■回路規模削減 /JavaRock

JavaRock で生成される回路規模を削減するぞ !!
という，強い意思でしばらくすすめてみよう，かと．

で，とりあえず，敵を知り己を知れば云々に従って，

- どんな回路が FPGA で小さく実装できるか
- JavaRock のどの部分が大きさに影響しているか

を調べながら，削減方法を探ってみることに．

昨夜，JavaRock の発表に際して共著の船田さんと，
FPGA で LUT は高価だよね，マルチプレクサで LUT 消費されるよね，
という話になったので，マルチプレクサに着目してみる．
マルチプレクサといえば，まず思いつくのがステートマシンの駆動部分．
JavaRock では，基本的に各文をステートに分割したステートマシンを構築するという
シンプルな変換をしているので，これが大きいのではないかと予想してみる．

さて，まずは，↓のような Java プログラムをコンパイルしてみることを考える．

```
public class for_test{
  int a;
  int b;
  public int test(int x){
    for(int i = 0; i < x; i++){
      a++;
    }
    for(int i = 0; i < x; i++){
      for(int j = 0; j < x; j++){
  b++;
      }
    }
    return a + b;
  }
}
```

これを普通に JavaRock でコンパイルして合成すると，↓のような回路になる．

おおまかな構造としては，↓のようなステートマシンになっているので，
とりあえず二つの for ループを別の process 文として外にだしてみることにする．
点線のところは，それぞれ，矢印の元のステートを監視して，
独立に駆動するようにしているので，メソッド呼出しと違って，呼出しにかかるサイクルのオーバヘッドはない．

で合成したみたときの回路は↓のような感じ．

JavaRock の生成した回路の真ん中にあるでかい回路がなくなって,
なんとなく回路がすっきりしているように，みえる.

Synthesis したとこでの結果は次の通り.

	#. Slice Regs	#. Slice LUTs
オリジナル	299	526
改変後 (一つ目の for ループのみ)	291	494
改変後 (二つの for ループとも)	310	538
改変後 (二つの for ループの最内ループも)	313	540

一つ目の for ループを改変した段階では効果ありか，とか思ったけど ... いまいちだな.

もう少し調べつつ，実装を検討しよう.
ちなみに，別プロセスに分けた変数同士に別々のレジスタを割り当てないといけないので，その点の考慮が必要.
前向きに考えれば，レジスタで処理を区切れるのでパイプライン化という方向にはすすみやすい.
また，逆に，分けたプロセスが，完全に独立している場合には，並列実行という方向も考えられる.