Issue 118: Make Pulse-Fitting Fast Again

Sebouh Paul

July 12, 2017

Intro

Pulse-fitting takes up about 30% of the recon-time, according to Maurik's profiling

- Uses jMinuit to find the time of the start of pulses, t_0
- The following function calculates the amplitude at each sample for a given t_0:

  `org.hps.tracking.PulseShape.FourPole.getAmplitudePeakNorm(time)`
- This function is called inside of the jMinuit minimization routine. Optimization at two levels:
  - iss118: individual amplitudes calculated separately.
  - iss118a: multiple amplitudes calculated together.

Optimization Level 1: `getAmplitudePeakNorm(time)`

Original Code

From Master Branch

```
public static class FourPole extends PulseShape {

    private double tp;
    private double tp2;
    private double peak_t, peak_amp;

    @Override
    public void setParameters(int channel, HpsSiSensor sensor) {
        tp = sensor.getShapeFitParameters(channel)[HpsSiSensor.TP_INDEX];
        tp2 = sensor.getShapeFitParameters(channel)[HpsSiSensor.TP_INDEX + 1];
        peak_t = 3.0 * Math.pow(tp * Math.pow(tp2, 3), 0.25); //approximate solution to exp(x)=1+x+x^2*tp/(2*tp2), where x=(1/tp2-1/tp)*t
        peak_amp = getAmplitudeIntegralNorm(peak_t);
    }

    @Override
    public double getAmplitudeIntegralNorm(double time) {
        if (time < 0) {
            return 0;
        }
        //===> return (time / channelConstants.getTp()) * Math.exp(1 - time / channelConstants.getTp());
        return (Math.pow(tp, 2) / Math.pow(tp - tp2, 3))
                * (Math.exp(-time / tp)
                - Math.exp(-time / tp2) * (1 + time * (tp - tp2) / (tp * tp2) + 0.5 * Math.pow(time * (tp - tp2) / (tp * tp2), 2)));
          return (time / tp) * Math.exp(1 - time / tp);
    }

    @Override
    public double getAmplitudePeakNorm(double time) {
        return getAmplitudeIntegralNorm(time) / peak_amp;
    }
}
```

$$\frac{t'^2}{(t'-t'_2)^3}\left[e^{-t/t'} - e^{-t/t'_2}\left(1 + t\frac{t'-t'_2}{t't'_2} + 0.5\left(t\frac{t'-t'_2}{t't'_2}\right)^2\right)\right]$$

Optimization Level 1: `getAmplitudePeakNorm(time)`

Identify combinations of constants that are re-computed for every call to function:

```
public static class FourPole extends PulseShape {

    private double tp;
    private double tp2;
    private double peak_t, peak_amp;

    @Override
    public void setParameters(int channel, HpsSiSensor sensor) {
        tp = sensor.getShapeFitParameters(channel)[HpsSiSensor.TP_INDEX];
        tp2 = sensor.getShapeFitParameters(channel)[HpsSiSensor.TP_INDEX + 1];
        peak_t = 3.0 * Math.pow(tp * Math.pow(tp2, 3), 0.25); //approximate solution to exp(x)=1+x+x^2*tp/(2*tp2), where x=(1/tp2-1/tp)*t
        peak_amp = getAmplitudeIntegralNorm(peak_t);
    }

    @Override
    public double getAmplitudeIntegralNorm(double time) {
        if (time < 0) {
            return 0;
        }
        //===> return (time / channelConstants.getTp()) * Math.exp(1 - time / channelConstants.getTp());
        return (Math.pow(tp, 2) / Math.pow(tp - tp2, 3))
                * (Math.exp(-time / tp)
                - Math.exp(-time / tp2) * (1 + time * (tp - tp2) / (tp * tp2) + 0.5 * Math.pow(time * (tp - tp2) / (tp * tp2), 2)));
          return (time / tp) * Math.exp(1 - time / tp);
    }

    @Override
    public double getAmplitudePeakNorm(double time) {
        return getAmplitudeIntegralNorm(time) / peak_amp;
    }
}
```

$$A\left[e^{-t/t'} - e^{-t/t'_2}\left(1 + Bt + 0.5B^2t^2\right)\right],\ A = \frac{t'^2}{(t'-t'_2)^3},\ B = \frac{t'-t'_2}{t't'_2}$$

Optimization Level 1: `getAmplitudePeakNorm(time)`

Cache these values when setParameters(...) is called. Also, use multiplication instead of `Math.pow` function

```
    public static class FourPole extends PulseShape {

        private double tp;
        private double tp2;
        private double peak_t, peak_amp;

        //combinations of tp and tp2:
        private double A, B;

        @Override
        public void setParameters(int channel, HpsSiSensor sensor) {
            tp = sensor.getShapeFitParameters(channel)[HpsSiSensor.TP_INDEX];
            tp2 = sensor.getShapeFitParameters(channel)[HpsSiSensor.TP_INDEX + 1];
            peak_t = 3.0 * Math.pow(tp * Math.pow(tp2, 3), 0.25); //approximate solution to exp(x)=1+x+x^2*tp/(2*tp2), where x=(1/tp2-1/tp)*t

            A = (Math.pow(tp, 2) / Math.pow(tp - tp2, 3));
            B = (tp - tp2) / (tp * tp2);
            peak_amp = getAmplitudeIntegralNorm(peak_t);

        }

        @Override
        public double getAmplitudeIntegralNorm(double time) {
            if (time < 0) {
                return 0;
            }
            //===> return (time / channelConstants.getTp()) * Math.exp(1 - time / channelConstants.getTp());
            return A * (Math.exp(-time / tp)
                            - Math.exp(-time / tp2) * (1 + time * B + 0.5 * time * time * B*B));

//            return (time / tp) * Math.exp(1 - time / tp);
        }
```

Optimization Level 1: Continued

- `getAmplitudePeakNorm(time)` is called inside `getAmplitudeIntegralNorm(time)`
  - Can copy the code of the former and put it inside the latter (reduces the number of calls being made).

Optimization Level 2: Calculating Multiple Amplitudes

- The function`ShaperLinearFitAlgorithm.doLinFit(...)` calls `getAmplitudeIntegralNorm(time)` for several values of `time`, which are evenly spaced (since samples are evenly spaced in time)
  - We can take advantage of the fact that the values for which the amplitudes are calculated are evenly spaced:
    - Write new function that calculates amplitudes for multiple evenly-spaced time input values..
    - reduce number of calls to `Math.exp(...)`, since it is not very fast.

Testing performance

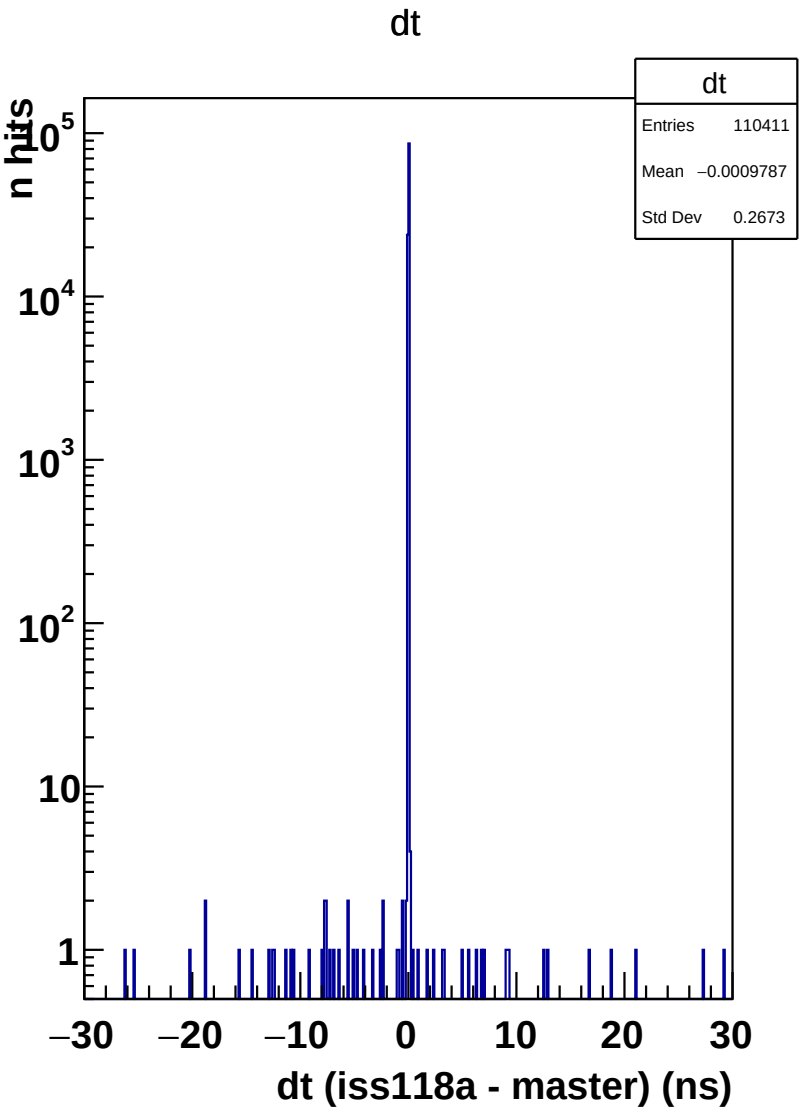
- Use first 5000 events of run 7796
- Run this with master branch, and with level 1 and level 2 optimizations.
- compare execution times.
  - total wall time
  - avg. cpu time per invocation of driver
- plot differences in amplitudes and t0s for each hit.
- compare some DQM plots between master and level 2 optimized recon outputs . (backup slides)

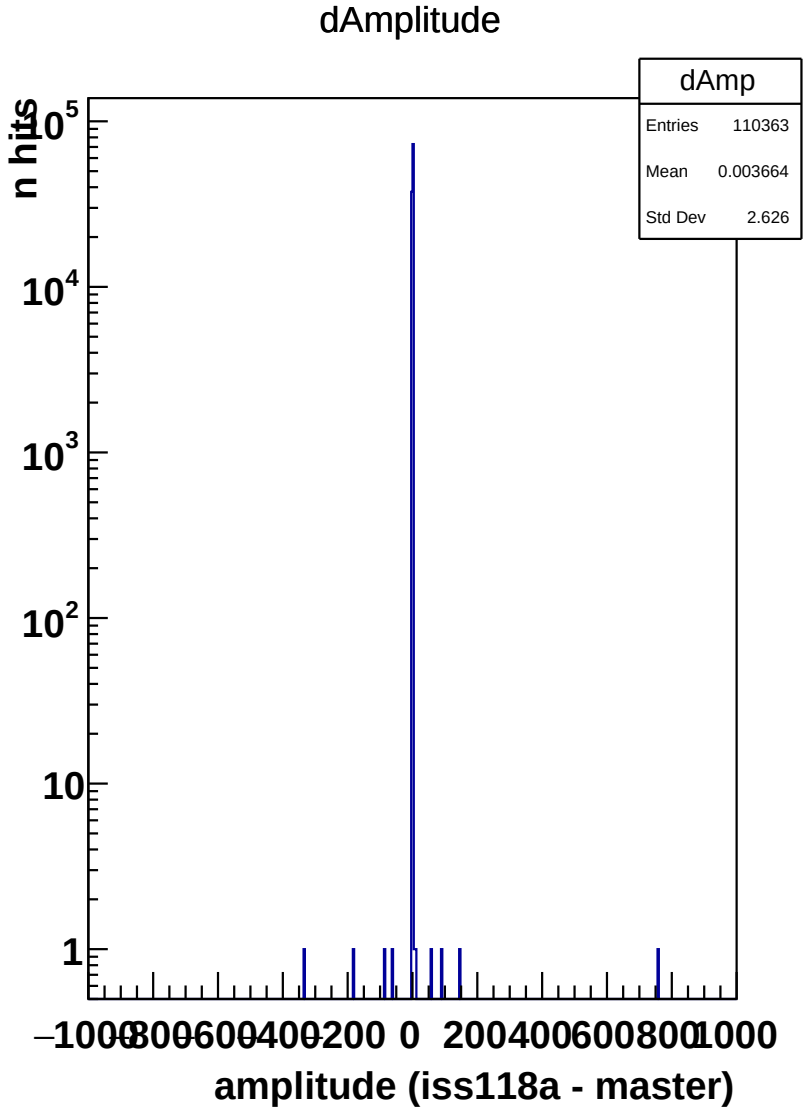
Execution time*

branch	tot. wall time*	%	avg cpu time**	% of master
master	9:08	100.0	102.7	100
iss118	8:29	92.8	100.7	98.1
iss118a	7:44	84.7	80.9	78.8

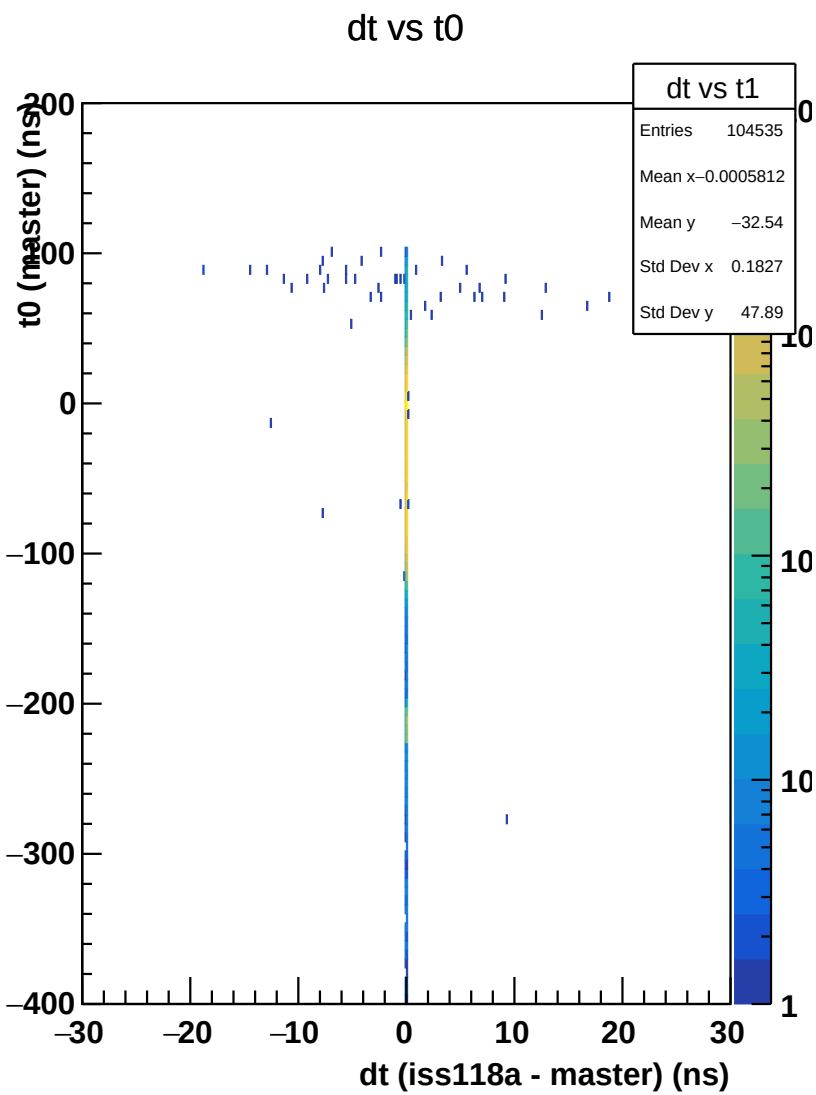
*difference between first and last timestamps printed from in the EventMarkerDriver (min:sec)
**per invocation of RawTrackerHitFitterDriver (ms)

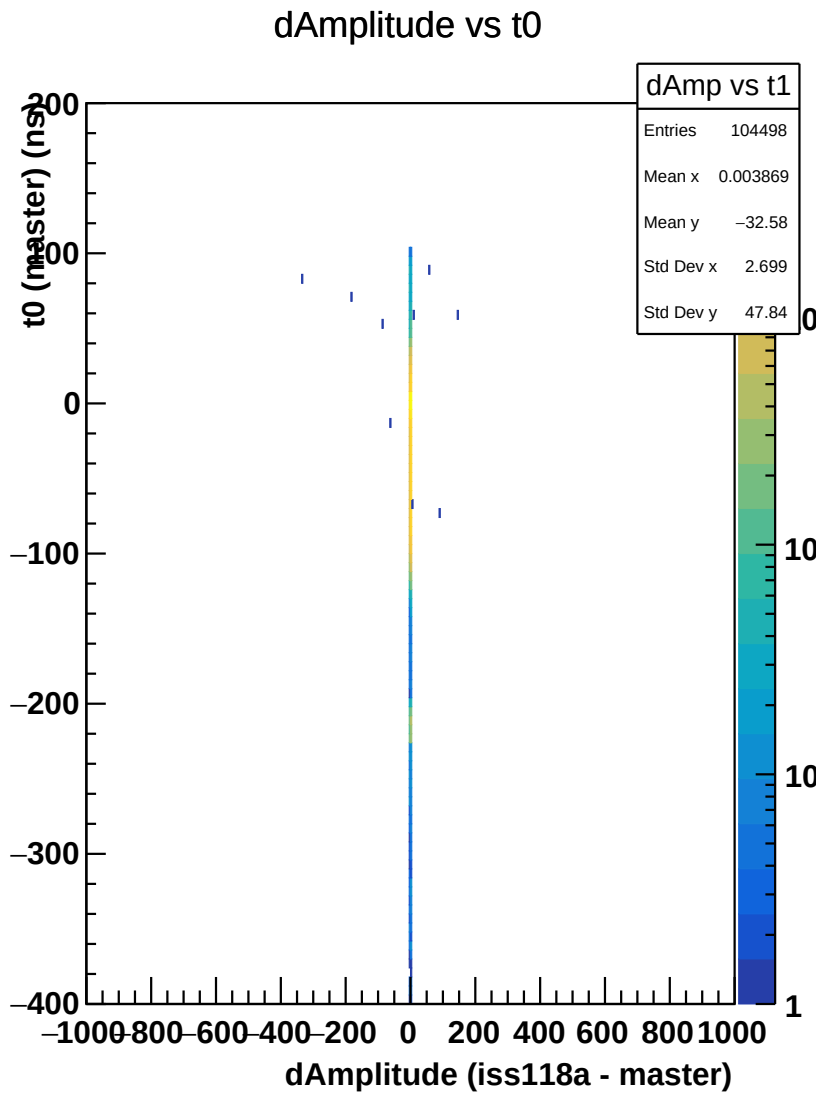
Difference in t0 and amplitudes (all hits)



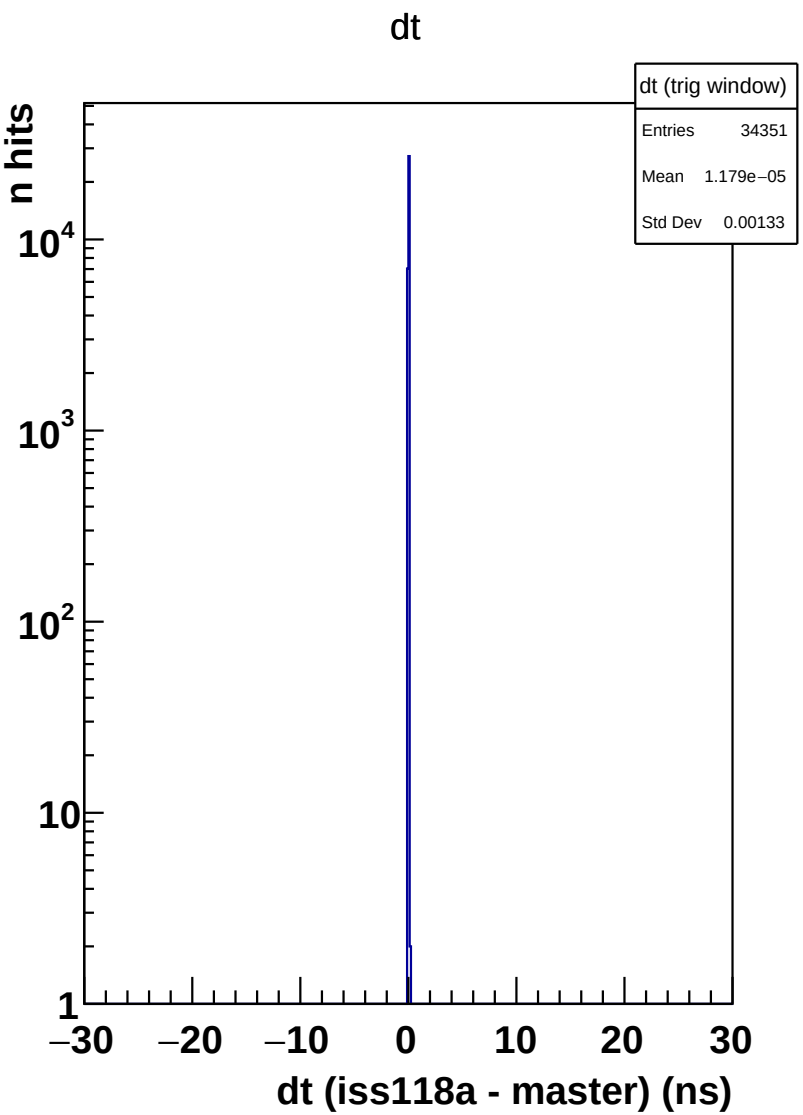


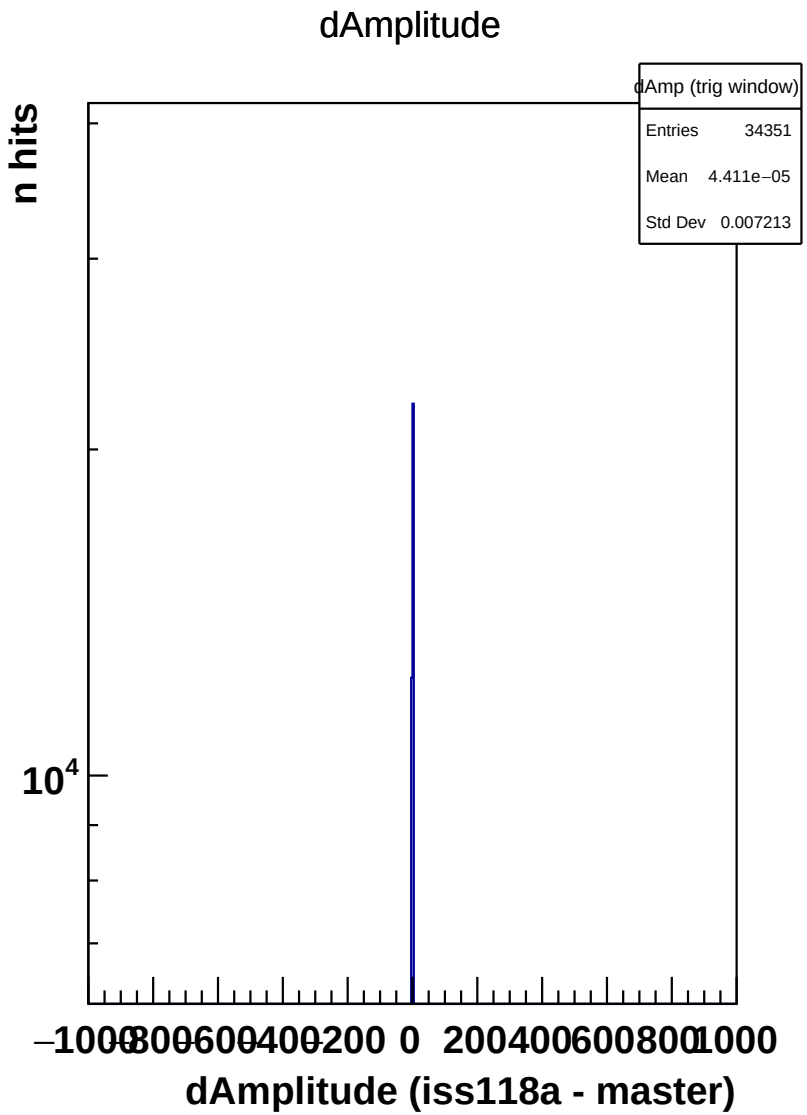
Difference in t0 and amplitudes (vs time)





Difference in t0 and amplitudes (hits in trigger window: $\pm$ 12 ns)

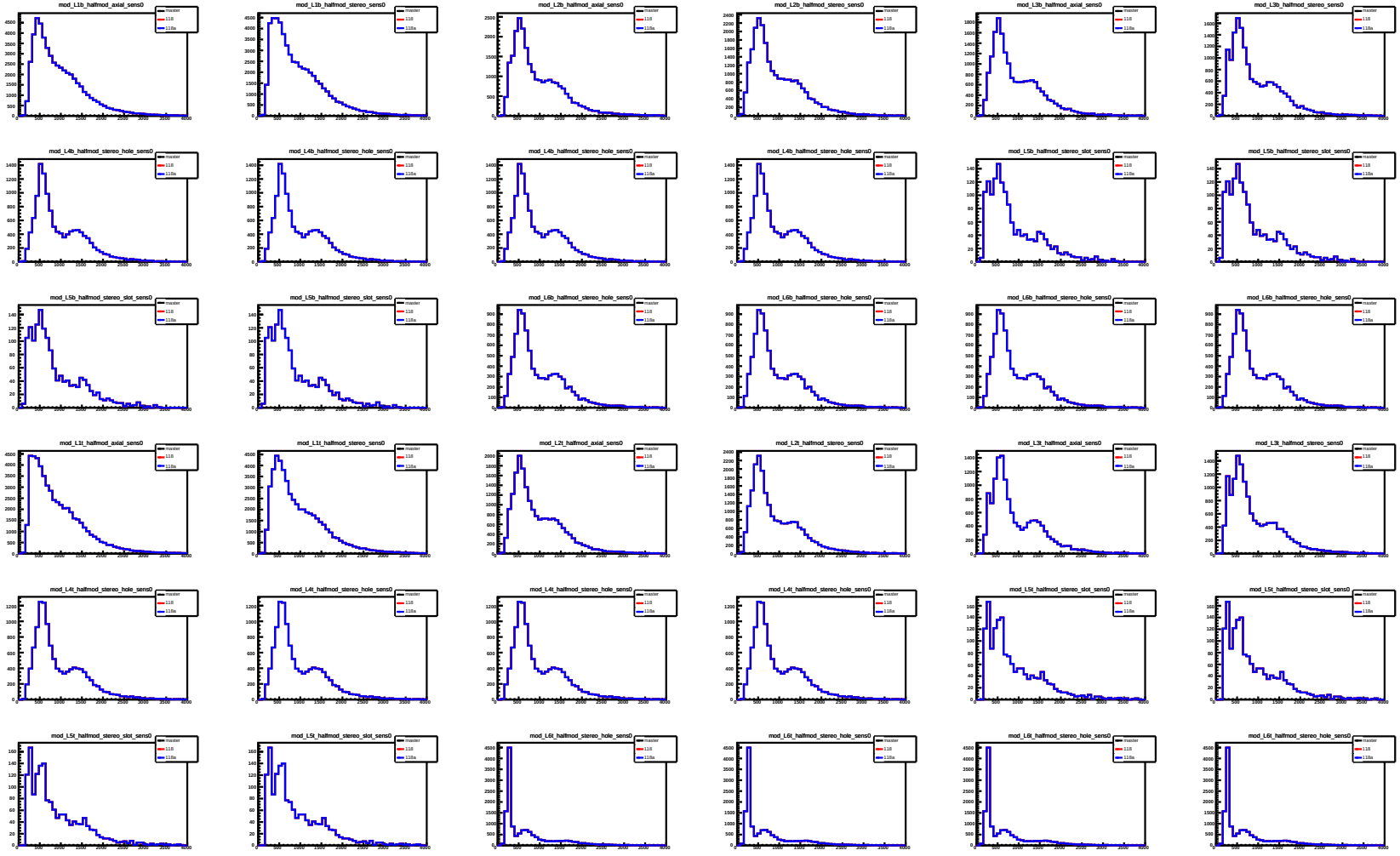




BACKUP SLIDES

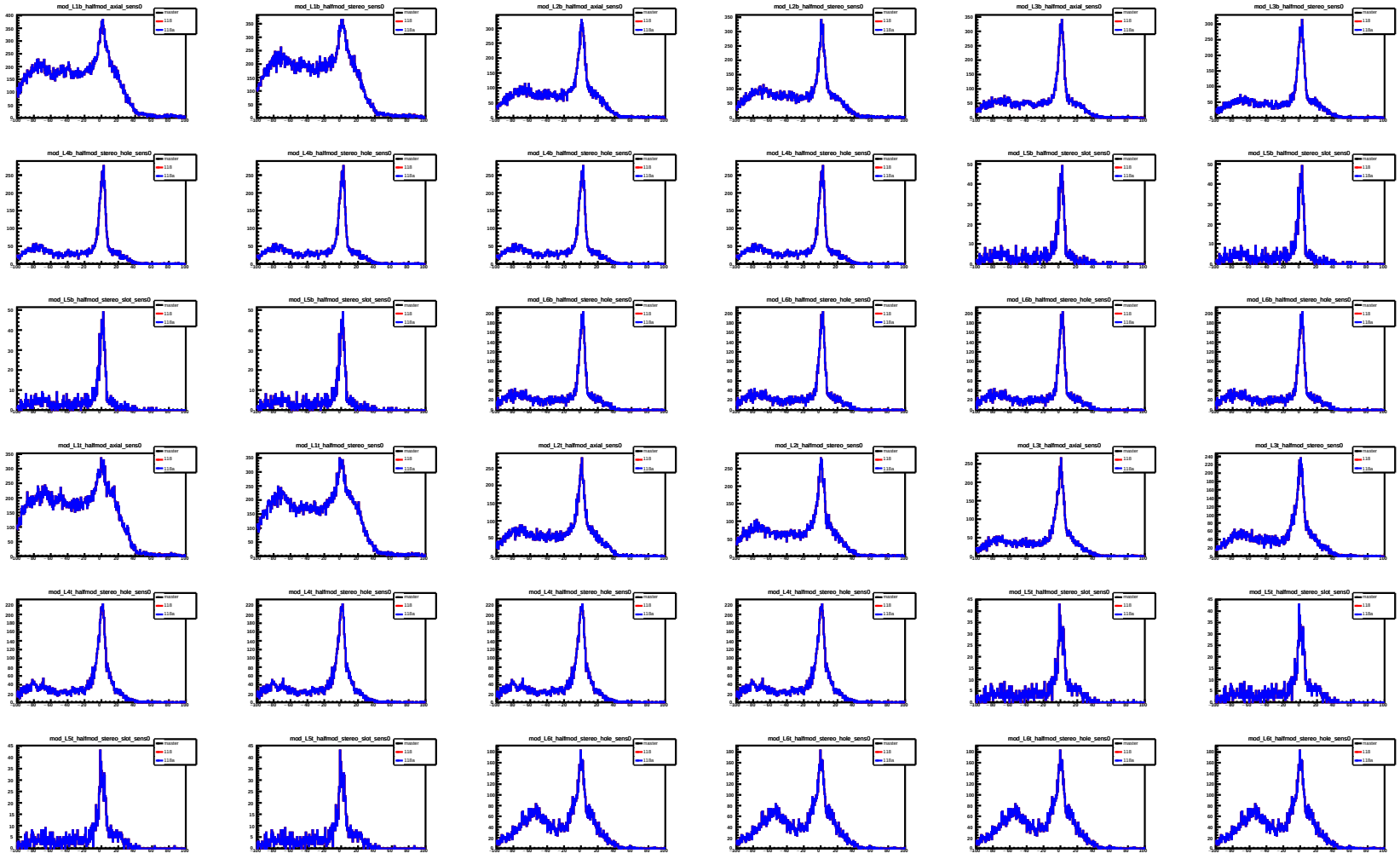
SvtMonitoring/all/amplitudes

master 118 118a



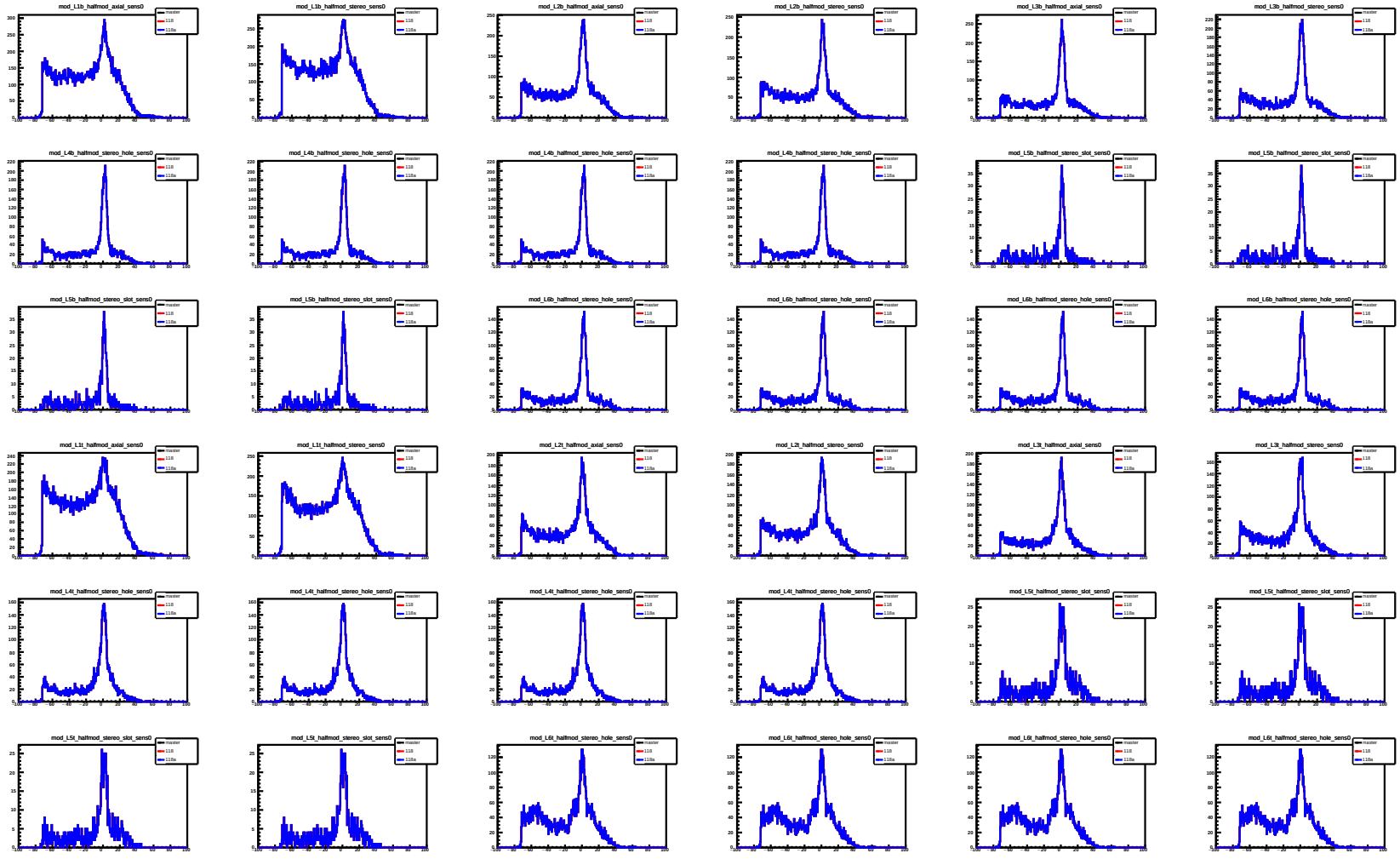
SvtMonitoring/all/t0Hit

master 118 118a



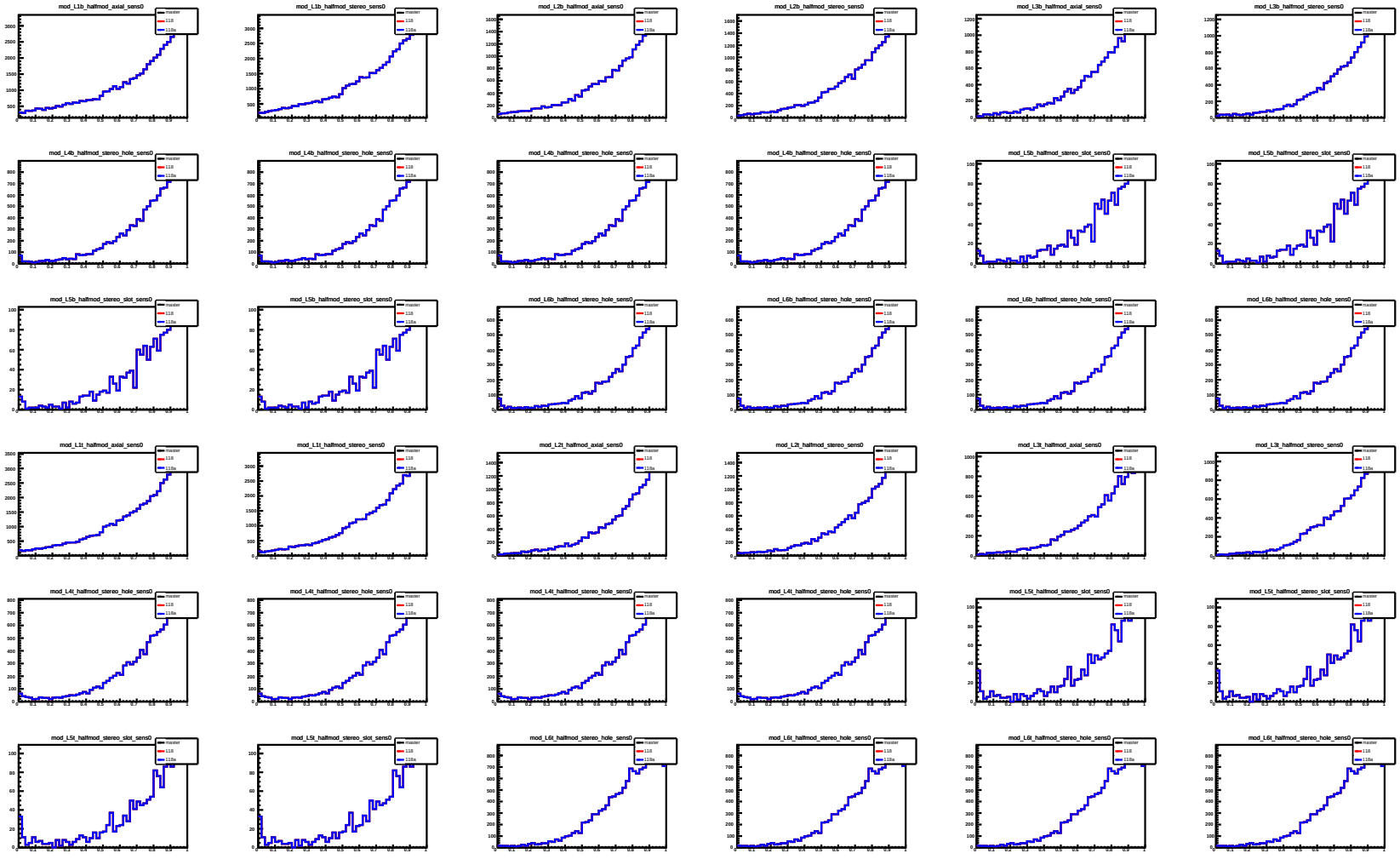
SvtMonitoring/all/t0Cluster

master 118 118a



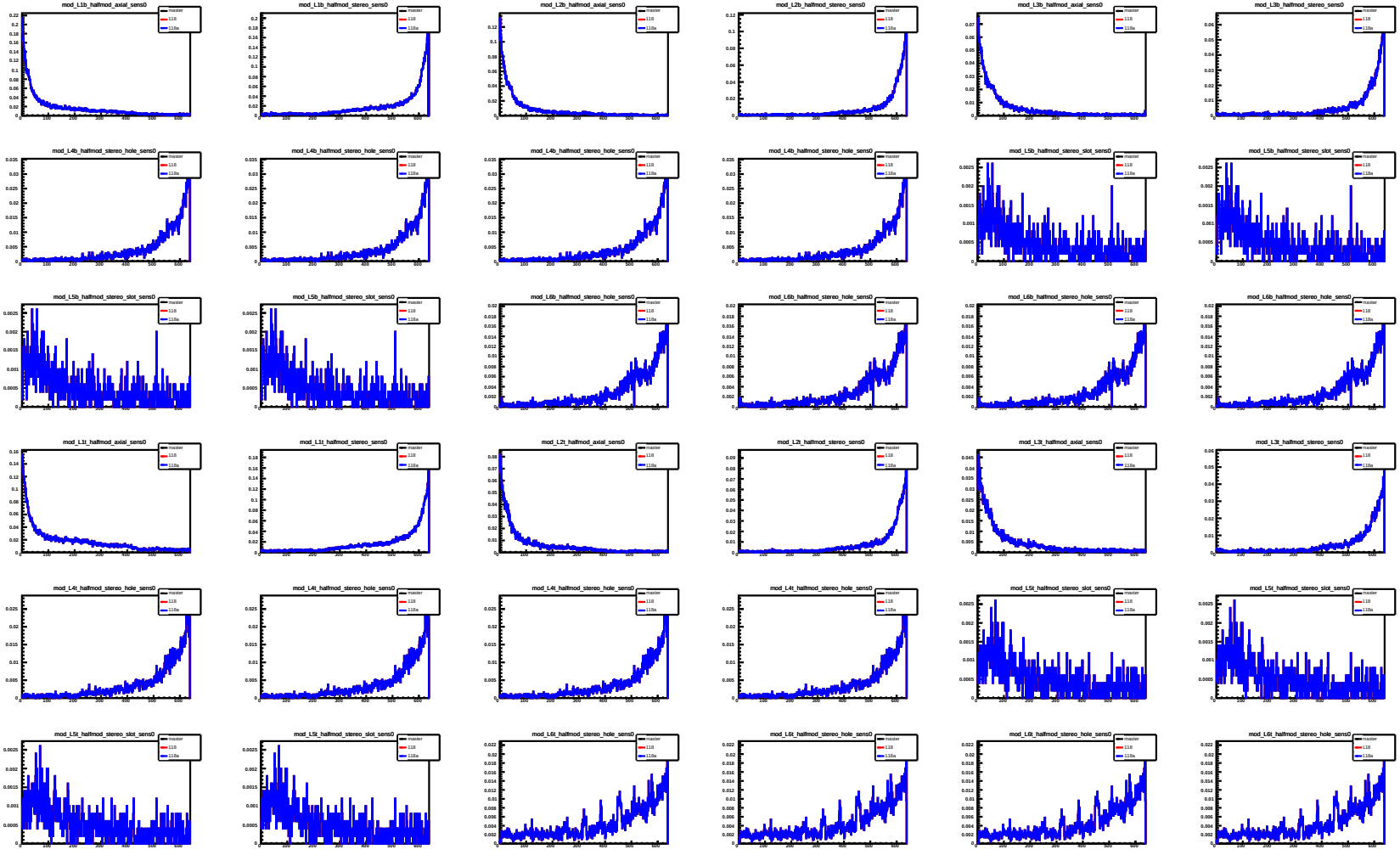
SvtMonitoring/all/chiProb

master 118 118a



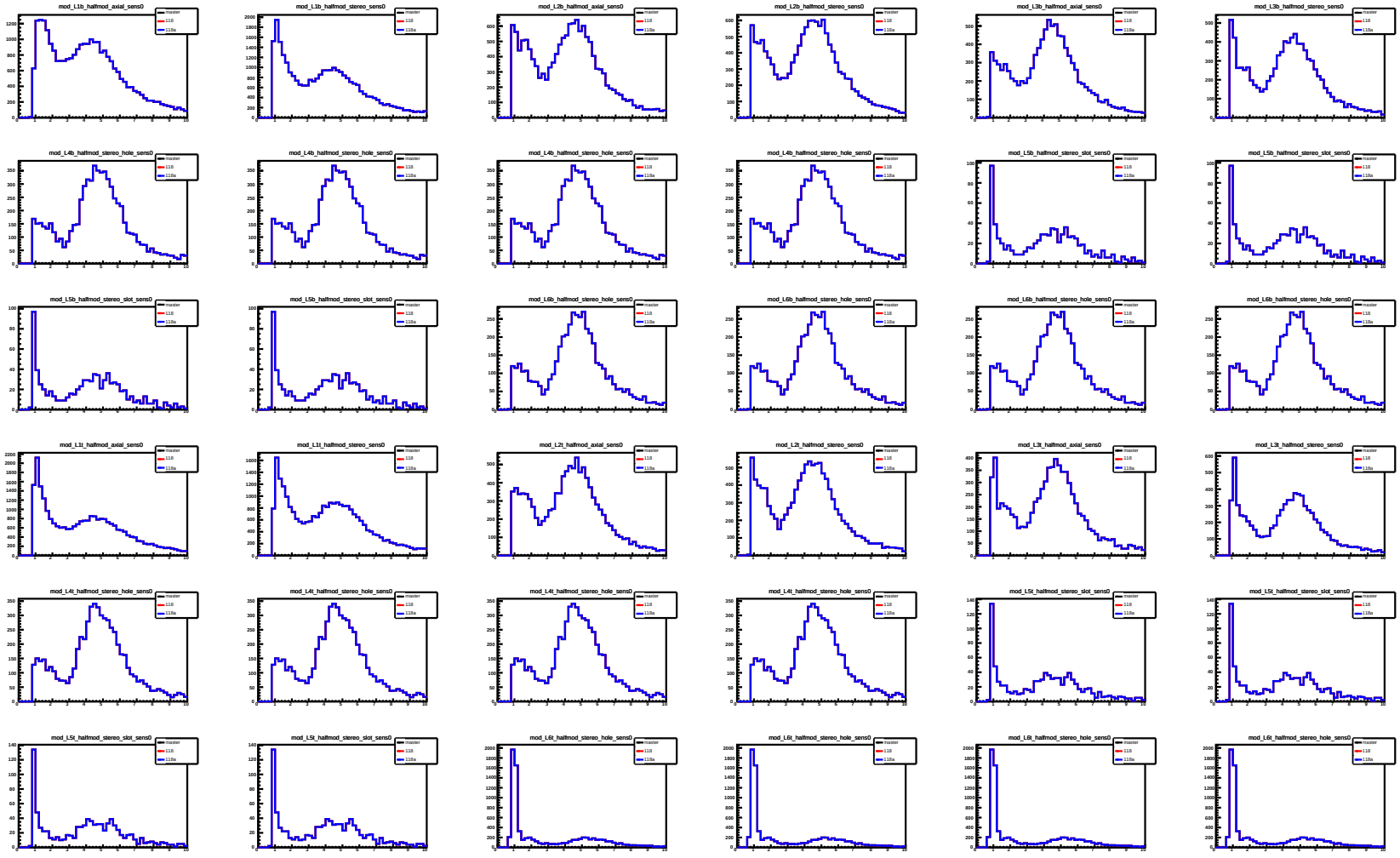
SvtMonitoring/all/occupancy

master 118 118a



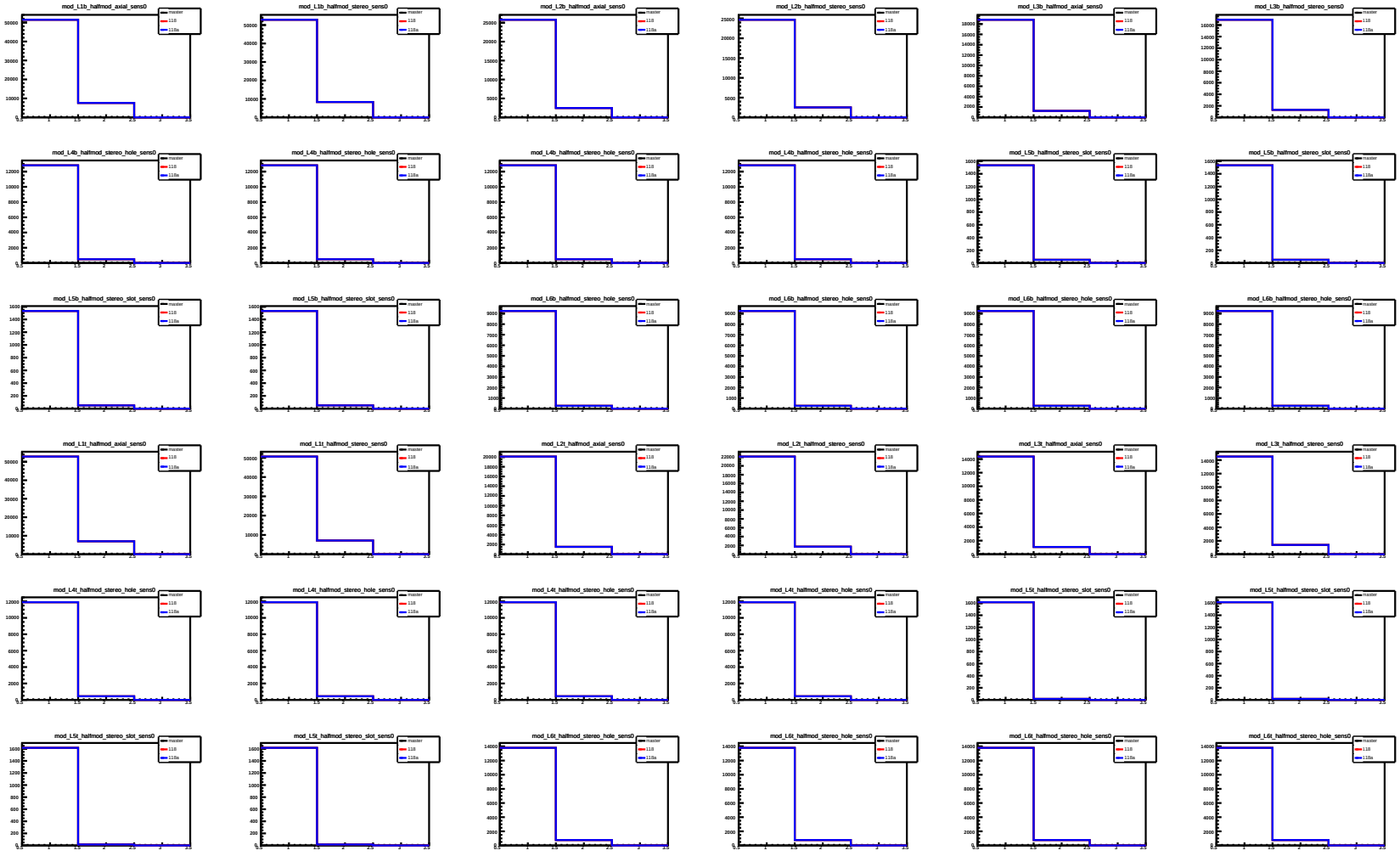
SvtMonitoring/all/electrons

master 118 118a

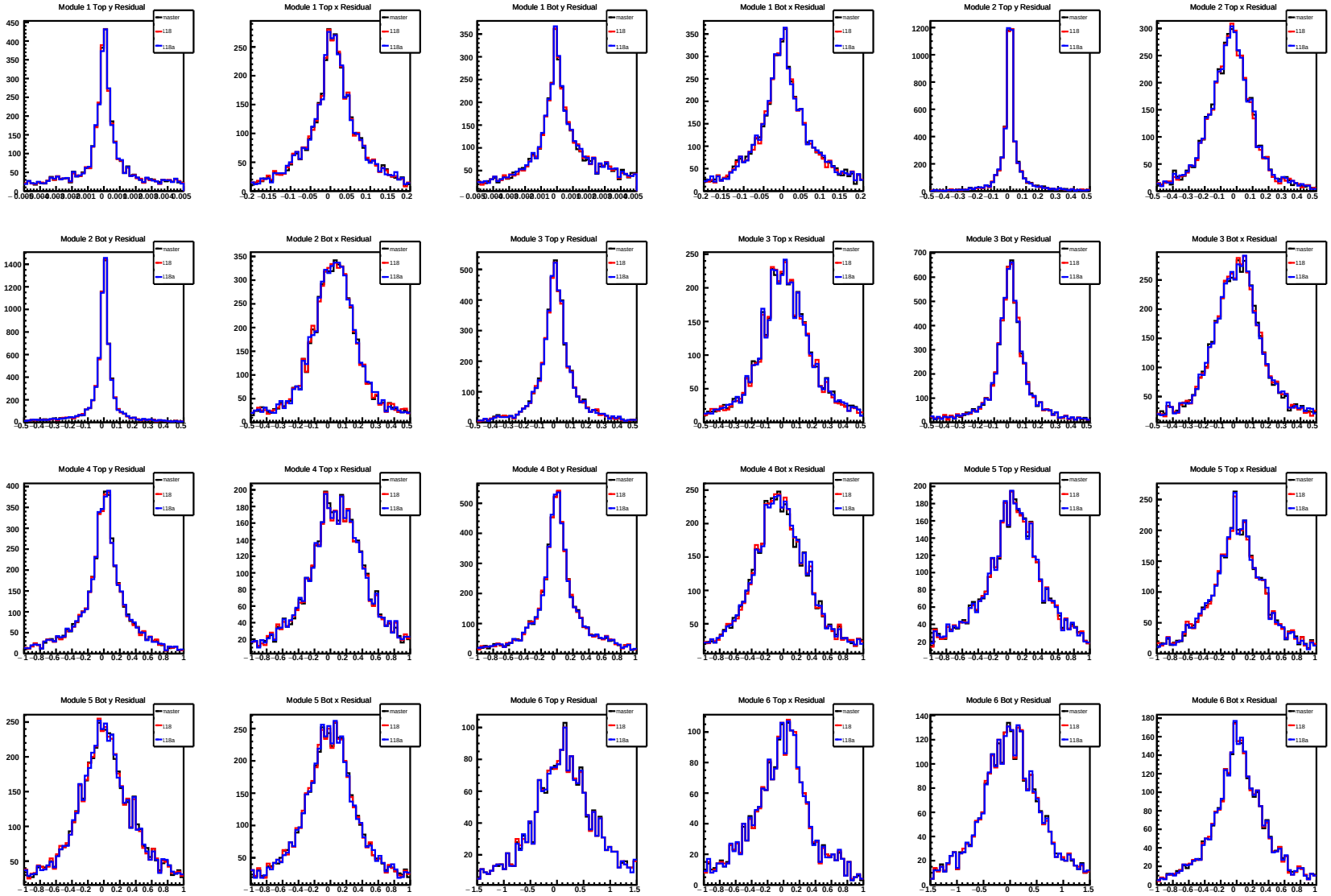


SvtMonitoring/all/nFitsPerHit

master 118 118a



TrackResiduals/all/PostionResiduals/ master 118 118a



Tracks/GBLTracks/all/ master 118 118a