```cpp
#ifndef Modules_HitRecoModule_hh
#define Modules_HitRecoModule_hh
//==========================================================================
// HitRecoModule.hh
// Module for reconstructing hits from strip infomation
//==========================================================================

#include "Framework/include/Module.hh"
#include "DataObjects/include/StripSet.hh"
#include "TVector3.h"
#include <vector>
#include <string>

namespace fc {

  class DetectorGeometry;
  class Hit;
  class HitSet;
  struct SensorDescriptor;

  class HitRecoModule : public Module {

   public:

    HitRecoModule(int, const std::string& iInputStripsLabel,
                  const std::string& iOutputHitsLabel, const DetectorGeometry&);

    void processEvent(Event&) override;

   private:

    int const _debugLevel;
    std::string const _inStripsLabel;
    std::string const _outHitsLabel;
    DetectorGeometry const& _detectorGeometry;

    int _num = 0;

    void processLayers(const StripSet&, HitSet&) const;
    void makeHits(int layer, LayerStripMap const& strips, HitSet& hits) const;
  };
} // end namespace fc

#endif // Modules_HitRecoModule_hh
```

```cpp
#include <algorithm>
#include <iostream>
#include <iterator>
#include <numeric>

#include "Geometry/include/StripHitFunctions.hh"
#include "Geometry/include/DetectorGeometry.hh"
#include "DataObjects/include/Hit.hh"
#include "DataObjects/include/HitSet.hh"
#include "DataObjects/include/StripSet.hh"
#include "Modules/include/HitRecoModule.hh"
#include "Modules/include/Strip.hh"
#include "Modules/include/Layer.hh"
#include "Modules/include/HitAccum.hh"

fc::HitRecoModule::HitRecoModule(int debugLevel,
                                 const std::string& iInputStripsLabel,
                                 const std::string& iOutputHitsLabel,
                                 const DetectorGeometry& detectorGeometry):
  _debugLevel(debugLevel),
  _inStripsLabel(iInputStripsLabel),
  _outHitsLabel(iOutputHitsLabel),
  _detectorGeometry(detectorGeometry) {
}

void fc::HitRecoModule::processEvent(fc::Event& event) {
  Handle<StripSet> const genStripSet = event.get<StripSet>(_inStripsLabel);
  std::unique_ptr<HitSet> recoHitSet(new HitSet);

  processLayers(*genStripSet, *recoHitSet);

  if (_debugLevel >= 2) { recoHitSet->print(std::cout); }

  event.put(_outHitsLabel, std::move(recoHitSet));
  ++_num;
}

void fc::HitRecoModule::processLayers(const StripSet& stripSet,
                                      HitSet& hitSet) const {
  auto const& layers = stripSet.getStrips();
  for (size_t i = 0; i < layers.size(); ++i) {
    std::cout << "calling makeHits for layer " << i << "\n";
    makeHits(i, layers[i], hitSet);
  }
}

void fc::HitRecoModule::makeHits(int layer,
                                 LayerStripMap const& strips,
                                 HitSet& hits) const {
  // LayerStripMap is not a convenient structure for our use, so we
  // transform it here, to a vector of pairs. The vector is sorted,
  // which we rely upon.
  Layer currentLayer(layer);
  std::transform(begin(strips), end(strips),
                 std::back_inserter(currentLayer.strips),
  [](LayerStripMap::value_type p) { return Strip(p.first, p.second); });

  HitAccum currentCluster(layer, _detectorGeometry, hits);

  for (auto const& strip : currentLayer.strips) {
    currentCluster.processStrip(strip);
  }
}
```

```cpp
#ifndef Modules_HitAccum_hh
#define Modules_HitAccum_hh

#include <vector>
#include "DataObjects/include/HitSet.hh"
#include "Geometry/include/DetectorGeometry.hh"
#include "Modules/include/Strip.hh"

namespace fc {
  // This structure is a helper that is used to accumulate hits within
  // one layer.

  struct HitAccum {

    // Mutable data, representing the current state of the accumulation
    // algorithm.
    int start;
    int curr_strip;
    std::vector<int> cnts;
    HitSet&          hits;

    // Immutable data, representing the context in which this HitAccum
    // object is working.
    int const               layer;
    DetectorGeometry const& geom;
    SensorDescriptor const& desc;

    // C'tor establishes our working context. This accumulator will put
    // hits into 'results'.
    HitAccum(int lyr, DetectorGeometry const& dg, HitSet& results);

    // D'tor makes sure that the last hit is completed.
    ~HitAccum();

    void clear() { start = -1; curr_strip = -1; cnts.clear(); }
    void add(Strip const& s) { curr_strip = s.id, cnts.push_back(s.cnt); }
    void startNewHit(int strip, int cnt) { start = strip; curr_strip = strip; cnts.push_back(cnt); }
    bool isAdjacent(int strip) const { return strip == curr_strip + 1; }
    bool makingCluster() const { return start >= 0; }

    // This member function defines what it means to be a strip that is a
    // candidate to go into a cluster.
    bool goodStrip(Strip const& s) const { return s.cnt > desc._threshold; }

    // This member function defines what it means for a strip to be
    // appropriate to add to the current cluster.
    bool inSameCluster(Strip const& s) const { return goodStrip(s) && isAdjacent(s.id); }

    void makeHit();
    void processStrip(Strip const& strip);
  };
}

#endif
```

```cpp
#include <algorithm>
#include <iostream>

#include "Modules/include/HitAccum.hh"
#include "Geometry/include/StripHitFunctions.hh"

fc::HitAccum::HitAccum(int lyr, DetectorGeometry const& dg, HitSet& results) :
  start(-1),
  curr_strip(-1),
  cnts(),
  hits(results),
  layer(lyr),
  geom(dg),
  desc(dg.getSensor(layer))
{ }

fc::HitAccum::~HitAccum() {
  if (makingCluster()) { makeHit(); }
}

void fc::HitAccum::makeHit() {
  if (cnts.size() < 2) { return; }
  double const pos = fcf::calculateStripHitPositionFromCluster(start, cnts);
  double const local = fcf::calculateLocalFromStripPosition(pos, layer, geom);
  TVector3 const hpos = fcf::calculateGlobalFromLocalPosition(local, layer, geom);
  int const charge = std::accumulate(cnts.begin(), cnts.end(), 0);
  bool const good_hit = cnts.size() <= 2 && charge <= geom.getMIP();
  double const res = good_hit ? desc._hitResolution : desc._badHitResolution;
  hits.insertHit(Hit(hpos, layer, cnts.size(), charge, good_hit, res));

  std::cout << "closing:"
            << " layer " << layer
            << " strip " << start
            << " hits " << cnts.size()
            << " charge " << charge
            << " good " << good_hit
            << " pos " << hpos.x() << " " << hpos.y() << " " << hpos.z()
            << "\n";
  clear();
}

void fc::HitAccum::processStrip(Strip const& strip) {
  if (makingCluster()) {
    // ongoing cluster ...
    if (inSameCluster(strip)) {
      add(strip);
    }
    else {
      makeHit();
      startNewHit(strip.id, strip.cnt);
    }
  }
  else {
    // no cluster ...
    if (goodStrip(strip)) {
      startNewHit(strip.id, strip.cnt);
    }
  }
}
```

```cpp
#ifndef Modules_Strip_hh
#define Modules_Strip_hh

namespace fc {
  // An instance of Strip represents a strip with a signal. We use small
  // integers for data members because their range is small. Our c'tor
  // has to use what the surrounding code uses.
  struct Strip {
    Strip(int i, int adc) : id(i), cnt(adc) { }
    unsigned short id;
    unsigned short cnt;
  };
}

#endif
```

```cpp
#ifndef Modules_Layer_hh
#define Modules_Layer_hh

#include <vector>
#include "Modules/include/Strip.hh"

namespace fc {
  struct Layer {
    explicit Layer(int lyr) : strips(), layer(lyr) { }
    std::vector<Strip>   strips;
    unsigned short const layer;
  };
}

#endif
```