# Overhaul of EMTF emulator

Sept 26, 2016

University of Florida
Darin Acosta, Andrew Brinkerhoff, Matt Carver, Jia Fu Low, Alex Madorsky

# Introduction

- Goals
  - Provide the same accuracy and functionality as Matt's emulator
  - Follow Alex's firmware as much as possible including all the integer operations
  - Restructure the emulator to look more similar to the old CSCTF
  - Provide flexibility for future upgrades, e.g. adding RPC hits
- Status
  - Decided to rewrite from scratch (Sept 1)
    - Easier to restructure, also allow myself to understand fully the codes (software + firmware)
    - A lot of codes are actually borrowed from Matt's emulator
  - As of today, first alpha version available
    - Based on Alex's “emtf_core_modelsim_1” firmware version
    - Based on Matt's “EmuAccuracy” branch
    - Still missing many features in/built around Matt's emulator
  - In the coming weeks
    - Update to Alex's “emtf_core_modelsim_2nd_earliest” firmware version
    - Update to cms-l1t-offline integration branch
    - If possible, will put a switch that allows the user to pick the version
    - Check against unpacker tracks (I'm sure there will be many corner cases to be discovered)

# Codes

- Override two packages:
  - DataFormats/L1TMuon
    - EMTFHit, EMTFHitExtra
    - EMTFTrack, EMTFTrackExtra
  - L1Trigger/L1TMuonEndCap
    - L1TMuonEndCapProducer
    - + almost everything else
- Code checkout:
  - https://github.com/jiafulow/DataFormatsSep2016
  - https://github.com/jiafulow/L1TriggerSep2016
- The new emulator should be a drop-in replacement (at least that's the goal)
  - They can be used in any CMSSW release without worrying about merge conflicts
    - Physical locations differ from the existing L1Trigger and DataFormats with the suffix "Sep2016"
  - Certain classes are reused (with modifications), but wrapped under “L1TMuonEndCap” namespace to avoid name collisions
    - EMTFHit, EMTFHitExtra, EMTFTrack, EMTFTrackExtra, MuonTriggerPrimitive, MuonTriggerPrimitiveFwd, EMTFHitTools, EMTFTrackTools
    - Except L1TMuonEndCapProducer is renamed to L1TMuonEndCapProducerSep2016

# Differences

- Main differences w.r.t. current emulator
  - I try to follow very closely Alex's firmware, matching the integer variables where possible.
  - On the flip side, this means that the emulator doesn't really provide cross checks that Matt's emulator provides.
    - I switched off my brain about track finding, just doing C++
  - In terms of structure, the biggest changes are
    - BX processing more similar to the old CSCTF (more later)
    - Consolidating functions under the “sector processor”
  - Though, I'm aiming for 99.9% accuracy, not 100%, so I use C++ sorting instead of the firmware sorting in certain places
    - But not in the best track selection, where I try to preserve the order.

# DataFormats/L1TMuon

- EMTFHit & EMTFTrack remain the same, but put under "L1TMuonEndCap" namespace instead
- EMTFHitExtra & EMTFTrackExtra are completely revamped
- Also added new classes: EMTFRoadExtra, EMTFPtLUTData
- Need some discussion
  - At the moment, it's very flexible

EMTFHitExtra

```
struct EMTFHitExtra {

  EMTFHitExtra() : endcap(0), station(0), ring(0), chamber(0), sector(0), subsector(0), csc_ID(0), cscn_ID(0),
                   bx(0), subsystem(0), pc_station(0), pc_chamber(0),
                   valid(0), strip(0), wire(0), quality(0), pattern(0), bend(0),
                   phi_fp(0), theta_fp(0), phzvl(0), ph_hit(0), zone_hit(0), zone_code(0),
                   bc0(0), mpc_link(0), sync_err(0), track_num(0), stub_num(0), bx0(0), layer(0)
                   {}

  // DetId
  int16_t endcap;
  int16_t station;
  int16_t ring;
  int16_t chamber;
  int16_t sector;
  int16_t subsector;
  int16_t csc_ID;
  int16_t cscn_ID;

  // BX
  int16_t bx;

  // Subsystem
  int16_t subsystem;

  // Station and chamber in firmware
  uint16_t pc_station;
  uint16_t pc_chamber;

  // Input to PrimitiveConversion
  uint16_t valid;
  uint16_t strip;
  uint16_t wire;
  uint16_t quality;
  uint16_t pattern;
  uint16_t bend;

  // Output from PrimitiveConversion
  uint16_t phi_fp;
  uint16_t theta_fp;
  uint16_t phzvl;
  uint16_t ph_hit;
  uint16_t zone_hit;
  uint16_t zone_code;

  // Other
  uint16_t bc0;
  uint16_t mpc_link;
  uint16_t sync_err;
  uint16_t track_num;
  uint16_t stub_num;
  uint16_t bx0;
  uint16_t layer;

};  // class EMTFHitExtra
```

# DataFormats/L1TMuon

- EMTFHit & EMTFTrack remain the same, but put under "L1TMuonEndCap" namespace instead
- EMTFHitExtra & EMTFTrackExtra are completely revamped
- Also added new classes: EMTFRoadExtra, EMTFPtLUTData
- Need some discussion
  - At the moment, it's very flexible

EMTFTrackExtra

```
struct EMTFTrackExtra {

  EMTFTrackExtra() : endcap(0), sector(0), bx(0), first_bx(0), second_bx(0),
                     rank(0), winner(0), mode(0), mode_inv(0),
                     ptlut_address(0),
                     pt(0.), pt_xml(0.),
                     pt_int(0), phi_int(0), theta_int(0),
                     gmt_phi(0), gmt_eta(0), gmt_quality(0), gmt_charge(0),
                     ptlut_data(),
                     xroad(),
                     num_xhits(0),
                     xhits(), xhits_ph_diff()
                     {}

  // DetId
  int16_t endcap;
  int16_t sector;

  // BX
  int16_t bx;
  int16_t first_bx;
  int16_t second_bx;

  // Rank
  uint16_t rank;
  uint16_t winner;  // 0: first winner, 1: second winner, ...

  // Mode
  uint16_t mode;
  uint16_t mode_inv;

  // pT LUT address
  uint64_t ptlut_address;

  // pT LUT output
  float    pt;
  float    pt_xml;

  // Momentum
  uint32_t pt_int;
  uint32_t phi_int;
  uint32_t theta_int;

  // GMT
  int gmt_phi;
  int gmt_eta;
  int gmt_quality;
  int gmt_charge;

  // pT LUT data
  EMTFPtLUTData ptlut_data;

  // Road
  EMTFRoadExtra xroad;

  // Number of hits
  uint32_t num_xhits;

  // Hits
```

# L1Trigger/L1TMuonEndCap

- Main driver is L1Trigger/L1TMuonEndCap/python/simEmtfDigis_cfi.py
- Added many configurable parameters: pattern definitions, zone definitions, theta window, etc
- Call L1TMuonEndCapTrackProducerSep2016

simEmtfDigis_cfi.py

```
simEmtfDigis = cms.EDProducer("L1TMuonEndCapTrackProducerSep2016",
    # Verbosity level
    verbosity = cms.untracked.int32(0),

    # Input collections
    CSCInput = cms.InputTag('simCscTriggerPrimitiveDigis','MPCSORTED'),
    RPCInput = cms.InputTag('simMuonRPCDigis'),
    #GEMInput = cms.InputTag('simMuonGEMPadDigis'),

    # Run with CSC, RPC
    CSCEnable = cms.bool(True),
    RPCEnable = cms.bool(False),

    # LUT files
    PhThLUT = cms.string('ph_lut_v1'),

    # Sector processor primitive-conversion parameters
    spPCParams16 = cms.PSet(
        IncludeNeighbor = cms.bool(True),
        DuplicateWires = cms.bool(True),
    ),

    # Sector processor pattern-recognition parameters
    spPRParams16 = cms.PSet(
        MinBX = cms.int32(-3),
        MaxBX = cms.int32(+4),
        BXWindow = cms.int32(3),
        ZoneBoundaries1 = cms.vint32(0,42,50,88),
        ZoneBoundaries2 = cms.vint32(41,49,87,127),
        ZoneOverlap = cms.int32(2),
        PatternDefinitions = cms.vstring(
            # straightness, hits in ME1, hits in ME2, hits in ME3, hits in ME4
            "4,15:15,7:7,7:7,7:7",
            "3,16:16,7:7,7:6,7:6",
            "3,14:14,7:7,8:7,8:7",
            "2,18:17,7:7,7:5,7:5",
            "2,13:12,7:7,10:7,10:7",  # should be 9:7 in ME3,4
            "1,22:19,7:7,7:0,7:0",
            "1,11:8,7:7,14:7,14:7",
            "0,30:23,7:7,7:0,7:0",
            "0,7:0,7:7,14:7,14:7",
        ),
        MaxRoadsPerZone = cms.int32(3),
        ThetaWindow = cms.int32(4),
        MaxTracks = cms.int32(3),
    ),

    # Sector processor pt-assignment parameters
    spPAParams16 = cms.PSet(
        TreeVer = cms.string('v_16_02_21'),
    ),

    # Sector processor ghost-cancellation parameters
    spGCParams16 = cms.PSet(

    ),
)
```

# L1Trigger/L1TMuonEndCap

L1TMuonEndCapTrackProducer

- L1TMuonEndCapTrackProducerSep2016 consists of 3 member objects
- EMTFTrackFinder does all the work and produces EMTFHitExtraCollection and EMTFTrackExtraCollection
- EMTFTrackAdaptor adapts the collections into EMTFHitCollection and EMTFTrackCollection
- EMTFMicroGMTConverter adapts the EMTFTrackExtraCollection into uGMT format

```
// Class declaration
class L1TMuonEndCapTrackProducer : public edm::EDProducer {
public:
  explicit L1TMuonEndCapTrackProducer(const edm::ParameterSet&);
  ~L1TMuonEndCapTrackProducer();

  static void fillDescriptions(edm::ConfigurationDescriptions& descriptions);

private:
  virtual void beginJob() override;
  virtual void produce(edm::Event&, const edm::EventSetup&) override;
  virtual void endJob() override;

  //virtual void beginRun(edm::Run const&, edm::EventSetup const&);
  //virtual void endRun(edm::Run const&, edm::EventSetup const&);
  //virtual void beginLuminosityBlock(edm::LuminosityBlock const&, edm::EventSetup const&);
  //virtual void endLuminosityBlock(edm::LuminosityBlock const&, edm::EventSetup const&);

private:
  std::unique_ptr<EMTFTrackFinder>       track_finder_;
  std::unique_ptr<EMTFTrackAdaptor>      track_adaptor_;
  std::unique_ptr<EMTFMicroGMTConverter> uGMT_converter_;

  const edm::ParameterSet& config_;
};
```

```
void L1TMuonEndCapTrackProducer::produce(edm::Event& iEvent, const edm::EventSetup& iSetup) {
  // Create pointers to output products
  auto out_xhits   = std::make_unique<EMTFHitExtraCollection>();
  auto out_xtracks = std::make_unique<EMTFTrackExtraCollection>();
  auto out_cands   = std::make_unique<l1t::RegionalMuonCandBxCollection>();

  auto out_hits    = std::make_unique<EMTFHitCollection>();
  auto out_tracks  = std::make_unique<EMTFTrackCollection>();

  // Run
  track_finder_->process(iEvent, iSetup, *out_xhits, *out_xtracks);

  // Put into old EMTFHit, EMTFTrack formats
  track_adaptor_->convert_all(*out_xhits, *out_xtracks, *out_hits, *out_tracks);

  // Put into uGMT format
  uGMT_converter_->convert_all(*out_xtracks, *out_cands);

  // Fill the output products
  iEvent.put(std::move(out_xhits)  , "");
  iEvent.put(std::move(out_xtracks), "");
  iEvent.put(std::move(out_cands)  , "EMTF");

  iEvent.put(std::move(out_hits)   , "");
  iEvent.put(std::move(out_tracks) , "");
}
```

# L1Trigger/L1TMuonEndCap



- EMTFTrackFinder parses input parameters, instantiates sector processors, LUTs, pT assignment “engine”
- Then, in each event, call the sector processors

```
try {
  // Configure sector processor LUT
  sector_processor_lut_.read(ph_th_lut_);

  // Configure pT assignment engine
  pt_assignment_engine_.read(tree_ver);

  // Configure sector processors
  for (int endcap = MIN_ENDCAP; endcap <= MAX_ENDCAP; ++endcap) {
    for (int sector = MIN_TRIGSECTOR; sector <= MAX_TRIGSECTOR; ++sector) {
      sector_processors_.push_back(EMTFSectorProcessor());

      sector_processors_.back().configure(
          &sector_processor_lut_,
          &pt_assignment_engine_,
          endcap, sector,
          includeNeighbor, duplicateWires,
          minBX, maxBX, bxWindow,
          zoneBoundaries1, zoneBoundaries2, zoneOverlap,
          pattDefinitions,
          maxRoadsPerZone, thetaWindow, maxTracks
      );
    }
  }
} catch (...) {
  throw;
}
```

```
// ____________________________________________________________________________
// Run each sector processor

for (int endcap = MIN_ENDCAP; endcap <= MAX_ENDCAP; ++endcap) {
  for (int sector = MIN_TRIGSECTOR; sector <= MAX_TRIGSECTOR; ++sector) {
    int es = (endcap-1) * 6 + (sector-1);

    sector_processors_.at(es).process(
        iEvent.id().event(),
        muon_primitives,
        out_hits,
        out_tracks
    );
  }
}
```

# L1Trigger/L1TMuonEndCap

EMTFSectorProcessor

- EMTFSectorProcessor will run N times, where N = maxBX - minBX + BXWindow (=2)
- During each pass, it does primitive selection, primitive conversion, pattern recognition, primitive matching, angle calculation, best track selection, pT assignment

```
void EMTFSectorProcessor::process(
    EventNumber_t ievent,
    const TriggerPrimitiveCollection& muon_primitives,
    EMTFHitExtraCollection& out_hits,
    EMTFTrackExtraCollection& out_tracks
) const {

  //if (!(endcap_ == 1 && sector_ == 2))  return;  // debug

  // List of converted hits, extended from previous BXs
  std::deque<EMTFHitExtraCollection> extended_conv_hits;

  // Map of pattern detector --> lifetime, tracked across BXs
  std::map<EMTFPatternId, int> patt_lifetime_map;

  int delayBX = bxWindow_ - 1;  // = 2

  for (int ibx = minBX_; ibx <= maxBX_ + delayBX; ++ibx) {
    if (true) {  // debug
      std::cout << "Endcap: " << endcap_ << " Sector: " << sector_ << " Event: " << ievent << " BX: " << ibx << std:
    }

    process_single_bx(ibx, muon_primitives, out_hits, out_tracks, extended_conv_hits, patt_lifetime_map);

    if (ibx >= minBX_ + delayBX) {
      extended_conv_hits.pop_front();
    }
  }

  return;
}
```

# L1Trigger/L1TMuonEndCap

EMTFSectorProcessor

- EMTFSectorProcessor will run N times, where N = maxBX - minBX + BXWindow (=2)
- During each pass, it does primitive selection, primitive conversion, pattern recognition, primitive matching, angle calculation, best track selection, pT assignment

```
// _____________________________________________________________________
// Configure

EMTFPrimitiveSelection prim_sel;
prim_sel.configure(
    endcap_, sector_, bx,
    includeNeighbor_, duplicateWires_
);

EMTFPrimitiveConversion prim_conv;
prim_conv.configure(
    lut_,
    endcap_, sector_, bx,
    zoneBoundaries1_, zoneBoundaries2_, zoneOverlap_
);

EMTFPatternRecognition patt_recog;
patt_recog.configure(
    endcap_, sector_, bx,
    minBX_, maxBX_, bxWindow_,
    pattDefinitions_, maxRoadsPerZone_
);

EMTFPrimitiveMatching prim_match;
prim_match.configure(
    endcap_, sector_, bx
);

EMTFAngleCalculation angle_calc;
angle_calc.configure(
    endcap_, sector_, bx,
    thetaWindow_
);

EMTFBestTrackSelection btrack_sel;
btrack_sel.configure(
    endcap_, sector_, bx,
    maxRoadsPerZone_, maxTracks_
);

EMTFPtAssignment pt_assign;
pt_assign.configure(
    pt_assign_engine_,
    endcap_, sector_, bx
);
```

# L1Trigger/L1TMuonEndCap

EMTFPtAssignment

- EMTFPtAssignment calls EMTFPtAssignmentEngine which does all the heavy work
- Can have more than one engine

```
void EMTFPtAssignment::process(EMTFTrackExtraCollection& best_tracks) {
  using address_t = EMTFPtAssignmentEngine::address_t;

  const int ntracks = best_tracks.size();

  for (int i = 0; i < ntracks; ++i) {
    EMTFTrackExtra& track = best_tracks.at(i);  // pass by reference

    address_t address = pt_assign_engine_->calculate_address(track);
    float     xmlpt   = pt_assign_engine_->calculate_pt(address, track);


    // convert phi into gmt scale according to DN15-017
    // full scale is -16 to 100, or 116 values, covers range -10 to 62.5 deg
    // my internal ph scale is 0..5000, covers from -22 to 63.333 deg
    // converted to GMT scale it is from -35 to 95
    // bt_phi * 107.01/4096, equivalent to bt_phi * 6849/0x40000
    int gmt_phi_mult = track.phi_int * 6849;
    int gmt_phi = (gmt_phi_mult>>18); // divide by 0x40000
    gmt_phi -= 35; // offset of -22 deg

    int gmt_eta = getGMTEta(track.theta_int, endcap_);

    int gmt_quality = getGMTQuality(track.mode, track.theta_int);

    const EMTFPtLUTData& ptlut_data = track.ptlut_data;
    int gmt_charge = getGMTCharge(ptlut_data.ph[0], ptlut_data.ph[1], ptlut_data.ph[2], ptlut_data.ph[3], track.mode

    // ____________________________________________________________________________
    // Output
    track.ptlut_address = address;
    track.pt_xml        = xmlpt;
    track.pt            = xmlpt*1.4;

    track.gmt_phi       = gmt_phi;
    track.gmt_eta       = gmt_eta;
    track.gmt_quality   = gmt_quality;
    track.gmt_charge    = gmt_charge;
  }
```

# Accuracy?

- Testing against the firmware simulator, these 7 events match exactly from the converted hits to the pT XML address:
  - Run 278018, Event 1539957230
  - Run 278018, Event 1540061587
  - Run 278018, Event 1540745931
  - Run 278018, Event 1541093157
  - Run 278018, Event 1686541662
  - Run 278018, Event 1686648178
  - Run 278018, Event 1687278667

test_Event1687278667.py

```
def test_hits(self):
  hits = self.analyzer.handles["hits"].product()

  hit = hits[4]
  self.assertEqual(hit.phi_fp     , 3588)
  self.assertEqual(hit.theta_fp   , 32)
  self.assertEqual((1<<hit.ph_hit), 65536)
  self.assertEqual(hit.phzvl      , 1)

  hit = hits[5]
  self.assertEqual(hit.phi_fp     , 4314)
  self.assertEqual(hit.theta_fp   , 13)
  self.assertEqual((1<<hit.ph_hit), 1048576)
  self.assertEqual(hit.phzvl      , 1)

  hit = hits[6]
  self.assertEqual(hit.phi_fp     , 4043)
  self.assertEqual(hit.theta_fp   , 27)
  self.assertEqual((1<<hit.ph_hit), 8192)
  self.assertEqual(hit.phzvl      , 1)

  hit = hits[7]
  self.assertEqual(hit.phi_fp     , 4004)
  self.assertEqual(hit.theta_fp   , 25)
  self.assertEqual((1<<hit.ph_hit), 4096)
  self.assertEqual(hit.phzvl      , 1)

  hit = hits[8]
  self.assertEqual(hit.phi_fp     , 3869)
  self.assertEqual(hit.theta_fp   , 24)
  self.assertEqual((1<<hit.ph_hit), 256)
  self.assertEqual(hit.phzvl      , 1)

def test_tracks(self):
  tracks = self.analyzer.handles["tracks"].product()

  track = tracks[0]
  self.assertEqual(track.rank         , 14)
  self.assertEqual(track.mode         , 6)
  self.assertEqual(track.ptlut_address, 416285735)
```