

電気通信大学

工学研究部 部報

uec koken - <http://www.koken.club.uec.ac.jp/>

平成26年度第1回オープンキャンパス特別号

電気通信大学工学研究部 部報

平成 26 年度第 1 回オープンキャンパス特別号

目次

モータドライバ設計白書	2
石関	
C#のおはなし	4
池田	
Fuzz Face クローンの製造計画	7
まひる	
マイコンレジスタの謎を紐解く	9
安井	
応用技術者試験受験記	13
木村	

モータドライバ設計白書

知能機械工学科 四年 石関峻一

1. はじめに

この内容は工研というより、NHK ロボコンのものであるが、制作の精神が工研的であるので部報にすることにした。

2. 目的

今日、NHK ロボコン界隈ではモータの大出力化に伴い、モータドライバにも大電流化の波が押し寄せている。また制御の難易度の高度化により応答性も求められている。しかし現状市販品のモータドライバは価格が高く、予算のないロボコニストには手が出せない。そこで今回は安価かつ大電流・高速応答のモータドライバの制作を目指す。

3. 原理

原理のキーワードをいかに示す。内容は省略する。（関東ロボコン応援団で検索！）

FET・静電容量・立ち上がり・能動領域・飽和領域・H ブリッジ・ゲートドライバ・チャージポンプ・フレイホイールダイオード・電源分離・デットタイム・などなど

4. 設計手順

(ア) FET の選定

- ① 今回のマシンで使用する最も多く電流を流し・電圧をかけるものを調べる
- ② それに耐えうる FET を選択する（電圧・電流・静電容量・値段）
- ③ FET の寄生容量などから使用するハーフブリッジドライバの選定を行う。
- ④ 回路図の制作。この際どのような信号・ノイズが流れるのかをしっかり考える（インピーダンスは高いか低いか、電圧レベルはどのくらいか、この電源ラインにどのようなノイズが乗るのかまたそれをどうなくすかなど）
- ⑤ パターン図の制作。この際流れる電流は何 A か、大電流ラインはまっすぐか？ノイズが回りこんでくるパターンではないか・放熱板はしっかりと付いているか・耐圧は大丈夫かなど。
- ⑥ 制作（チップ部品から順番に）
- ⑦ ハーフブリッジドライバなどの仕様を加味しつつデバックプログラムの制作（デットタイムや duty の限界・PWM 周波数の限界・ショートブレーキ）
- ⑧ ライブリ化して仕様書とともにメインプログラマに渡す。

5. 設計結果

モータドライバ仕様をいかに記す。

最大電圧 - 15.0V

モータコネクター ギボシ端子

最大電流 - 150A

信号コネクター BOX ヘッダ 6P

信号 - PWM × 2 (duty 90%まで)

FET - IRLB3034PBF

電源コネクター T コネクタ

ゲートドライバ IC - IRS2003PBF

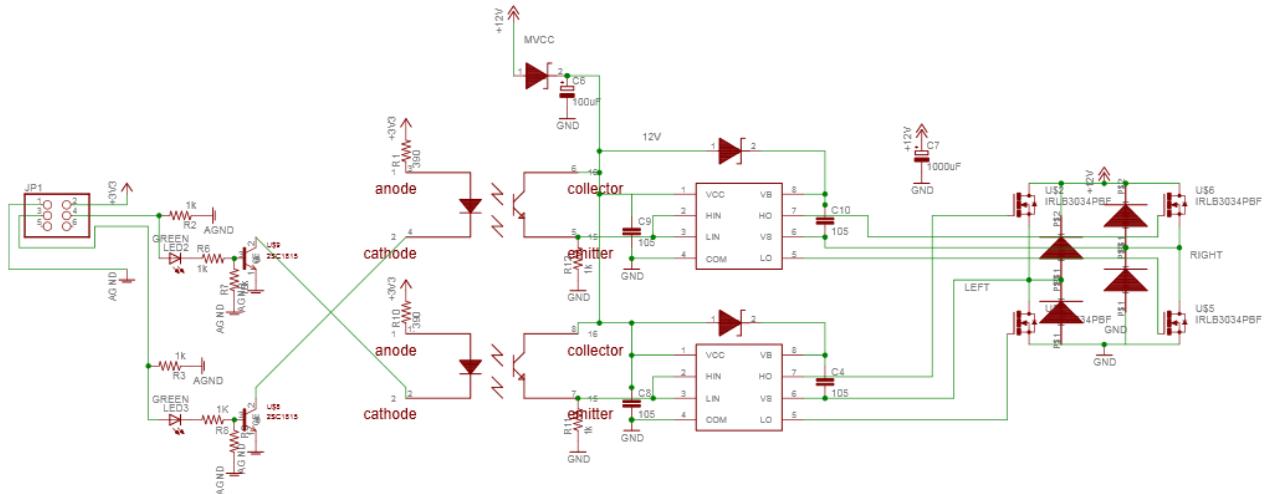


図 1.回路図

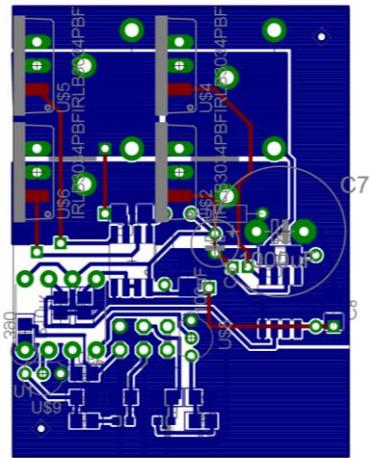


図 2.パターン図

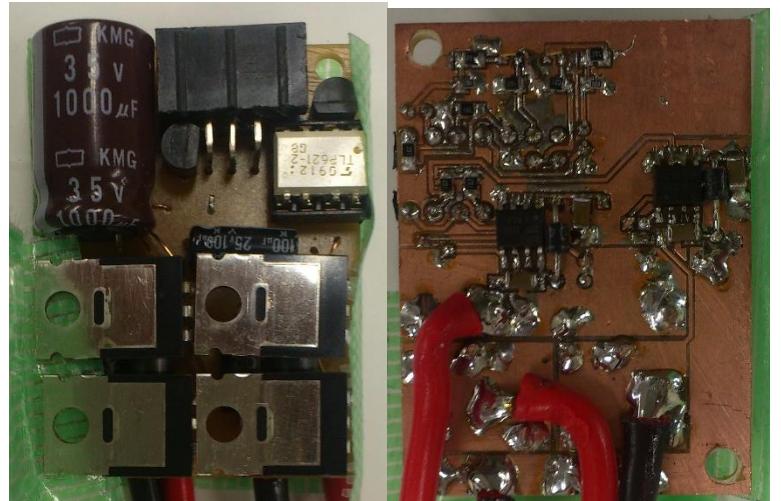


図 3 外観図

6. 考察

- (ア) ゲートドライバ IC がモータ始動時の電圧降下により正常に動作しなくなり、FET を貫通で破壊することが多々あった。そこで、ゲートドライバ IC の電源にダイオードと電解コンデンサを足した。そのことにより、電圧の降下が IC まで影響をしなくなった。
- (イ) FET のドレインのパターンを取るためにゲートは表でジャンパ処理を行った。これにより、モータ始動時の大電流でパターンが飛ぶということがなくなった。
- (ウ) C 基板ハーフの採用、今回の NHK ロボコンでは子機のサイズ制限が厳しかった。そのため回路にも小型化が求められた。そこで出来る限り詰めた設計や L 型コネクタの採用を行った。さらに配線材にはシリコン皮膜コードにし、コネクタを Φ3.5 バナナコネクタにすることで理想となると考えられる。
- (エ) モータドライバの換装。今回のモータドライバは一ドライバー基板である。また固定方法も楽なため、モータドライバに問題があった時はすぐに新品と交換を行えるような仕様となっている。

7. まとめ

今回のような超小型モータドライバの需要はごく一部である。しかし今回このような基板作成を行えたことは、回路ちっちゃいもののクラブの会員としてとても幸運であった。実質的にラジコンで使われる ESC と同等なサイズになった。今回は秋月に絞って部品調達を行った。今後更なる改良では RS などで部品を購入する必要性があると感じている。さらに信号線をロジック IC など用いて減らすことでマイコンの負担が減ると考えられる。

C#のおはなし

電気通信大学 工学研究部部報 4月号

知能機械工学科 2年 池田佳那

お父さん、C#の記事を書いて食っていこうと思うんだ。

というわけでネタが特に無いので C#のさわりの部分を少し書いてみようと思う。食っていけねえよ。

今回 C#を説明する上で、コードを記述する環境を Microsoft Visual C# 2010 Express とする。

※この記事は C 言語などに少し触れている人向けに書いている記事である。完全に初心者向けではない。あと、時間が無いので色々省いて書いた。

1. C と C#について

まず、C と C#のソースコードを比較してみる。Hello!という文字列を画面に出力するコードを記述する。

C	C#
#include<stdio.h> int main() { printf("Hello!¥n"); return 0; }	using System; class Sample { static void Main() { Console.WriteLine("Hello!"); } }

共通している部分：文の終わりに；（セミコロン）をつける、main 関数がある、文字列は””で括る

似ている部分：#include<stdio.h>でライブラリの読み込み・using System で参照の追加、printf・Console.WriteLine

基本構造としてはなんとなく似ているところもあるが、根本的に違う。
C#はCと違い、コードの文を.（ドット）で区切っていく。
C#にはクラスというものがあり、それを用いていくが、その説明をするとオブジェクト指向の話も含まれてきてしまうので今回は割愛する。

2. Windows フォームを使用する

C#で開発する手段の一つとして、Windows フォームを使用する方法がある。所謂ウインドウのことである。1番で記述したコードでは Console.WriteLine メソッドで Hello! という文字を表示させたが、今回は Windows フォームにその文字を表示させてみる。

```
using System.Windows.Forms;
using System.Drawing;

class Sample
{
    public static void Main()
    {
        Form fm = new Form();
        fm.Text = "サンプル";

        Label tx = new Label();
        tx.Text = "Hello!";
        tx.Parent = fm;

        Application.Run(fm);
    }
}
```

実行結果



いきなり色々増えたな。まず、参照の追加で、System と System.Drawing と System.Windows.Forms を追加する。今回はそれらの標準ライブラリを使用する為である。
次に、プログラムの文について 1 行ずつ軽く説明する。

```
* Form fm = new Form(); … Windows フォームクラスを新しく宣言する。
* fm.Text = "サンプル"; … フォームのタイトルを"サンプル"にする。
* Label tx = new Label(); … ラベルコントロールを宣言する。
* tx.Text = "Hello!"; … 宣言したラベル (tx) に文字列"Hello!"を入れる。
* tx.Parent = fm; … ラベル (tx) を Windows フォーム (fm) に貼り付ける。
* Application.Run(fm); … フォーム (fm) のアプリケーションを実行する。
```

tx.Parent = fm; が少し捉えづらいかも知れないが、文字が書かれたシールをウィンドウに貼り付けるというイメージである。

今回はここまでにしておく。少しでも C#について興味を持ったら、是非自分で調べてみる事をお勧めする。

Fuzz Face クローンの製作計画

情報・通信工学科 1年 まひる

1. はじめまして

こんにちはこんばんわ、新入生のまひるです。
せっかく工研に入ったんだからとりあえず何か作ってみようと思ったので、

- ・エレキギターの黎明期から存在する。
- ・簡単な回路の割に動作原理がよく解らない。
- ・何か変に歪んで変な音がする。
- ・あと個体差が激しいし安定しない。

そんな愛されエフェクターの Fuzz Face を作ってみようと思います。

今回は LTspice を使っての回路シミュレートを行います。

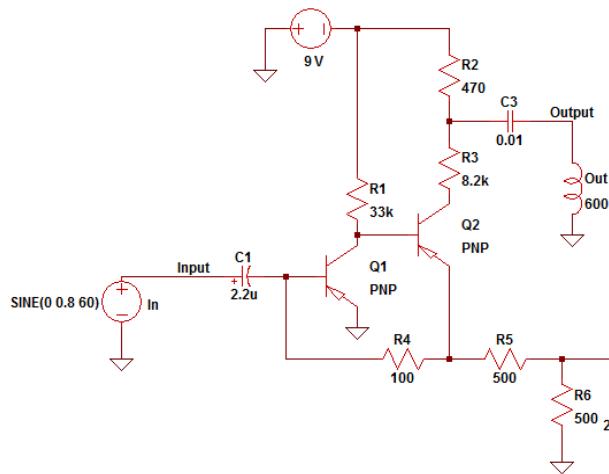
2. そもそもエフェクターって何さ

という話ですが、まずエレキギターが発する電気信号は微弱です。これをギターアンプとか呼ばれる真空管とか使った増幅器で信号を増幅し、スピーカーに繋いでないのでっかい音が鳴ります。

で、物好きな人達が、カッショイ音を出したいなどの目的で、ギターとアンプの間に音が変わる電気回路を挟むようになりました。これがエフェクターとか呼ばれるモノです。多分。きっと。

3. 実際どんな回路なのか

そんなこんなで今回製作計画を立てた Fuzz Face についての話に移ります。



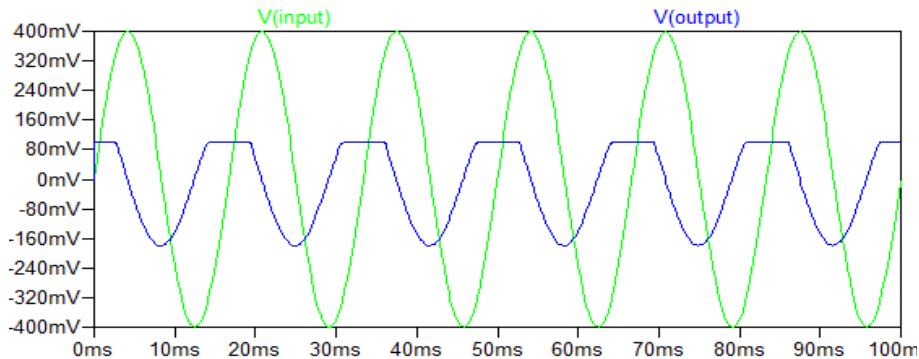
まず回路図ですが大体こんな感じです。ネットで探してきた回路図を元に、LTspice という電子回路シミュレータで作ってみました。

Fuzz Face は 1960 年代に作られたものなので、時代的に当時普及していたのか PNP 型のゲルマニウムトランジスタが使われています。おかげで GND が高電位になってて氣色悪いです、グランドから電流が流れてくる…。電子の動き的にはむしろ自然に見える気がしないでもないです。

本来 R5、R6 のは $1\text{K}\Omega$ の可変抵抗器で、ここでエフェクトの掛かり具合を調節するのですが、LTspice に 3 端子の可変抵抗がなかったので、適当に等価回路のようなものを作りました。シミュレーターを走らせる時にいちいち手動で定数変えるので面倒臭いです。

ついでに本来は Output の位置にも可変抵抗が付きます、こっちは出力レベルの調整になります。Output からそのまま GND に繋ぐと多分よろしく無い事になるだろうと予想されたので、適当にインダクタを挟んであります。

4. シミュレートしてみました



LTspiceでシミュレーターを走らせてみた結果、こんな感じになりました。Inputからの正弦波の入力が緑色。それに対するOutputでの出力電圧が青色です。

なんということでしょう。あんなにも綺麗な正弦波だった入力が、匠の技によって見事に波形が歪んでいます。出力波形を見た感じ、電位が正の側の山をぶった切った感じになっていますね。ダイオードの半波整流回路とか言う奴に似てます、Google先生に聞いて調べました。

この非対称な歪みがFuzz Faceの独特の音に繋がってるようです。 実際どんな音がするかなんですが、適当にYoutubeでFuzz Faceって入れて検索すると一杯出てきます。見た目が可愛いです。

5. 今後について

LTspiceで回路を組んでシミュレートしてみて、実測の波形に近いものが出てきましたが、なんでこうなるのかの原理はよく解っていません。

というわけで制作を進めると同時に、原理について調べて解ったらまた書いてみようかと思います。参考資料[1]に多分全部書いてあるんですが全部英語なので辛いですね。

6. 参考資料

[1]The Technology of the Fuzz Face

http://www.geofex.com/article_folders/fuzzface/fftech.htm

[2]ファズフェイス特集！ Part. 1 ~歴史と定番モデル – きになるおもちゃ

http://d.hatena.ne.jp/toy_love/20121126/1353941196

[3]Dr ZEE WORKSHOP Vintage FUZZ BOX (Fuzz Face) Lab Test Demonstration

<https://www.youtube.com/watch?v=zqDuADvNApc>

マイコンレジスタの謎を紐解く

情報・通信工学専攻 学籍番号:1331107
安井 駿斗

1 レジスタ

```
-----  
#include <avr/io.h>  
#include <util/delay.h>  
  
int main(){  
    DDRD = 0x01; // 出力設定  
  
    while(1){  
        PORTD ^= 0x01; // 出力反転  
        _delay_ms(1000); // 1秒待つ  
    }  
    return 0;  
}  
-----
```

こんなコード、マイコンを使う人は誰でも理解できると思います。誰でも最初に書くLEDチカチカのコードですが、ここで使用されているレジスタ、「DDRD」、「PORTD」とは、C言語上でどのように表現されているのでしょうか。代入したり値を読んだりできるのですから、変数であることには間違いないでしょうが、これをもう少し詳しく追ってみようと思います。

2 定義を追う

とりあえず、AVRマイコンATMega328Pのレジスタ定義ファイル「iom328p.h」から、レジスタの定義部分のマクロを引っ張ってきました。PORTBを例にすると、以下のように3段にマクロが組まれていました（正確には「_SFR_OFFSET」を含めて4段）。

```
-----  
#define PORTB _SFR_I08(0x05) // PORTB の定義  
#define _SFR_I08(io_addr) _MMIO_BYT((io_addr) + __SFR_OFFSET) // _SFR_I08 の定義  
#define _MMIO_BYT(mem_addr) (*(volatile uint8_t *) (mem_addr)) // _MMIO_BYT の定義  
-----
```

全てマクロですから、これらはコンパイル時にコンパイラが展開を行います。ではコンパイラがどのように展開しているのか、これらのマクロを組み合わせてPORTBレジスタの実態を見てみます。

```
-----  
PORTB <=> _SFR_I08(0x05)  
      <=> _MMIO_BYT(0x05 + __SFR_OFFSET)  
      <=> (*(volatile uint8_t *) (0x05 + __SFR_OFFSET))  
-----
```

と、このように展開されていることが分かります。

内側の () から順に説明します。($0x05 + _SFR_OFFSET$) は、マイコン内のアドレスを示している箇所です。PORTB レジスタの定義で書かれていた $0x05$ は、最終的にこの位置に挿入されます。 $_SFR_OFFSET$ はマイコンに応じたオフセット整数値であると思われるため、とにかくにもこの () は整数値が入っていることがわかります。

その手前にある (`volatile uint8_t *`) は、C 言語文法におけるキャストです。すなわちこれは、後に続く数値を「8 ビット長正数値の変数のアドレスだよ」とコンパイラに提示しているわけです。volatile ですから最適化防止変数として扱っていることも分かりますが、ここでの本題は別ですからあんまり気にしないで下さい。

よって、(`volatile uint8_t *`)($0x05 + _SFR_OFFSET$) はアドレス「 $0x05 + _SFR_OFFSET$ 」を持つ 8 ビット変数のアドレスを示しているわけです。となれば、アドレス値の前に '*' が 1 つついているのは、後ろに続くアドレスを参照した変数を生成しているのだと分かります。

すなわち PORTB の実態は、「ある値を 8 ビット変数のアドレスと認識させ、そのアドレスを参照した変数」ということになります。わかつてしまえば大したことのない正体でしたね。

3 応用してみる

正体がわかつただけではつまらないので、もうちょっと応用してみます。PORTB などのレジスタは、どれもアドレスを持った変数です。であれば、逆にアドレスを読み取ることもできるわけであり、「&PORTB」などの書き方が許されるはずです。

先の定義から、「&PORTB = ($0x05 + _SFR_OFFSET$)」というアドレス整数値であることが分かります。見ればこれは、_MMIO_BYT マクロに投げられてる () 内とまったく同じ！すなわち、次の左右は等価なものになります。

```
-----  
PORTB <=> _MMIO_BYT(&PORTB)  
-----
```

この記法を使って先の L チカプログラムを書き直せば、次のようにになります。

```
-----  
#include <avr/io.h>  
#include <util/delay.h>  
  
int main(){  
    _MMIO_BYT(&DDRD) = 0x01;  
  
    while(1){  
        _MMIO_BYT(&PORTD) ^= 0x01;  
        _delay_ms(1000);  
    }  
    return 0;  
}  
-----
```

正直なんの意味があるのか分からぬプログラムになってしましましたが、重要なのは「&DDRD」などは整数値である、ということです。これだけで、応用の幅はとても広がります。

4 実践応用 1 : PORT レジスタから DDR,PIN レジスタへ変換

_MMIO_BYTE マクロは、要は渡されたアドレス値を持つ 8 ビット変数へ変換するマクロです。これを使えば、適当な値を代入してレジスタを生成できる、ということになります。

ところで、マイコンで PORTD の 0 番ピンを使うとしたとき、DDRD レジスタ 0 番の設定を出力にし、PORTD に書き込んで出力をすると、という手順を踏みます。しかし先の例のように、同じような文を 2 つ書かなければいけません。これはなんとなく冗長ですね。

ATMega328P のデータシートを見ると、IO ポート関係のレジスタ (PORTx, DDRx, PINx) の 3 つは、アドレスが連続しているようです（順に減る）。他の AVR マイコンについては確認ていませんが、おそらく似たようなものでしょう。たとえば、ATMega328P の PORTD = 0x0B, DDRD = 0x0A, PIND = 0x09 となっています。

このようにアドレスが連続しているのなら、PORTD のアドレスを読み取り、1 つずらした値を _MMIO_BYTE に突っ込めば、DDRD レジスタや PIND レジスタが生成できる、という寸法です。これを使って先の例を書き換えてみましょう。

```
-----  
#include <avr/io.h>  
#include <util/delay.h>  
  
int main(){  
    _MMIO_BYTE(&PORTD-1) = 0x01;  
  
    while(1){  
        _MMIO_BYTE(&PORTD) ^= 0x01;  
        _delay_ms(1000);  
    }  
    return 0;  
}
```

可読性がひどく悪くなつたので、次例のようにマクロを組んで書き直してみます。

```
-----  
#include <avr/io.h>  
#include <util/delay.h>  
  
#define MYPORT PORTD // 出力させたいポート  
#define MYPIN 0 // 出力させたいピン番号  
  
#define PORT_to_DDR(adr) _MMIO_BYTE((adr)-1) // PORT アドレスから DDR レジスタへ  
#define PORT_to_PIN(adr) _MMIO_BYTE((adr)-2) // PORT アドレスから PIN レジスタへ  
  
int main(){  
    PORT_to_DDR(&MYPORT) = (1<<(MYPIN)); // DDR の設定  
  
    while(1){  
        MYPORT ^= (1<<(MYPIN)); // 出力  
        _delay_ms(1000);  
    }  
    return 0;  
}
```

これで、出力するポートやピン番号を変えたいとき、上部の MYPORT の辺りを書き直すだけで対応できるようになりました。

5 実践応用2：配列に登録されたIOポートの同期出力

マイコンを扱っていて、ある多数のピンを同時にONにしたりOFFにしたりする必要があるとします。不幸なことにその多数のピンはポートが異なるため、普通はポート別にDDRやPORTの設定を書く必要があります。さらに残念なことに、上司カッコカリから、汎用性をもたせるためにマイコンや環境が変わっても少しの訂正で済むようにを言い付けられてしまいました。

しがない部下は「たとえばプログラムの上方にPORTBの0番、PORTCの3番、など書いておいて、その部分を書き換えるだけで直せればなぁ」と思うわけです。でもPORTBなどのレジスタは変数なので、別の変数で置き換えたり配列に登録できたりはしません。

しかし、「&PORTB」などはただの整数値なので、これが可能です。この部下は次のようにコードを組み、先のレジスタ変換も併用して見事に汎用性をもたせたコードを仕上げました。

```
-----  
#include <avr/io.h>  
#include <util/delay.h>  
  
#define PORT_to_DDR(adr) _MMIO_BYTE((adr)-1) // PORT アドレスから DDR レジスタへ  
#define PORT_to_PIN(adr) _MMIO_BYTE((adr)-2) // PORT アドレスから PIN レジスタへ  
  
// ここにレジスタアドレスとピン番号を交互に登録  
uint8_t RegArray[] = {&PORTB, 0, &PORTC, 3, &PORTD, 2};  
  
int main(){  
    uint8_t i;  
  
    // DDR レジスタ設定  
    for(i=0; i<sizeof(RegArray); i+=2)  
        PORT_to_DDR(RegArray[i]) |= (1 << RegArray[i+1]);  
  
    while(1){  
        // 出力  
        for(i=0; i<sizeof(RegArray); i+=2){  
            _MMIO_BYTE(RegArray[i]) ^= (1 << RegArray[i+1]);  
            _delay_ms(1000);  
        }  
    }  
    return 0;  
}  
-----
```

これで上司もゴキゲンな、修正容易なコードになりました（なお、アドレス値を整数配列に突っ込んでいるため、コンパイラさんから警告が出ますが、あくまで警告なので実行には問題ないです。気になる人は、「(uint8_t)(&PORTB)」などとして8ビット変数にキャストすれば消えると思います）。

6 おわりに

最近ふと思いついた事柄だったのですが、意外に使えそうなのでこうして部報にしてみました。自作ライブラリを作りたい部員諸君にはぜひ応用してもらいたいです。

応用情報技術者試験受験記

情報・通信工学科 3年 木村敢

今年の 4/20(日) に受験し、合格した応用技術者試験について書きたいと思います。

1 応用情報技術者とは

応用情報技術者とは情報処理技術者試験の中の一つの試験であり、独立行政法人情報処理推進機構(略称 IPA)が実施している国家資格試験です。

1.1 資格所持によるメリット

資格所持者は IPA より

高度 IT 人材となるために必要な応用的知識・技能をもち、高度 IT 人材としての方向性を確立した者

と認められます。また、この資格の有資格者は企業が採用の際に考慮する項目でもあります。さらに、他の国家資格試験(中小企業診断士、弁理士)の一部試験免除が得られます。教員採用先行試験の一部免除が受けられる県もあるそうです。また、一部県警においてコンピュータ犯罪捜査官やサイバー犯罪捜査官の応募資格もあります。

まあ、持っておいて不便はなく、就職に役に立つ資格です。

1.2 種類

情報処理技術者試験には以下のような資格があります。括弧の中は略称です。

- IT パスポート試験 (IP)
- 基本情報技術者 (FE)
- 応用情報技術者 (AP)
- IT ストラテジスト (ST)
- システムアーキテクト (SA)
- プロジェクトマネージャ (PM)
- ネットワークスペシャリスト (NW)

- データベーススペシャリスト (DB)
- エンベデッドシステムスペシャリスト (ES)
- 情報セキュリティスペシャリスト (SC)
- IT サービスマネージャ(SM)
- システム監査技術者 (AU)

FE は基本的な知識、技能を AP は応用的な知識、技能が問われます。IP は少し毛色が違い、IT を利活用する社会人に求められる基礎知識が問われます。また、これら以外の ST などの試験はスペシャリストと呼ばれ、高度な知識、技能が問われます。難易度は早い順に IP, FE, AP, スペシャリストとなります。詳しいことは IPA のホームページに書いてあるので興味がある人は見てみるといいでしょう。

1.3 試験日程

IP は随時実施されていますが、他の試験は違い毎年 4 月と 10 月の第三日曜日に行われます。基本情報技術者、応用情報技術者は両方とも試験が行われますが、スペシャリストの中には 4 月のみ若しくは 10 月のみしか行われないものもあります。

2 内容

応用情報技術者は以下の知識が問われます。

- テクノロジ系
基礎理論や、コンピュータシステム、技術要素、開発技術について問われます
- マネジメント系
プロジェクトマネジメント、サービスマネジメントについて問われます
- ストラテジ系
システム戦略、経営戦略、企業と法務について問われます。

試験は午前と午後に分かれていて、午前は全問選択肢問題、午後は記述問題になっています。午前午後ともに 100 点満点です。

なお、合格点は午前午後共に 6 割の 60 点になっています。また、午前試験で 60 点に満たなかつた場合は午後は採点されません。

3 勉強方法

個人個人で合う勉強法は異なると思うので、私がした勉強法を書きます。

まず、応用情報技術者用の参考書を買います。私が購入したのは Ohm 社の”2013 年版 応用情報技術者標準教科書”です。正直あまり良くなかったので別の参考書を買うことをオススメします。ネットでオススメを探してもいいですし、実際に少し立ち読みなどをして決めるのがいいでしょう。

う。買ったら、一周は読みます、内容は全部理解しないでいいのでとりあえず読みます。読んだら一度午前試験の過去問をやります。過去問は IPA が解答と共に公開しているのでそれを使います。やり方としては、解らない問題が出たら参考書を読みながら問題を解く方法で解いていきます。多分 4 割も取れないと思うので、少し焦ります。そこからわからなかったところを中心に参考書を読み直します。あとはひたすら演習です。演習にはこの”応用情報技術者ドットコム”と言うサイトがいいと思います。このサイトに”Web アプリ過去問道場”というものがあり、これでひたすら午前試験の過去問の演習ができます。問題の解説もついているのでかなりいいです。私が午前試験を通過できたのはこのサイトのおかげと行っても過言ではありません。

これで午前試験の対策は大丈夫です。午後試験は記述問題があるので自分の実力がしっかり出ます。なので一夜漬けで合格等は難しいと思います。午後試験は午前試験に比べて難しいので、ある程度午前試験の勉強が終わってから取り組むといいでしよう。勉強方法としては午前と同じでひたすら演習です。午後用の対策問題集などが出版されているので、それらを活用してもいいでしよう。

これで合格の下準備は完了です。あとは、頑張って本番で合格点を取りましょう。

4 最後に

このやり方でやったからといって必ず合格するわけではありません。合格率も 20% とあまり高くありません。むしろ低い方です。実際私も一回落ちてます。なので、恐れずにどんどん挑戦していきましょう。

5 参考文献

情報処理推進機構ホームページ <http://www.ipa.go.jp/index.html>
応用情報技術者ドットコム <http://www.ap-siken.com/>

発行所 国立大学法人 電気通信大学工学研究部

〒182-8585 東京都調布市調布ヶ丘 1-5-1 サークル棟2階

E-mail koken@koken.club.uec.jp

URL <http://www.koken.club.uec.ac.jp/>

発行 横田嶺

編集人 吉村英幸

発行日 2014年7月20日

執筆 工学研究部 部員

