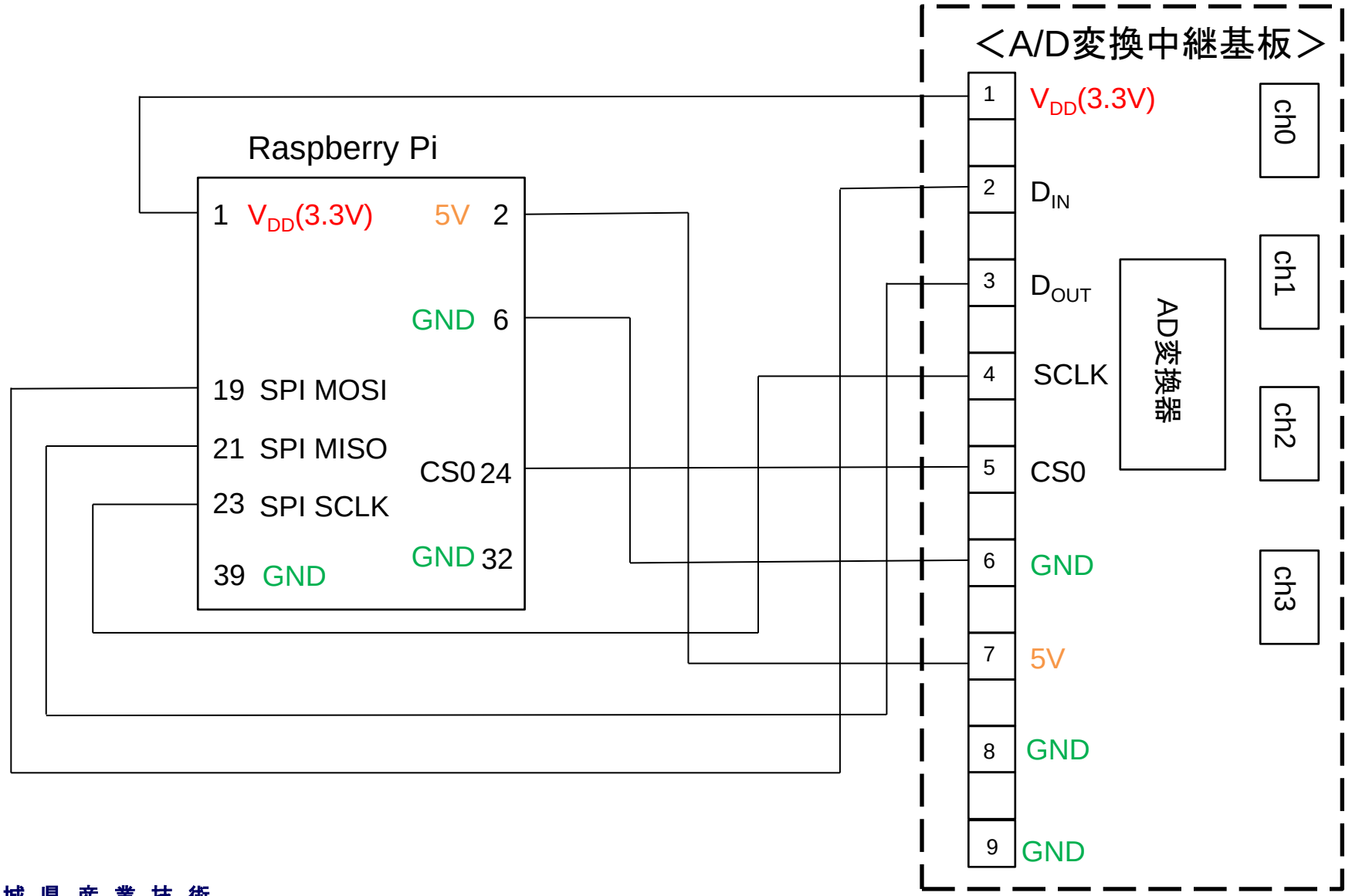
どのCHと通信する
か選択する



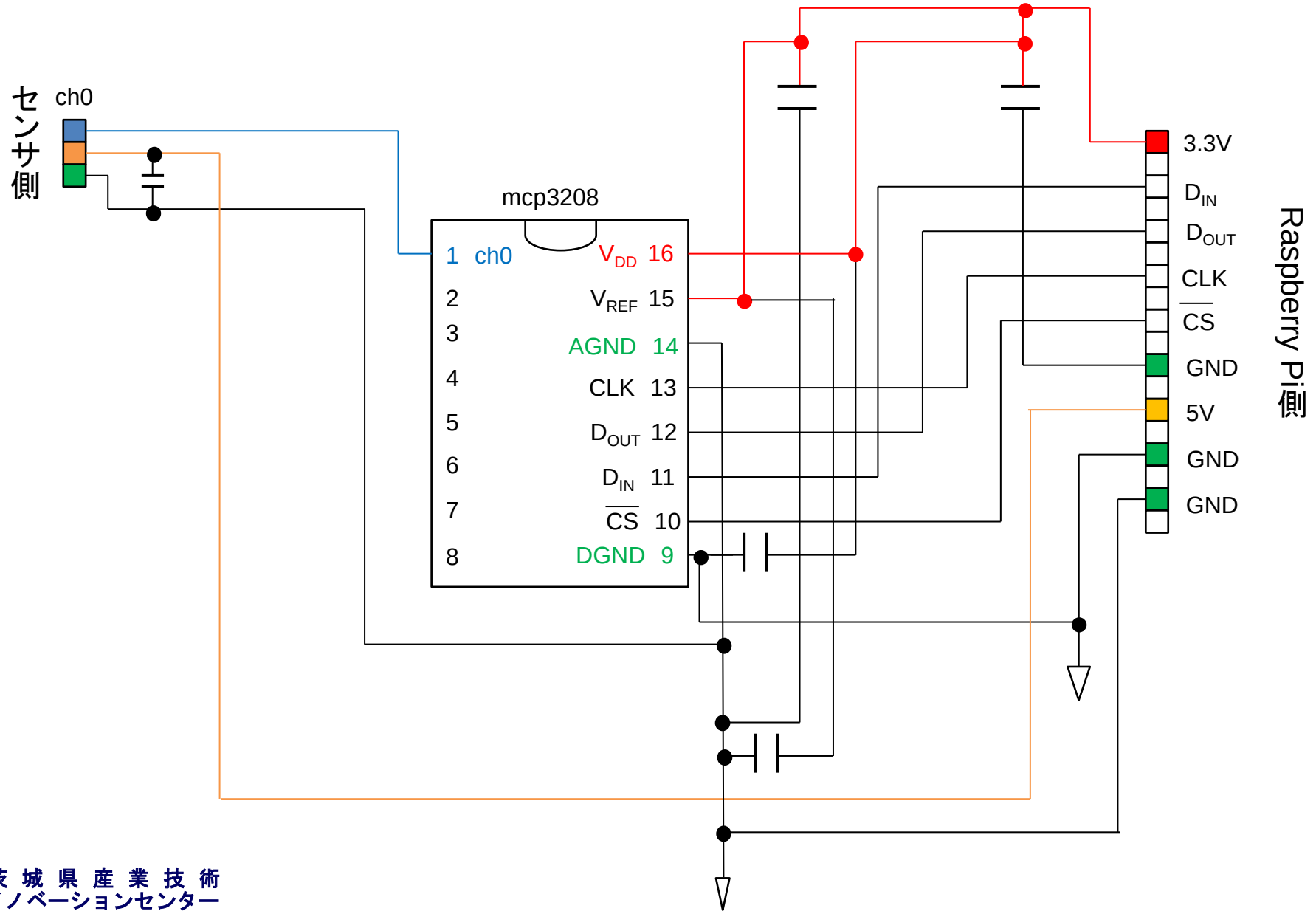
A/D変換値（12ビット分）を返してくれる。
ただし、最初の8ビット+2番目の8ビットの先頭3ビットは不明な値が入っているため
ビット演算してB11～B0のみを取り出す。



回路図1（Raspberry Pi – A/D変換中継基板）



回路図2（A/D変換中継基板）

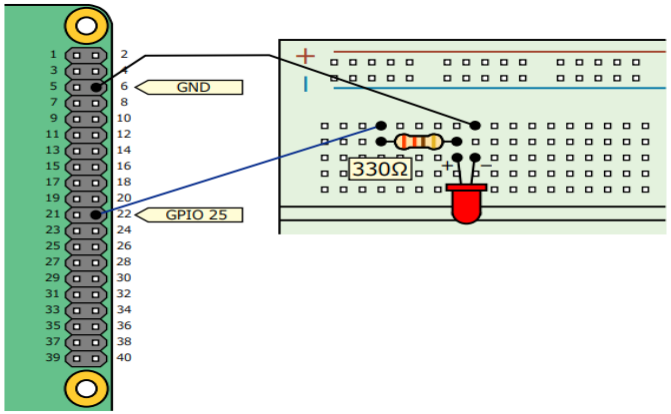


茨城県産業技術イノベーションセンター
Ibaraki Prefecture

Wiring Piの使用

○Wiring Pi

Raspberry PiでGPIOを制御するのに必要なC言語用のライブラリ。



- WiringPiの初期化
  wiringPiSetup関数で初期化する。
- GPIOピンのモード設定
  ピンの入力 or 出力の指定をする。
- GPIOの出力制御
  digitalWrite(GPIOピン名番号、値)
  値は0 or 1を指定する。
- GPIOの入力制御
  digitalRead(GPIOピン名番号)
  ピンの現在の値が返り値となる。

```
1   #include <wiringPi.h>
2
3   #define GPIO25 25
4
5   int main(void) {
6       int i;
7
8       if(wiringPiSetupGpio() == -1)
9           return -1;
10
11      pinMode(GPIO25, OUTPUT);
12
13      for(i=0; i<10; i++){
14          digitalWrite(GPIO25, 0);
15          delay(950);
16          digitalWrite(GPIO25, 1);
17          delay(50);
18      }
19
20      digitalWrite(GPIO25, 0);
21
22      return 0;
23  }
24
```

○実行ファイルを作成するコマンド

```
~$ g++ _ test.cpp _ -lwiringPi
```

※「-lwiringPi」で、ライブラリをリンクする必要がある。