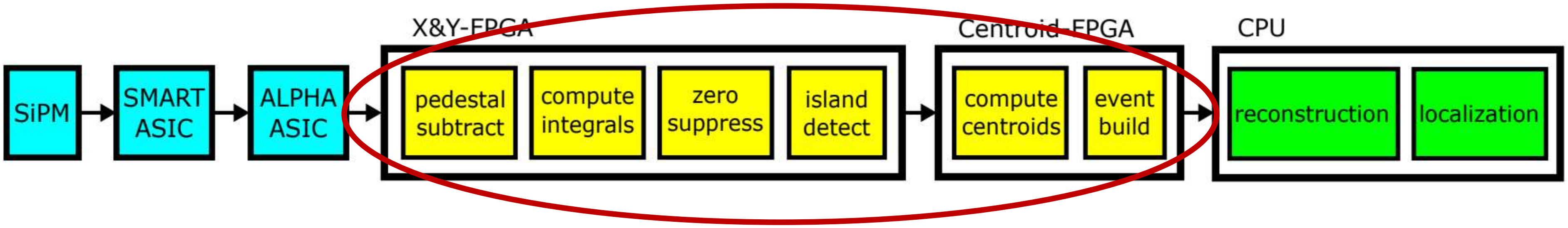
Frontend Computation for ADAPT/APT

Marion Sudvarg, Chenfeng Zhao, Meagan Konst, Thomas Lang, Nick Song, Blake Bal, Longhao Huang, Boran Yang, Kelli Liang, Ruoxi Wang

Roger Chamberlain

On-Instrument Computational Pipeline

ASICs to FPGAs to multicores



- SMART ASIC from INFN – Bari, Italy
- ALPHA ASIC from UH Manoa – Honolulu, HI, US
  - Now most are HDSoC ASICs
- FPGA and multicore functions from WashU – St. Louis, MO, US

Pedestal Subtraction

Pedestal Subtraction

Algorithm is not very complicated – C code is below:

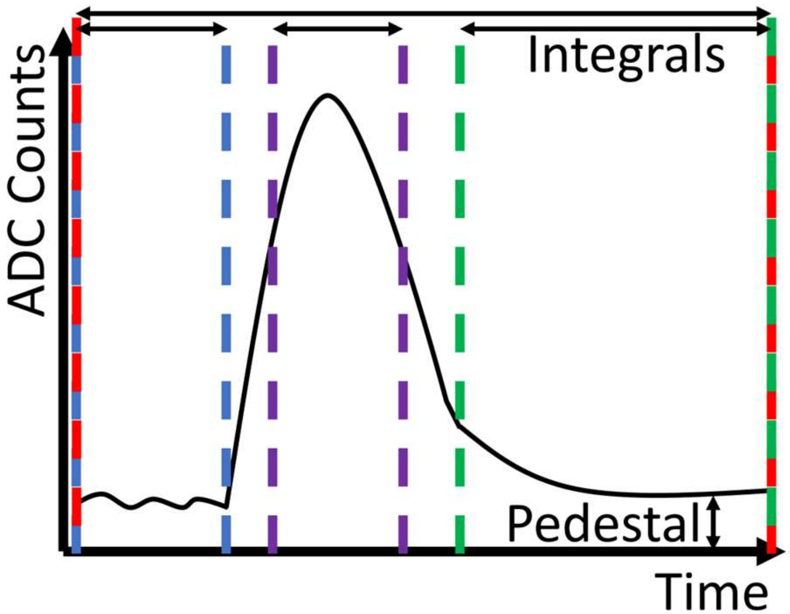
```
void ped_subtract(Packet *pkt, unsigned *peds, unsigned a) {
  for (unsigned s = 0; s < NUM_SAMPLES; ++s) {
    unsigned idx = (pkt->starting_sample + s) % NUM_SAMPLES;
    for (unsigned c = 0; c < NUM_CHANNELS; ++c) {
      unsigned ped_idx = pkt->bank * NUM_SAMPLES * NUM_CHANNELS +
                idx * NUM_CHANNELS + c;
      results[a][s][c] = pkt->samples[s][c] - peds[ped_idx];
} } }
```

- Just some index computations and one subtraction, looping samples and channels
- Can we use High-Level Synthesis (HLS) to implement it , and rest of front-end pipeline?

Compute Integrals – Photon Counting

1st Approach – Integration

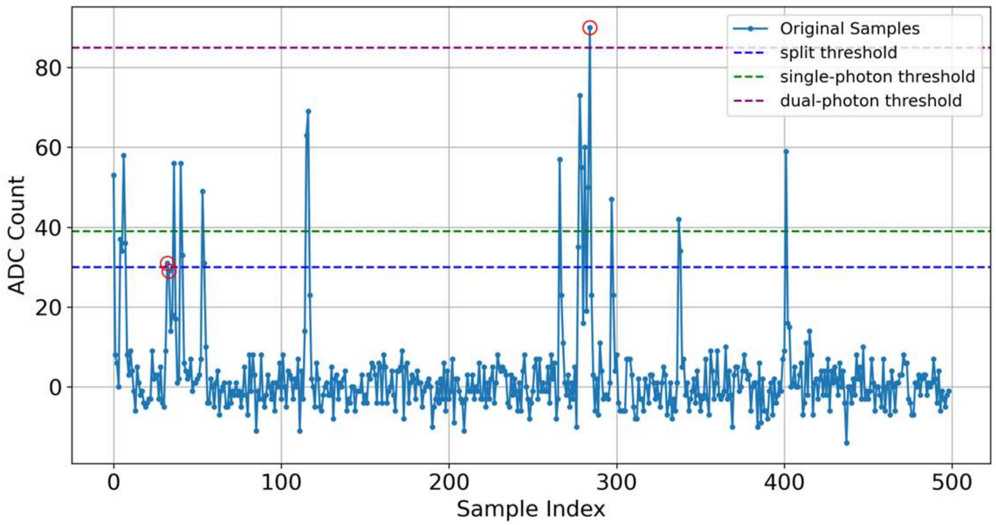
Output from pedestal subtraction



- Integrate portions of waveform
- Two approaches:
  - Unrolled looping sum
  - Parallel prefix sum
- Scale to determine photon count
- In Compton regime – lots of noise relative to actual signal

2^{nd} Approach – Threshold Photon Count

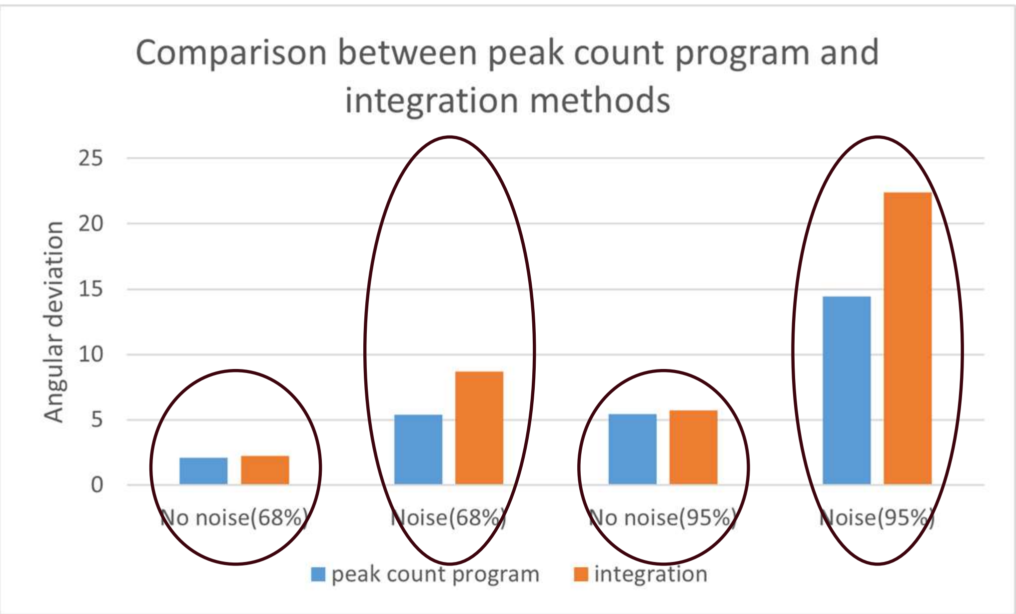
More realistic output from pedestal subtraction (i.e., noise included)



- Count photo electron peaks
- Better noise suppression between peaks
- Thresholds are challenge
  - Multiple needed for different conditions
    - E.g., peak split across two samples
  - Difficult to tune

2^{nd} Approach – Threshold Photon Count

How good are the localization results?



- With no noise, 1st and 2nd approaches are similar in performance
- With noise, threshold photon count has substantially better performance

3rd Approach – Local Filter

(1) Threshold **(2) Filter**

- Threshold test to check if sample is above noise level

- Filter small window around sample above threshold
  - Current filter is a simple box filter
  - Need to seriously investigate additional filters
    - E.g., matched filter to impulse response

- Good photon counts with less sensitivity to threshold values

Zero Suppression

Zero Suppression

Only needed for integral approach

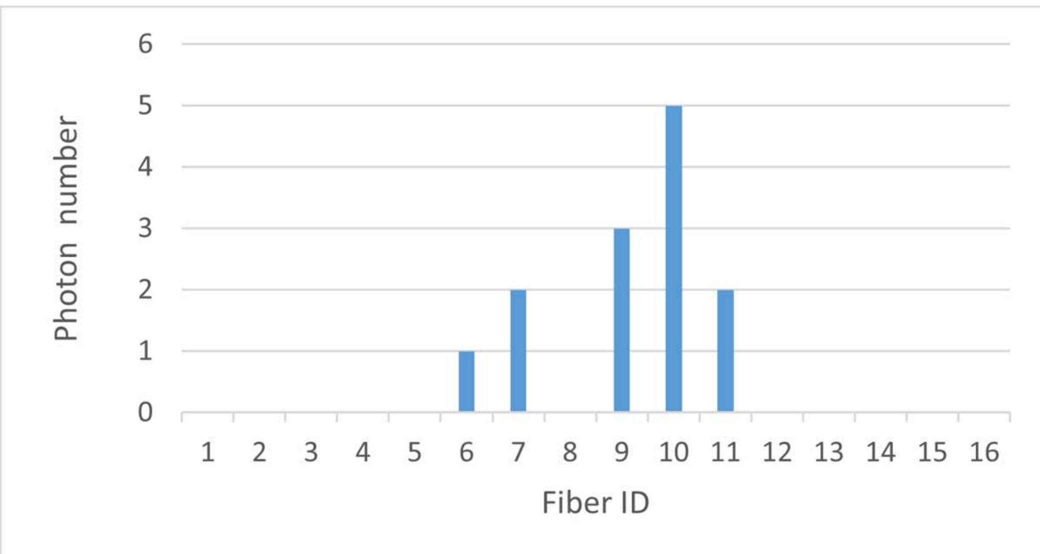
```
void zero_suppress(int32_t * integrals, const int32_t * thresholds) {
  for (unsigned i = 0; i < NUM_INTEGRALS; ++i) {
    for (unsigned c = 0; c < NUM_CHANNELS; ++c) {
      if (integrals[i*NUM_CHANNELS+c] < thresholds[i]) {
        integrals[i*NUM_CHANNELS+c] = 0;
} } }
```

- Just some index computations and one test, looping integrals and channels
- Photon counting approaches already have effectively done zero suppression

Island Detection

Island Detection

Contiguous set of fibers with non-zero photons

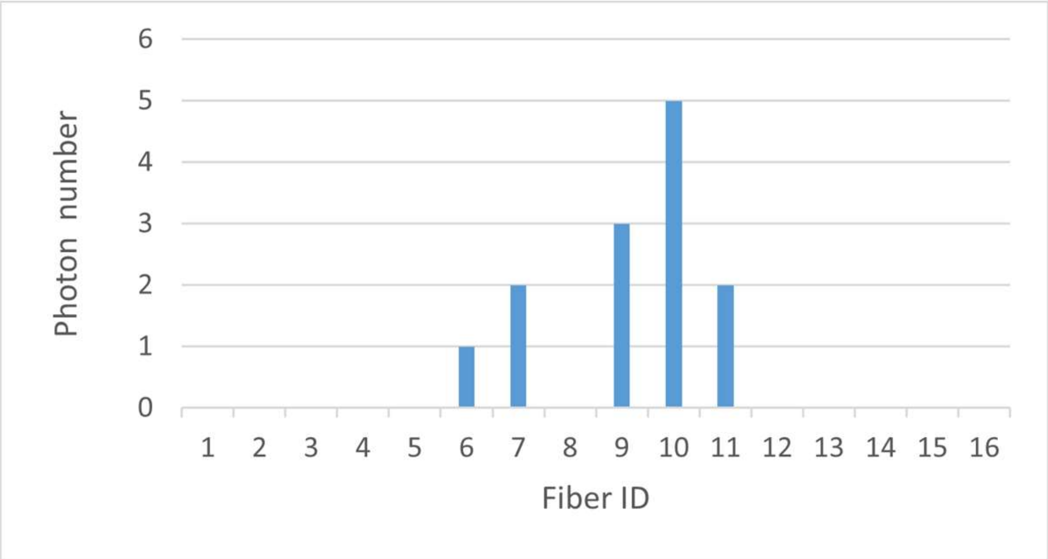


- Initially, graph resulted in 2 islands
- Revised to allow single fiber with zero photons
- Revised approach gives better localization results

Centroiding

Centroiding

Compute centroid of each island



- Dot product of fiber ID and photon count
- Corresponds to 1^{st} moment
- Centroids and width of islands are reported downstream

So, what about HLS?

HLS Results

Initial results aren't very promising

Implementation	Latency	Initiation Interval
Naïve	1 502 829 clocks	1 254 924

- And there is a race condition! ☹

HLS Results

Fix the race condition

Implementation	Latency	Initiation Interval
Naïve	1 502 829 clocks	1 254 924 clocks
Functional	1 399 745 clocks	1 254 840 clocks

- Not much faster, though ☹

HLS Results

Perform several optimizations to HLS code

- Move pedestal values to BRAM
- Insert #pragmas
  - Pipelining
  - Loop unrolling
- Code modifications
  - Copy data to local variables
  - Swap loop order

HLS Results

We're finally getting somewhere

Implementation	Latency	Initiation Interval
Naïve	1 502 829 clocks	1 254 924 clocks
Functional	1 399 745 clocks	1 254 840 clocks
HLS Optimizations	40 056 clocks	22 412 clocks

- Over 50 times faster! ☺

HLS Results

Can we do better?

- Dataflow pipeline
  - Move data from stage to stage in FIFO queues
- Vectorization
  - Operate on vector of 16 channels simultaneously

HLS Results

This is our best performance

Implementation	Latency	Initiation Interval
Naïve	1 502 829 clocks	1 254 924 clocks
Functional	1 399 745 clocks	1 254 840 clocks
HLS Optimizations	40 056 clocks	22 412 clocks
Dataflow & Vectors	440 clocks	299 clocks

- Another 75 times faster! ☺

HLS Results

Are there better tools?

- Previous result is from vendors HLS compiler
- There are 3 academic HLS compilers
  - HIDA-ScaleHLS from UIUC
  - Bambu from Politecnico di Milano
  - Dynamatic from EPFL
- We have students attempting to use the first two

Conclusions and Future Work

- HLS gives good implementations
- Requires non-trivial optimizations
- Plenty fast enough
- Doesn't overly consume FPGA resources

- Does it all fit on FPGA along with configurations?
- Can academic HLS compilers simplify design effort?
- Algorithm tuning underway
- Also adding 2nd and 3rd moments for pair-production events

Questions?

URL: adapt.physics.wustl.edu

(Credit: Mark Garlick/ U. Warwick)

My contact info:

roger@wustl.edu