mbed 講習会の指導時の注意点

2016/04/02 梶本

mbed 課題において理解していない人が多発するポイントをまとめます.

指導側が理解していないと無理解は世代を超えて拡大していきます. 指導者は, ソースコードやハードウエアがそうであることの理由として「おまじない (こうすればなぜかうごく)」は口にしないようにしましょう. これは研究にも通じる姿勢かと思います. 指導側もしっかり勉強するチャンス.

1　ハードウエアに関して

1.1　　LED の電流

LED に流れる電流は**抵抗の両端の電圧によって実際に計測してください**。これを、「LED は 1.2V で、電源は 3.3V だから○mA 流れる」というのは間違いです（流す電流を設計する際には使いますが)。例えば 1.2V は赤色 LED では代表的な値ですが、かなりブレますし、他の色、特に青色等では 3V 程度になります。

2　ソフトウエアに関して

2.1　　PWM の「パルス周期」

プログラム中の wait(0.25)で PWM の周期を作っていると誤解している場合が多々あります。PWM モジュールはほうっておいても勝手に繰り返します。その繰り返し周期は period 関数によって事前に指定されています。
このプログラムにおける wait(0.25)は実はほとんど意味がありません。

2.2　　DA 出力時の while ループの誤解

DA のプログラムで正弦波が出来る仕組みを理解していないことが良くあります。

```
while(1){
    t+=0.001;
    DA=sin(2πft);
}
```

というようなコードがあった時、while ループの一周期ごとに t が増え、それにともなって計算される正弦波の値が変化していきます。でもこのプログラムを見て、

DA=sin(2πft)；
を「一回」呼び出すだけで「一周期分の正弦波」が出力される、という不思議な理解をする人が結構出てきます。
つまり、上の sin 関数は「一つの値を返す」もので、それが while 文で連続されて初めて「波形」になるのですが、それを理解せず、「sin 関数を書いたら sin 関数が出る」と思っている場合があります．**while ループは意外に理解できないポイントです**.

2.3 printf による遅延

波形生成ではループ周期の速さと安定性が鍵となります．本来はタイマー割り込みを使うと良いところですが講習ではそこまでやっていません.
一方で,「波形生成の一番忙しいループ内で printf 関数を使い続ける」程度のことはしょっちゅう行われています．ちょっとプログラムをかじると常識的な「printf は重い」というような話は大学の講義の中では行われていないため，一回失敗してもらって printf を削ってもらうというプロセスはどうしても必要です.

2.4 SPI 通信時の while ループの誤解

類似の（逆の？）ことは SPI 通信でもあるようです。SPI 通信では 8～16 個のクロック信号によって一点のデータを送受信しますが、

```
while(1){
    spi.write(data);
}
```

というようなコードがあった時に、while ループの一周期ごとに「一クロック」だけ送られるという理解をしている人がいるようです。実際には spi.write という関数の中で 8-16 クロックが送られます.

2.5 シリアル通信における典型的な悪いループ

mbed で「シリアル通信で PC からキーボード入力し、波形を生成する」というプログラムで、多くの人が微妙に良くないプログラムを書きます．**実はこれも while ループ問題の一種です.**

（悪い例その１）

```
while(1){
    num = pc.getc()-'0'; //シリアル通信で一文字取得、'0'を引いた数値に
```

```
    t+=0.000034;   //時間の更新
    signal = 0.5*sin(2.0*3.1415*f*mm[num-1]*t)+0.5; //DA の更新
}
```

これが何故問題かというと、pc.getc()はデータが来るまで待ち続けるので、正弦波を出し続けることができません。出し続けるためにはキーボードを押し続けないといけないです。

信じられないかもしれませんが，多くの人がキーボードを押し続けることで解決（！）したりします.

（悪い例その２）

```
while(1){
      num = pc.getc()-'0'; //シリアル通信で一文字取得、'0'を引い数値に
      for(t=0;t<1.0;t+=0.000034){   //時間の更新
          signal = 0.5*sin(2.0*3.1415*f*mm[num-1]*t)+0.5; //DA の更新
      }
}
```

ある学年ではほとんどの人がこの、「while 文のなかで、for 文によって一周期の正弦波を作る」ことで問題を擬似的に解決（！）していました。

元の問題は「シリアル通信に時間が掛かる」ないし「シリアル通信データが来ていないとプログラムがストップしてしまう」というものでしたが、for 文で少なくとも一周期連続するという保証をしているわけです。悪い例その１よりはよく考えています。

しかしこれはやはり良くないプログラムです。なぜなら今度は、ユーザの入力に対する応答性が保証されないからです。

（多分正しい例）

実際には次のように書くのが普通です

```
while (1) {
    if(pc.readable()){
       num = pc.getc() - '0';
    }
    t+=0.000034;
```

```
    signal = 0.5*sin(2.0*3.1415*f*mm[num-1]*t)+0.5;
}
```

この場合、**readable 関数によって「シリアル通信バッファにデータが来ているかどうか」を確認します**。もし来ていれば getc をしてもそこで止まることはありません。来ていなければ数値 num に変更はないので、時間のみ更新され、正弦波が出力され続けます。

以上のことは PC 側のプログラムでも頻繁に問題になります。研究室では C++プログラミングの際に Serial.h と Serial.cpp を使っている場合が多いと思いますが、その初期化コードのなかに、「データを待ち続けるかどうか」を設定する部分があります（皆書いているはずです)。また上の例のようにバッファを確認するように書くことも出来ます。

こういう通信系は、「たまたま動いたり、調子が悪くなって動かなくなったり」するプログラムの原因となりますので明確な理解をお願いします。

さらに綺麗に書くには「割り込み」を使います。つまりシリアル通信データが来るというイベントで無条件に飛ぶ割り込み関数を定義します。それでメインのプログラムで「毎周入力をチェックする」という本来無駄な処理をしなくてよくなります。