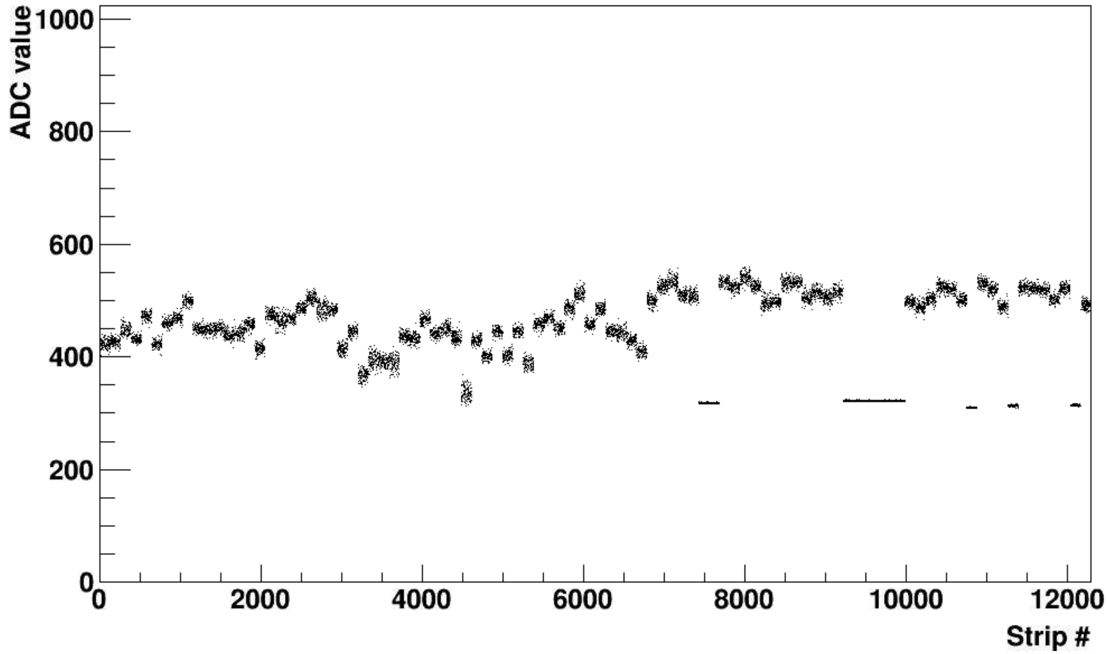
Common Mode Noise subtraction in the FPGAs
November Update I

Jim Thomas
Lawrence Berkeley National Laboratory

16317029 – Raw East P1

- RAW
- Pedestal_TCD_only
- IB = 10,000
  - 100 Hz
- Note that signal amplitude is 10 to 50 counts with a peak near 20

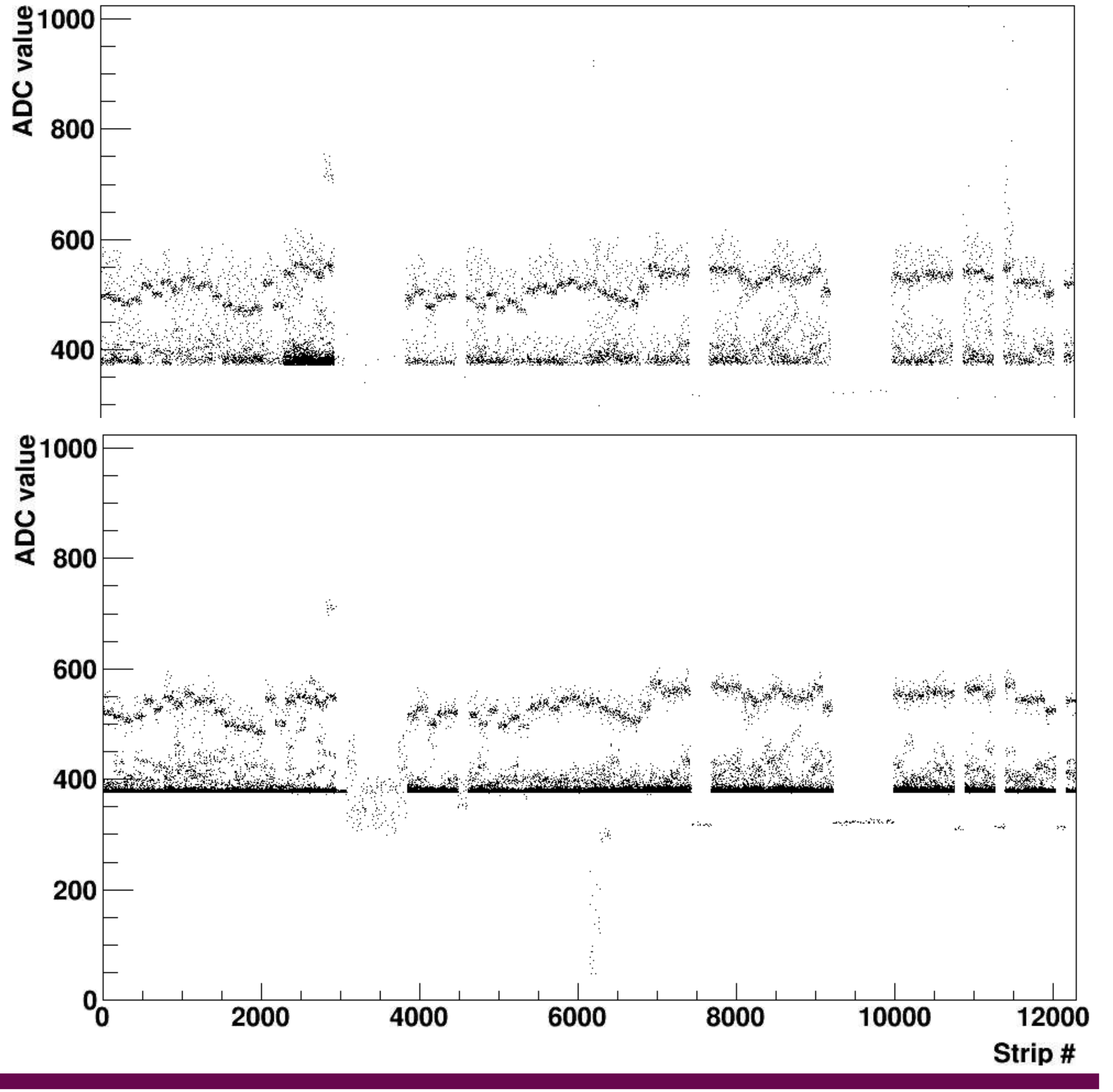


Runs 16317048 and 51 East P1

- Pulser_ZS_2016
- PedAsPhys_TCD
- IB=1500

- Zero Suppressed
  - 100 Hz

- CMN Suppressed
  - 100 Hz



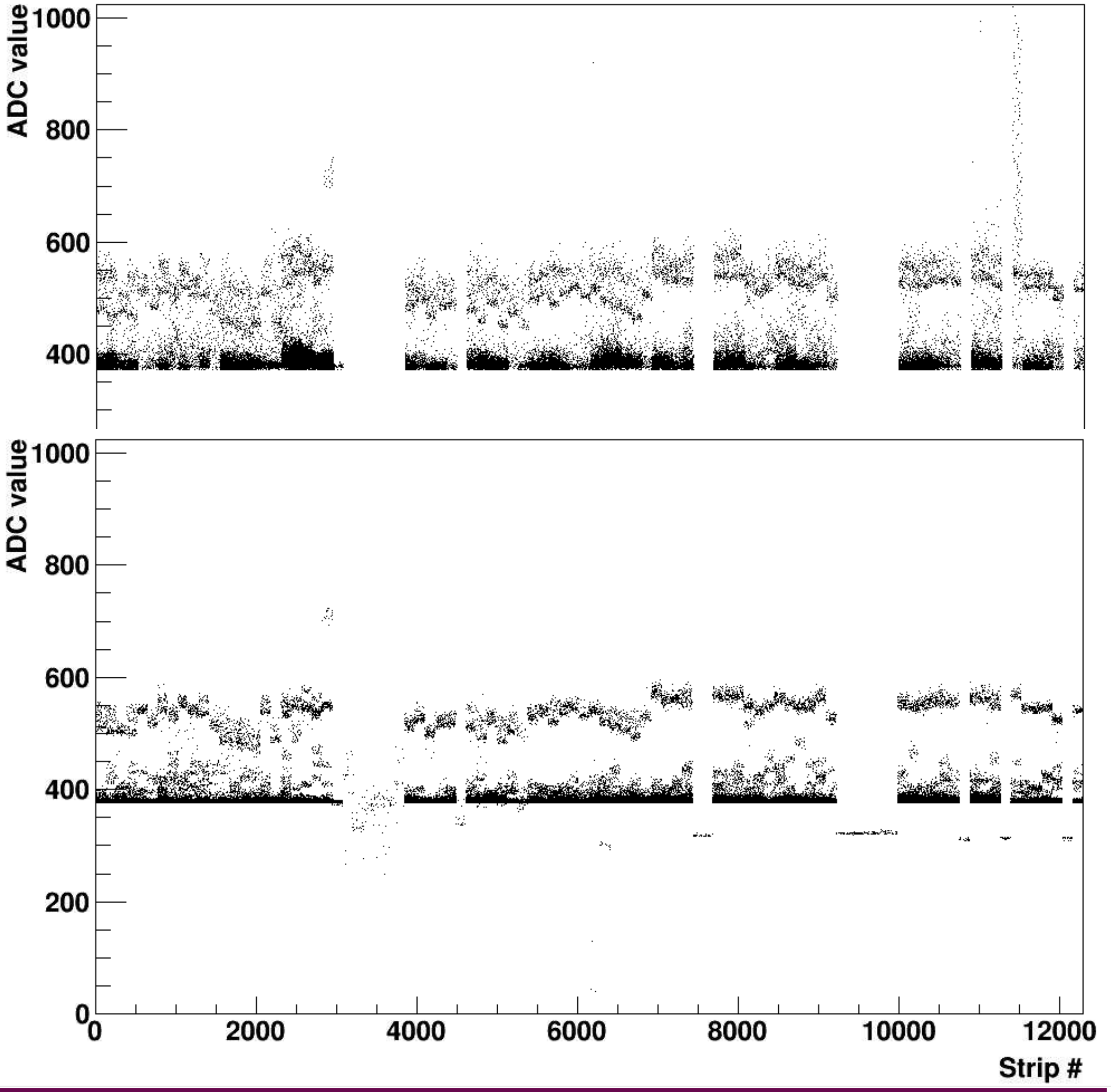
Runs 16317049 and 52   East P1

STAR SSD

- Pulser_ZS_2016
- PedAsPhys_TCD
- IB=1500

- Zero Suppressed
  - 500 Hz

- CMN Suppressed
  - 500 Hz

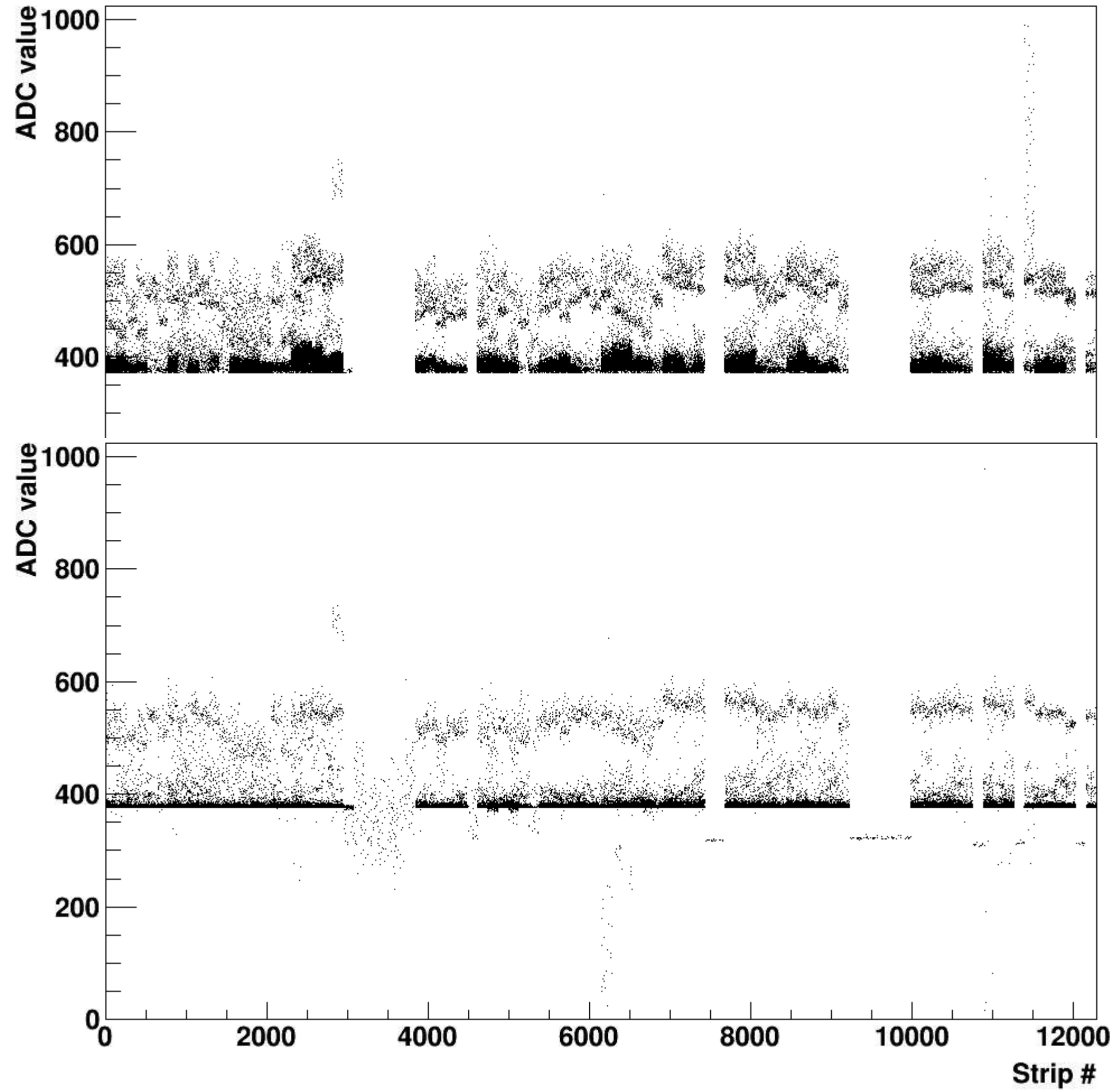


Runs 16317050 and 53   East P1

- Pulser_ZS_2016
- PedAsPhys_TCD
- IB=1500

- Zero Suppressed
  - 1000 Hz

- CMN Suppressed
  - 1000 Hz



Backup slides

Use adjustable limits to hone in on <avg>

STAR SSD

binary

- Select 2 channels using mean = 0 and cut ± 240 counts
  - Accept first 2 channels that are within 0 ± 240, calculate mean
- Select 4 channels using mean = mean(2c) and cut ± 120 counts
  - Accept next 4 channels that are within mean(2c) ± 120, calculate new mean
- Select 8 channels using mean = mean(4c) and cut ± 60 counts
  - Accept next 8 channels that are within mean(4c) ± 60, calculate new mean

linear

- Select 16 channels using mean = mean(8c) and cut ± 30 counts
  - Accept next 16 channels, calculate mean, etc. but now keep data in sum
- Select 16 more channels using mean = mean(16c) and cut ± 20 counts
  - Accept next 16 channels, calculate mean, etc. but now keep data in sum
- Select 32 more channels using mean = mean(32c) and cut ± 10 counts
  - Accept next 32 channels, calculate mean, etc., using most recent 32+16+16 = 64 channels to calculate the final mean value

- 78 channels required (80 if we skip first and last channels)
  - if 78 (80) good channels cannot be found then mark A128 as bad

This can be done in an FPGA friendly way

STAR SSD

```
Float_t StSstDaqMaker::CalculateCommonModeNoise_FPGA( vector<int> vadc )
{

  // Self-tuning algorithm adopted for FPGA

  Int_t Threshold     = 240 ;        // Adjustable threshold for good/bad data.  Start large (240 or 50)
  Int_t StepSize      = 10  ;        // Decrease Threshold in small steps (-10 or -0)
  Int_t ChangeSlope   = 8   ;        // Change from binary limits to linear decrease (8)
  Int_t Divide        = 2   ;        // Decrease Threshold in large steps (/2  or /1)
  Int_t Target        = 2   ;        // Initialize
  Int_t Mean          = 0   ;        // Initialize
  Int_t Sum           = 0   ;        // Initialize
  Int_t Counter       = 0   ;        // Initialize

  for ( int i = 1 ; i < 127 ; i++ )  // Skip first and last channels (for good luck)
    {
      if ( (vadc[i] < (Mean-Threshold)) || (vadc[i] > (Mean+Threshold)) ) continue ;
      Sum += vadc[i]                ;
      Counter++                     ;
      if ( Counter == Target )       // Targets are Powers of 2 (e.g. 2, 4, 8, 16, 32 and 64)
        {
          Mean = Sum/Counter        ;  // Note integer arithmetic may lead to 0.5 variations
          Target *= 2               ;  // Increase Target, always powers of 2
          if ( Counter <= ChangeSlope )
            {
              Threshold /= Divide ;  // Fast binary decrement of the threshold
              Counter = 0         ;  // Reset counter during the fast decrease of threshold
              Sum = 0             ;  // Reset sum to avoid using large threshold data in the
            }                        // next estimate of the mean
          else
            Threshold -= StepSize ;  // Start slow linear decrease of the threshold
        }
      if ( Counter == 64 ) return (float)Mean ;
    }

    return 0 ;
}
```

- All data is taken serially, no looping over old data. All multiplications and divisions by powers of 2.
- 78 channels of data required, otherwise return 0.   Can adjust parameters to make it work as constant threshold algorithm, too.