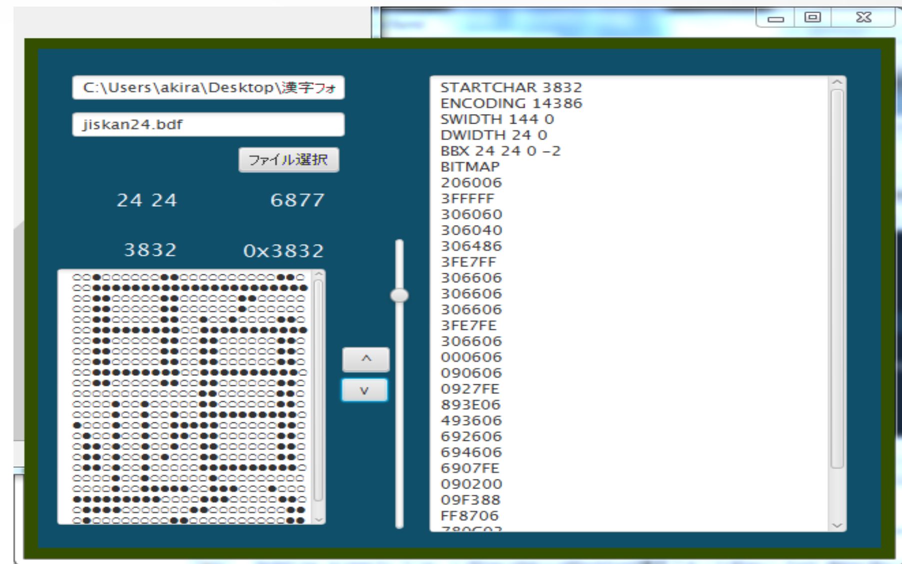
RangeCoderの開発



シャープのモノクロ液晶モジュール（2．7インチ）を使って、日本語を扱うアプリケーションの構想を練り始めた。

先ずは準備作業として、漢字フォント(jiska24)をPICに搭載する方法を考えてみた。

jiskan24

jiskan（じすかん）には、二種類のビットマップフォントがある

＝＝＞「jiskan16」及び「jiskan24」

何故24bitフォントを選ぶのか？

＝＝＞16bitフォントでは字が小さくて読めそうも無いから

最新版はjiskan24-2003-1.bdf（矢木達也氏作成）で8836文字のフォントが含まれる

大きなデータ

単純計算

8836文字×72バイト = 636kバイト以上

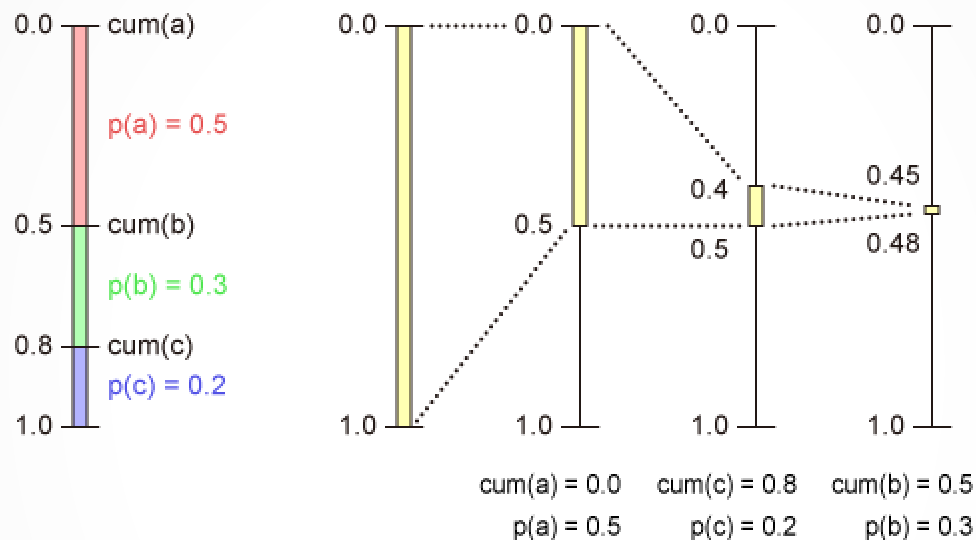
文字コード変換テーブル

8836文字×8バイト（？）=70kバイト

（他に半角文字のフォントも必要）

RangeCoderを使ってデータ圧縮することを考えてみた

RangeCoder



http://codezine.jp/article/detail/443より抜粋

算術符号を整数演算で実現したアルゴリズム

精度の面では算術符号に劣るが、8bit単位で処理するため高速

試してみた

Kanji_Font_Analizer（Javaプログラム）に組み込んでみた

Step-1

jskan24-2003のフォント（8836文字分）を読み取り、8bitパターンの出現頻度テーブルを作成する

Step-2

出現頻度テーブルを出現頻度の降順にソートして8bitパターンごとに累積頻度を求める

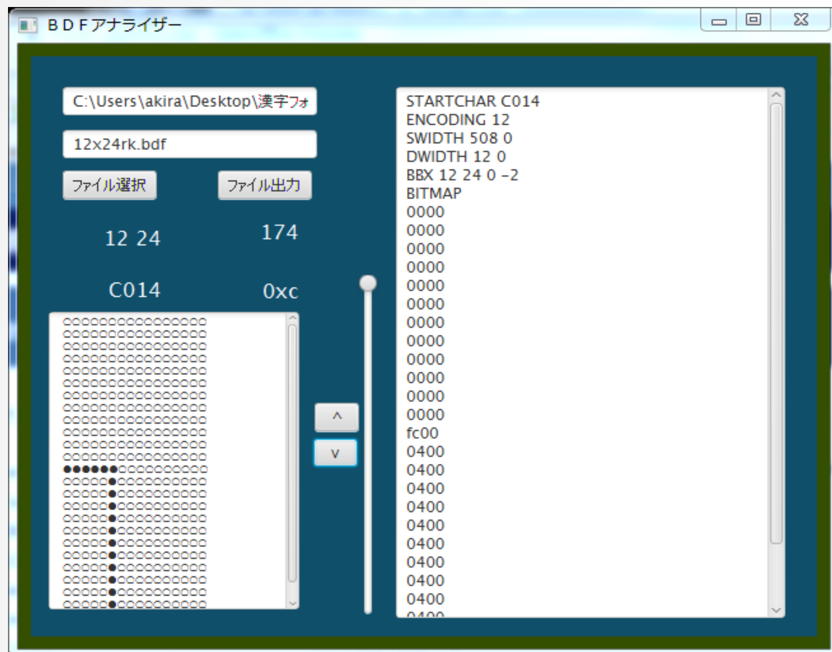
Step-3

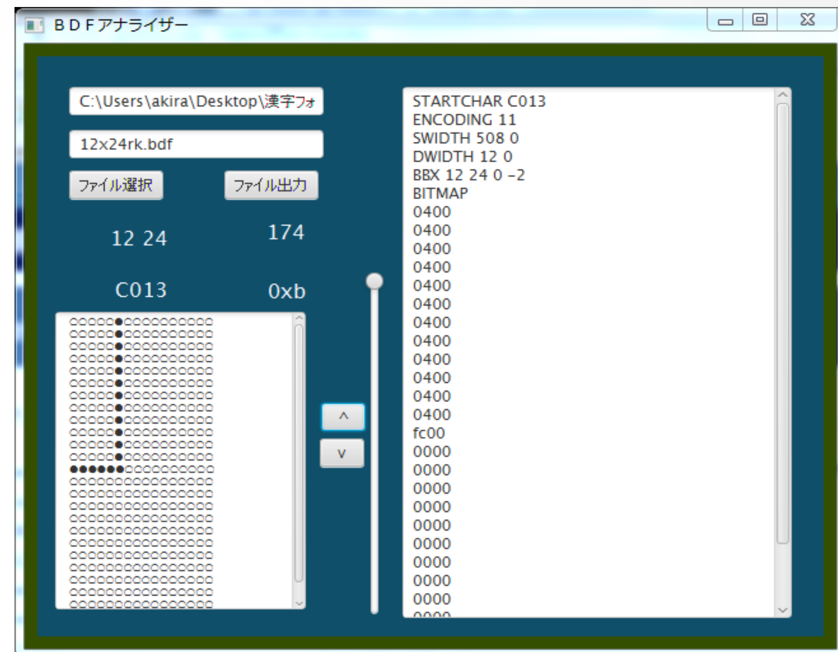
１文字ごとに符号化する

Step-4

１文字ごとに復号化して元のフォントと比較する

一致しない





一致しているものもあったが、違うパターンになるもの、余計な0が出力されるものもあった

何故こんなことになるのか？

プログラムのバグ？

RangeCoderの処理を追った

RangeCoderの処理過程をOpenCalcでシミュレートして、プログラムの動作と比較してみた

＝＝＞RangeCoderプログラムは正しく動作している

違いを生じたのは打ち切り誤差の影響だった！

＝＝＞整数除算では不可避

エンコード処理を修正

```
for( Byte b: input ) {
        code <<= 8;
        range <<= 8;
        bb = b.intValue();
        bb &= 0x0ff;
        code += bb;
        while( range >= (1L<<24) ) {
           c =  ( code+1 ) << Range_Coder.total_bit;        <==ここ
           c /= range;
           keyfh.key = c.intValue();
           index = Collections.binarySearch(fh_list, keyfh, fhrc );
```

疑問を感じる処理

```
uint r = R / total;
if (high < total){
    R = r * (high-low);
}
else {
    R -= r * low;
}
```

RangeCoderが動いた

修正の結果、全ての文字について符号化前のフォントと復号化後のフォントが一致した

ただし、復号化後のフォントに不要な0が出力される件は残っている

＝＝＞原理的なものと考えている

フォントに現われる8bitパターンは256種あるのでEOFを付け加える方法は使えない

フォントのデータサイズ（72バイト固定）まで出力したら復号処理を停止する方法で対処する

圧縮率

jiskan24の場合、符号語の圧縮率は75%
（25%削減）

RangeCoderの効果は微妙・・・

第一水準だけにすれば50%削減される

PIC32の試作

PIC32MX250F128B（1個360円@秋月）でPIC32の試作を始めた

PIC32MX1XX/2XX Family Silicon Errata and Data Sheet Clarificationは必読！