■ Bluespec で sort

そういえば Bluepsec デザインコンテスト向けに書いたソート回路を
公開しようと思って忘れていた.

トップモジュール

普通のソート

サンプルで示されているバブルソートは，入力が 5 に固定されている.
より柔軟に活用可能なモジュールにするため,
インスタンス時に入力サイズをパラメタで指定できるように修正する.
具体的には BubSort_ifc にサイズを指定できるよう以下のように変更を加えた.

```
interface BubSort_ifc#(type size_t);
   method Action start(Vector #(size_t, int) a);
   method Vector#(size_t, int) result();
endinterface
```

また, ルールを生成する部分のイタレーションの定数を, 型から値を取り出す valueOf を使用することで,

```
   for(Integer i = 0; i < valueof(size_t) - 1; i = i + 1) begin
      rule swap((x[i] > x[i+1]));
         x[i] <= x[i + 1];
         x[i+1] <= x[i];
      endrule
   end
```

と修正した．これによりモジュールをインスタンス化する時に,

```
BubSort_ifc#(5) sort <- mkBubSort();
```

などのように入力するデータの個数を指定できるようになった.

基数ソート

基数ソートは，ソートアルゴリズムの一つで，データをキーに分類して順にソートする.
各キーをソートするアルゴリズムの計算量が O(n) であるとき, 基数ソートの計算量は O(nk) であり,
逐次プロセッサで実行する場合には高速なアルゴリズムである.
Verilog や VHDL では，各ステップを処理する状態を明示的に管理する必要があるが,
Bluespec では，FSM クラスを用いることで容易に実装することができる.

基数ソートでは，データをキーに分類し，各キー毎にソートする．
今回は，入力データを 4bit 桁毎に分類する．すべてのキーに対して処理を順次行う必要があるため，
バブルソートと違い，入力データの幅が必要になる．
従って,次のように入力データの個数数 size_t とその幅 width をパラメタに取るモジュールとして定義する．
ただし，簡単のため，入力データの幅は 4 の倍数で与えられることを仮定する．

```
interface RadixSort_ifc#(type size_t,
                         numeric type width);
   method Action start(Vector#(size_t, int) a);
   method Vector#(size_t, int) result();
endinterface
```

全てのキーに対してソートを実行するステートマシン (外側ステートマシン) と，
各キーに対するソートを行うステートマシン (内側ステートマシン) の二段階で構成する．

○ 外側ステートマシン
外側ステートマシンの定義は次の通り．ここで fsm1 は, 各キーのソートを行うステートマシンであり，
すべてのキーをソートしおわるまで，内側ステートマシンを起動し続ける．

```
Stmt radix_seq = seq
                   while(base < fromInteger(valueOf(width)))
                      fsm1.start();
                   end_flag <= True;
                 endseq;
FSM fsm0 <- mkFSM(radix_seq);
```

○ 内側ステートマシン
全体の計算量を O(kn) にするためには，各キーをソートする内側ステートマシンは，
計算量 O(n) のものを選ぶ必要がある．バブルソートでもよいが，
ここでは，Bluespec での記述の容易性をみるため，少し複雑なバケットソートを定義してみる．
バケットソートの各手順を function として定義することで，ステートマシンは次のように記述できる．

```
Stmt sort_seq = seq
                  countElement();
                  setElementOffset();
                  for(index <= 0;
                      index < fromInteger(valueOf(size_t));
                      index <= index + 1)
                  doSortBody();
                  clean_called.send();
                endseq;
FSM fsm1 <- mkFSM(sort_seq);
```

逐次版・基数ソート (入力データの個数に回路規模が比例しない版)

基数ソートの実装に用いたバケットソートでは，各バケットの個数の数え上げを次のように実装していた．
これは，入力データ数 size_t が大きくなると，回路サイズが大きくなることが懸念される．

そこで，これを入力データを逐次的に検査するように明示的に記述することで，
与えられたデータの個数に比例して回路サイズが増大することを防止する
「逐次版・基数ソート」を実装することを考える．

```
function countElement();
   action
      int sum = 0;
      for(int i = 0; i < fromInteger(16); i = i + 1) begin
         sum = 0;
         for(Integer j = 0; j < valueof(size_t); j = j + 1) begin
            if(((src[j] >> base) & 'hF) == i) sum = sum + 1;
         end
         count[i] <= sum;
      end
   endaction
endfunction
```

逐次版･基数ソートでは，基数 / バケットソートの処理を司さどるトップモジュール RadixSort_ifc と，
バケットソートのために各バケットの要素の数え上げを行うモジュール CountElement_ifc から成る．
ここでは，CountElement_ifc の実装方法とインスタンス化手法について述べる．

○ CountElement_ifc の実装

基数 4bit つまり 0 〜 15 の値をキートする基数ソートのためのバケットソートでは，
0 〜 15 までの各バケットに相当する値の個数をカウントする必要がある．
そこであるバケットに対する個数をカウントするための

```
interface CountElement_ifc#(type size_t);
   method Action start(Vector#(size_t, int) a, int idx, int bs);
   method int result();
endinterface
```

なるモジュールを定義する．
size_t は入力変数の個数である．start メソッドが数え上げを実行するためのメソッドであり，
担当するバケットの値を index にセットし，次のステートマシンを起動することで逐次的に数え上げを実行する．

```
Stmt stmt
= seq
     end_flag <= False;
     for(i <= 0; i < fromInteger(valueOf(size_t)); i <= i + 1)
        if(((src[i] >> base) & 'hF) == index)
           count <= count + 1;
     end_flag <= True;
endseq;
FSM fsm <- mkFSM(stmt);
```

入力されるデータの個数である size_t まで逐次的にカウンタをインクリメントするため，
この回路サイズは入力されるデータの個数によらず一定である．

○ CountElement_ifc のインスタンス化
各バケットに対応する個数をカウントするため，
0 〜 15 までに対応する CountElement_ifc のインスタンスが必要である．

そこでトップモジュールである RadixSort_ifc で,

```
Vector#(16, CountElement_ifc#(size_t)) count <- replicateM(mkCountElement());
```

としてインスタンス化する.
これにより，16 個の CountElement_ifc をインスタンス化する記述を一つに簡略化できる.
また，数え上げを実行するには，Vector では各要素に [i] などとしてアクセスすることができる.
この添字アクセスして得られる値は，型 CountElement_ifc のインスタンスであるから,
メソッド呼び出しができる．つまり，次のように for ループで簡潔な記述ができる.

```
for(Integer i = 0; i < 16; i = i + 1) begin
   count[i].start(readVReg(src), fromInteger(i), base);
end
```

■一言メモ

- iPhone Application の講義資料か．http://www.stanford.edu/class/cs193p/cgi-bin/drupal/downloads-2010-winter (Fri Aug 20 15:37:24 2010 +0900)