Implementation of waveform compression for ICECUBE

Klaus Helbing

KHelbing@lbl.gov

LBNL

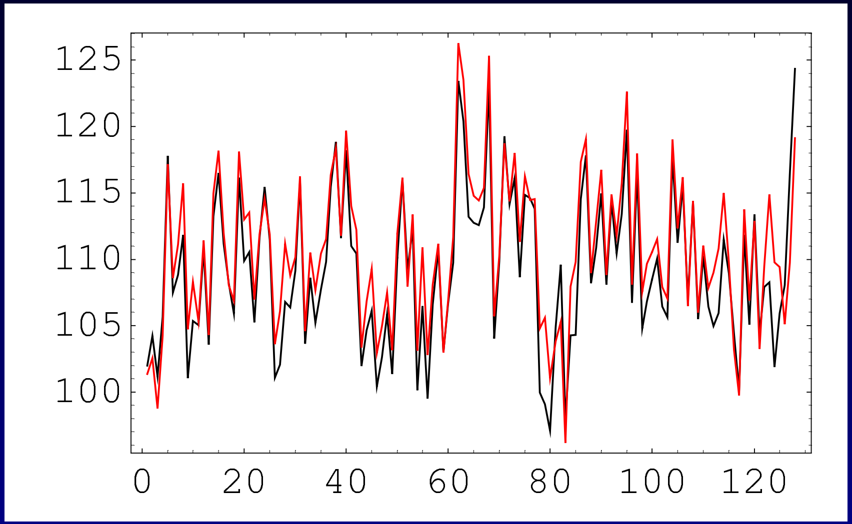
Constraints on compression

- Bandwidth: 1 Mbaud per 2 DOMs $\longrightarrow$ Karl-Heinz Sulanke
- Computing resources:
  - CPU: 40 MHz ARM (16.8 MHz in String-18)
  - FPGA: $\sim$ 16,000 Logic elements ($\sim$ 3000 in String-18, no room left for compression)
- Noise rate: 500 Hz (required; 1kHz in String-18)
- Correlation of noise:
  $\Rightarrow$ feature extract SPEs and send MPEs uncompressed doesn't work.

Principle idea

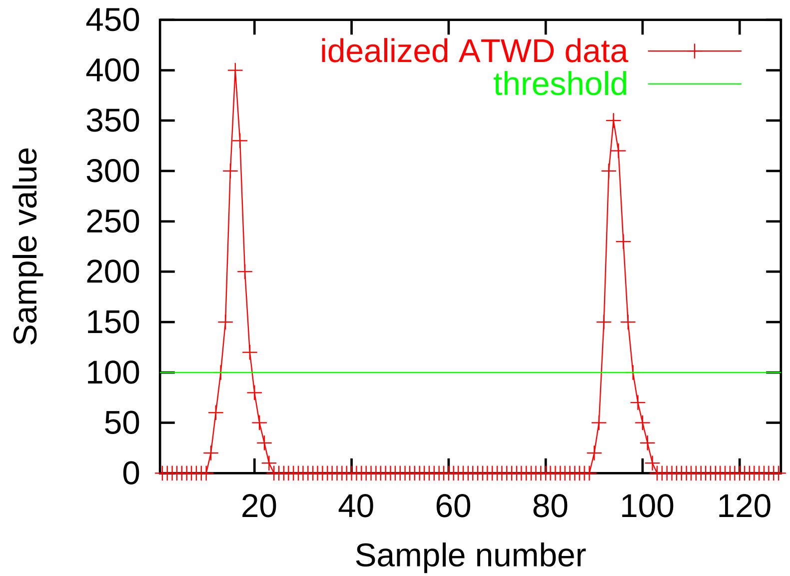
- Waveform of 1 ATWD channel or fADC similar to a facsimile scan line.
- Waveform corresponds to gray-scale representation of FAX line.
- Original FAX encoding: run-length encoding followed by Huffman encoding.
- Group 3 CCITT facsimile standard: fixed, immutable, Huffman code, optimized for a set of eight standard documents.
- Do Huffman encoding of waveforms with static table (to be generated as a calibration task).
- Further simplify this to “Huffman-lite”

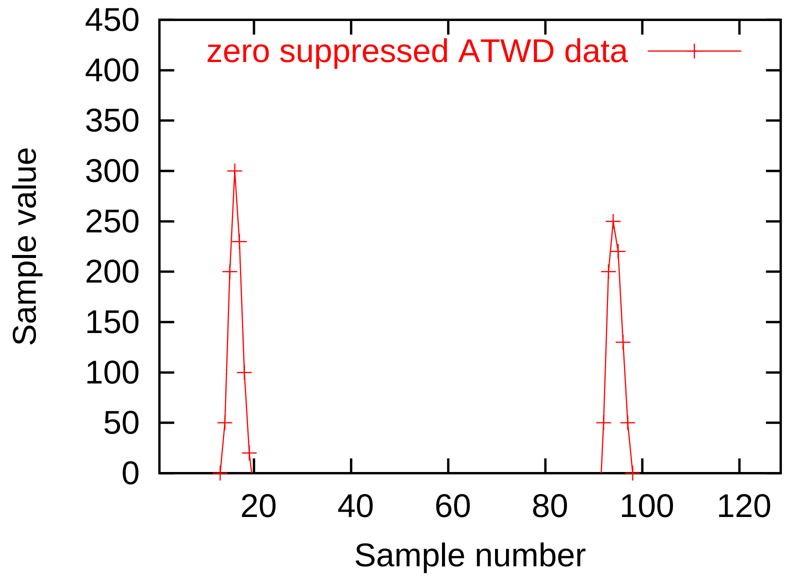
Step I: Pedestal etc

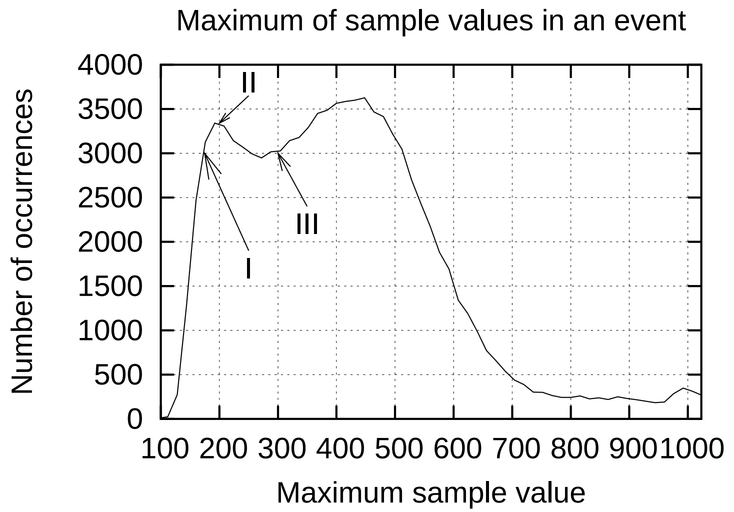


- Pedestal investigated by David Seckel: not perfectly constant in String-18, integrated IceCube DOM should deliver constant fingerprint (ATWD tester, STF, TestDAQ)
- Ringing should not occur in IceCube DOMs!
- Baseline: Low frequency noise on different ground levels.
- Pick ATWD channel with highest gain that doesn’t saturate.

Step II: Zero suppression

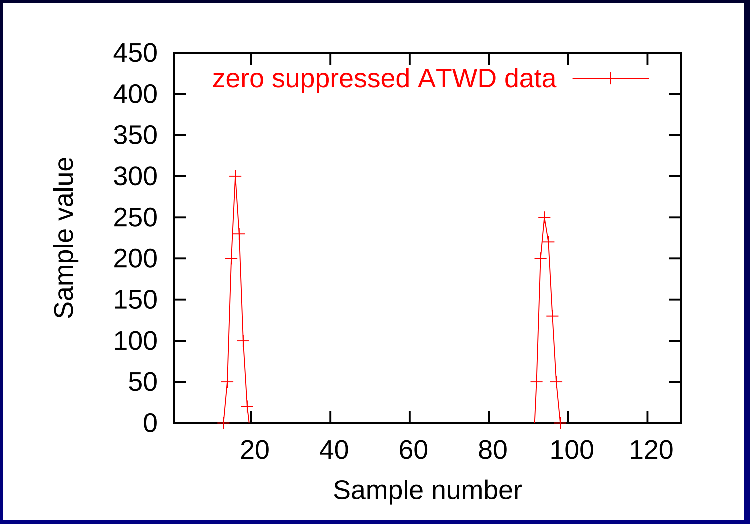






Typical setting of threshold of discriminator and zero suppression: 25% - 33% of average SPE pulse hight over baseline.

Step III: Run-length encoding



0 . . . 0 50 200 300 230 100 20 0 . . . 0 50 200 250 220 130 50 0 . . . 0: **128 Bytes**

$\Downarrow \quad \Downarrow \quad \Downarrow \quad \Downarrow$

12 0, 0 50, 0 200, 0 300, 0 230, 0 100, 0 20, 71 0, 0 50, 0 200, 0 250, 0 220, 0 130, 0 50, 30 0: **30 Bytes**

Meaning of pair here: Number of consecutive *repetitions*, Value.

Standard RL: Number of consecutive *occurencies*, Value.

tweak of standard $\Rightarrow \sim$ 50% of numbers are zeros (less entropy).

Step IV: Huffman-lite

Huffman encoding:

- Minimal variable-length “character” encoding based on the frequency of each “character”.
- more frequent “characters” are encoded with few bits, and rare “characters” are encoded with many bits.

Huffman-lite:

- $\sim$ 50% of numbers (“characters”) are zeros.
- Minimize bits used for zeros, don’t work on finite values.
- Convert “0” (8 bits) $\rightarrow$ ’0’ (1 bit)
  Convert “N” (8 bits) $\rightarrow$ ’1’,“N” (9 bits)

$$12\ 0,\ 0\ 50,\ \ldots \rightarrow \text{'}100001100010011001 0\ \ldots\text{'}$$

Compression efficiency

[Bytes/event]	STRING-18			ICECUBE (estimate)		
Method	ATWD	fADC	header	ATWD	fADC	header
RAW	128	128	6	128	128	6
run-length	19	6	6	11 [a]+3 [b]	6	6
"gzip –fast"	15		6			6
"bzip2 -9"	10		6			6
"Huffman-lite"	15		6	12		6
Total: "run-length + Huffman lite"	**21**			**18**		

- **STRING-18:** 1 kHz PMT noise, 1 DOM/cable → 21 kB/s
- **ICECUBE (estimate):** 0.5 kHz, 2 DOM/cable → **18 kB/s**
- **Available bandwidth (IceCube): 100 kB/s/cable**

[a] 11: Half the sampling speed of ATWD.

[b] 3: Estimate for longer ATWD time window in ICECUBE.

Computing requirements

- Intel Pentium-4:
  $\sim$ 10,000 CPU clock cycles per event.
  - Pedestal subtraction, zero supp. (Step I&II): $\sim$ 50%.
  - Run-lenght encoding (Step III): $\sim$ 30%.
  - Huffman encoding (Step IV): $\sim$ 20%.
- ARM on IceCube MB:
  $\sim$ **20,000** CPU clock cycles per event.
- GTP & TS confident: Whole algorithm fits in FPGA.
- Certainly, biggest chunk for CPU (Step I) fits.

IceCube MB CPU: 40 MHz (can be increased); 500 Hz noise

$\Rightarrow$ **CPU load due to compression $<$ 15%**.

Induced losses

- Step I (pedestal pattern etc): lossless.
- Step II (zero suppression): lossy!
  - Concept in particle physics for half a century.
  - Obvious way to distinguish electronic noise from PMT.
  - Equivalent to discriminator effect if threshold is set the same "value": we already accepted to lose these pulses below threshold.
- Step III (Run-length): lossless.
- Step IV (Huffman-lite): lossless.

$\Rightarrow$ *I don't understand the excitement about the losses!*

Room for improvements

- Full Huffman encoding ($\sim$ factor 1.5)
- Reduce resolution of sample value ($\longrightarrow$ charge):
  PDD: 5%; 10 bits: 0.1%, 8 bits: 0.4%, 4 bits: 6%
- Reduce time resolution:
  PDD: 5 ns; ATWD sampling speed: 3.3 ns; going to 6.7 ns and fitting the pulse shape
  $\longrightarrow$ still meet the requirement.
- Going to assembler (instead of C)
  $\longrightarrow$ reduce CPU cycles.

Conclusion

- Algorithm is simple.
- Bandwidth: Factor of 5 safety margin!
- Several parameters left to improve further.
- Computing resources (CPU, FPGA) sufficient.
- Induced losses: Already accepted!
- Some implementation details need to be sorted out (header alignment, where to decompress at surface . . .)

$\Downarrow$

• **Compression is on sound basis**

• My opinion: “Local coincidence” obsolete if glass/noise and cable meet requirements.