SANDIA REPORT
SAND2019-12876
Printed October 2019

SPARR: Spiking/Processing Array for Wide Dynamic Range and High Resolution Photonic Sensing

Park E. Hays, Matthew J. Kagie, David B. Karelitz, Randolph "Rex" Kay, John Steven Mincey, Mark Woods

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185
Livermore, California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

NOTICE: This report was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government, nor any agency thereof, nor any of their employees, nor any of their contractors, subcontractors, or their employees, make any warranty, express or implied, or assume any legal liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represent that its use would not infringe privately owned rights. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government, any agency thereof, or any of their contractors or subcontractors. The views and opinions expressed herein do not necessarily state or reflect those of the United States Government, any agency thereof, or any of their contractors.

Printed in the United States of America. This report has been reproduced directly from the best available copy.

Available to DOE and DOE contractors from

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods

ABSTRACT

The Spiking/Processing Array (SPARR) is a novel photonic focal plane that uses pixels which generate electronic spikes autonomously and without a clock. These spikes feed into a network of digital asynchronous processing elements or DAPES. By building a useful assemblage of DAPES, and connecting them together in the correct way, sophisticated signal processing can be accomplished within the focal plane.

Autonomous self-resetting pixels (ASP) enable SPARR to generate electronic response with very small signals–as little as a single photon in the case of Geiger mode avalanche photodiodes to as few as several hundred photons for in-CMOS photodetectors. These spiking pixels enable fast detector response, but do not draw as much continuous power as synchronous clocked designs. The spikes emitted by the pixels all have the same magnitude, the information from the scene is effectively encoded into the rate of spikes and the time at which the spike is emitted.

The spiking pixels, having converted incident light into electronic spikes, supply the spikes to a network of digital asynchronous processors. These are small state machines which respond to the spikes arriving at their input ports by either remaining unchanged or updating their internal state and possibly emitting a spike on one or more output ports.

We show a design that accomplishes the sophisticated signal processing of a Haar spatial wavelet transform with spatial-spectral whitening. We furthermore show how this design results in a data streams which support imaging and transient optical source detection. Two simulators support this analysis: SPICE and `sparrow`. The CMOS SPICE simulator Cadence provides accurate CMOS design with accounting for effects of circuit parasitics throughout layout, accurate timing, and accurate energy consumption estimates. To more rapidly assess larger networks with more pixels, `sparrow` is a custom discrete event simulator that supports the non-homogeneous Poisson processes that lie behind photoelectric interaction. `Sparrow` is a photon-exact simulator that nevertheless performs SPARR system simulator for large-scale systems.

ACKNOWLEDGMENT

We would like to thank the Sandia National Laboratories Lab Directed Research and Development (LDRD) office for sponsoring this research. In addition several individuals were vital to this research's success. Stephen M. Gentry advocated for the research project, asked insightful questions which helped the research find relevance. Furthermore, Philip L. Dreike provided invaluable guidance in communicating this research to audiences with diverse technical backgrounds.

CONTENTS

1. Introduction **13**
1.1. How SPARR Works . . . 18
1.2. Advantages Over Other Focal Plane Array Designs . . . 24
1.3. Potential Applications . . . 25
1.4. Sparrow Introduction . . . 26
1.4.1. Overview . . . 26
1.4.2. Modeling a Pixel . . . 26
1.4.3. Modeling the Haar Transform . . . 27
1.4.4. Modeling the Quadtree Readout . . . 27
1.5. How to Read this Report . . . 27

2. Statistics for a Nominal Earth Scene **29**
2.1. Introduction . . . 29
2.2. Pixel Spikes . . . 30
2.2.1. Photon Arrivals . . . 30
2.2.2. Pixel Spike Probability . . . 31
2.2.3. Pixel Spike Count Simulation . . . 31
2.3. Haar Spikes . . . 32
2.3.1. The Haar Transform in a Static Domain . . . 33
2.3.2. Throttling for a Typical Earth Scene . . . 34
2.3.3. The Haar Transform in a Dynamic Domain . . . 38
2.3.4. Haar Net Spike Probability . . . 41
2.3.5. Haar Net Spike Count Simulation . . . 45
2.3.6. Positive and Negative Haar Spike Means . . . 45
2.3.7. Haar Positive and Negative Spike Mean Simulation . . . 49
2.4. Quadtree Spikes . . . 50
2.5. Conclusion . . . 51

3. Circuit Design **53**
3.1. Circuits Introduction . . . 53
3.2. Asynchronous Self-Resetting Pixel . . . 53
3.2.1. How the Pixel Works . . . 53
3.2.2. Monolithic Front-Side Illuminated FPA . . . 55
3.2.3. Monolithic Back-Side Illuminated FPA . . . 56
3.2.4. Hybrid FPA Configuration 1 . . . 57
3.2.5. Hybrid FPA Configuration 2 . . . 57
3.2.6. Example ASP Circuit . . . 58

3.2.7. Fixed Pattern Noise and Manufacturing Variability 60
3.3. Haar Node Implementation 62
3.3.1. Approximation Coefficient Implementation 62
3.3.2. Horizontal and Vertical Coefficient Implementation 64
3.3.3. Diagonal Coefficient Implementation 67
3.4. Haar Node Performance 68
3.4.1. Haar Node Simulations in Normal Operation 69
3.4.2. Haar Node Simulations in Saturation 72
3.4.3. Haar Node Energy Consumption 72
3.5. Haar Node Alternative Implementation 74
3.5.1. Alternative Approximation Coefficient Implementation 75
3.5.2. Alternative Horizontal, Vertical, and Diagonal Coefficient Implementation . 76
3.6. Quadtree Implementation 78
3.6.1. Quadtree Decimation and Output Spike Generation 78
3.6.2. Quadtree Address Output Generation 79
3.7. Quadtree Performance 80
3.7.1. Quadtree Simulations in Normal Operation 81
3.7.2. Quadtree Simulation in Saturation 81
3.7.3. Quadtree Energy Consumption 83
3.8. Full System Layout and Simulations 85
3.8.1. Haar Node Layout 85
3.8.2. Quadtree Layout 86
3.8.3. Haar Node and Quadtree PEX Simulations 86
3.8.4. 4×4 Pixel Array Layout 87
3.8.5. 8×8 Pixel Array Layout 88
3.8.6. 16×16 Pixel Array Layout 88
3.8.7. Concerns With Layout Feasibility 89
3.9. Power Scaling 90

4. Power Estimates **92**
4.1. Analysis 92
4.1.1. Leakage Power 92
4.1.2. Scene-Induced or Spike Power 94
4.1.3. Estimating the Haar Spike Rate Reduction f_w 97
4.1.4. Scaling Power by Process Node 98
4.1.5. Power Analysis of an Example 99

5. Uses for SPARR **104**
5.1. Signal Detection and Reconstruction 104
5.1.1. Introduction 104
5.1.2. Data 104
5.1.3. Algorithm Overview 107
5.1.4. Hypotheses 108
5.1.5. Parameter Estimation 108
5.1.6. Detection Computation 110

5.1.7. Time-of-Arrival Estimation 112
5.1.8. Implementation Details 113
5.1.9. Results 115
5.1.10. Future Detection and TOA Algorithm Work 118
5.2. Object Tracking 119
5.2.1. Experiment 120
5.2.2. Detection 121
5.2.3. Multiple Hypothesis Tracker 122
5.2.4. Future Tracking Work 123
5.3. Imaging with SPARR 124
5.3.1. Imaging Example 124
5.3.2. Conclusion 127

6. Future Work **130**

References **133**

Appendices **137**

A. Acronyms **137**

B. Sparrow User's Guide **139**
B.1. Executive Summary 139
B.2. Input File 139
B.2.1. Input YAML File 139
B.2.2. Input HDF5 File 146
B.3. Output File 146
B.4. Implementation Details 147
B.4.1. Performance Improvements to Likelihood Computation 147
B.4.2. Likelihood Normalization 147
B.4.3. Parallelizing Sparrow 148
B.5. Code Structure 148
B.5.1. ADEVS 148
B.5.2. Pixel Types 148
B.5.3. Processing Layers 149
B.5.4. Readout Methods 150
B.5.5. Data Output 151

C. Alternative Signal Processing **152**
C.1. Alternative Signal Processing 152
C.1.1. Matched Filter Spikes 152
C.1.2. Differentiator Spikes 153

D. Signal-to-Noise Ratio **155**

E. Spike Intensity Reconstruction via the Gamma Distribution **157**
E.1. Signal Model 157
E.2. Maximum Likelihood Estimator 158
E.3. Modified Estimator 160
E.4. Performance of the Estimators 161
E.5. Detection 162
E.6. Detection Examples 166
E.7. Reconstruction Algorithm 168
E.8. Performance Simulation 168

F. Time Decoding Machine for Reconstructing the Input Signal **173**
F.1. An Overview of TDMs 174
F.2. Integrate and Fire Reconstruction 175
F.3. Modifications for Real-time 176
F.4. Decimation in Time Decoding Machine Algorithm 177

G. Piecewise Constant Rate Approximation **179**
G.1. Reconstruction 179
G.1.1. Estimation 179
G.1.2. Performance 180
G.1.3. Implementation 181

LIST OF FIGURES

Figure 1-1. Integrate and fire circuit 14
Figure 1-2. Two-input/two-output digital asynchronous processing element (DAPE). 14
Figure 1-3. Example state machine 15
Figure 1-4. A full sunlit earth viewed by the sensor. 18
Figure 1-5. Haar transform for features and orientation. 20
Figure 1-6. Multi-level Haar transform 21
Figure 1-7. Levels within the Haar transform 22
Figure 1-8. Multi-level quadtree readout 22
Figure 1-9. Quadtree levels and example readout 23

Figure 2-1. Theoretical and empirical probability mass functions (PMFS) 32
Figure 2-2. Schematic of a Haar transform on a 2×2 array. 34
Figure 2-3. Schematic of a Haar transform on a 4×4 array. 35
Figure 2-4. Mean coefficient magnitudes by Haar discrete wavelet transform (DWT) level . . 36
Figure 2-5. The mean coefficient magnitude for each Haar element 37
Figure 2-6. Mean coefficient magnitude for horizontal/vertical throttling 38
Figure 2-7. The mean coefficient magnitude for each Haar element 39
Figure 2-8. The approximation state machine for all DWT levels except the apex. 40
Figure 2-9. The approximation state machine for the apex DWT level. 40
Figure 2-10. The horizontal and vertical state machine for all DWT levels. 40
Figure 2-11. The diagonal state machine for all DWT levels. 41
Figure 2-12. Synthetic earth scene with Kern spectrum 41
Figure 2-13. Throttled Haar plane from a `sparrow` simulation 42
Figure 2-14. Histogram of spikes in given pixel 42
Figure 2-15. Positive portion of the throttled Haar plane from a `sparrow` simulation. 43
Figure 2-16. The negative portion of the throttled Haar plane from a `sparrow` simulation. . . 43
Figure 2-17. The theoretical and empirical PMFS for the probability of net Haar spikes. 45
Figure 2-18. The enlarged state space for any of the major states. 46

Figure 3-1. asynchronous self-resetting pixel (ASP) conceptual schematic. 54
Figure 3-2. Focal plane array configurations. 55
Figure 3-3. Example of a simple asynchronous self-resetting pixel (ASP) circuit. 58
Figure 3-4. ASP with 3 μA photocurrent 59
Figure 3-5. ASP simulation around pixel reset 59
Figure 3-6. ASP frequency and power dissipation 60
Figure 3-7. Manufacturing variability effect on ASP frequency 61
Figure 3-8. Enlarged inverter effect on ASP frequency 61
Figure 3-9. Enlarged inverted effect on ASP power 62

Figure 3-10. Approximation coefficient block diagram. 63
Figure 3-11. Approximation coefficient circuit implementation. 64
Figure 3-12. Horizontal and vertical coefficient block diagram. 64
Figure 3-13. H/V circuit up/down control and clock generation 66
Figure 3-14. Horizontal and vertical coefficient circuit implementation – state machine. 66
Figure 3-15. Horizontal and vertical coefficient circuit implementation – output pulse driver. 66
Figure 3-16. Diagonal coefficient block diagram. 67
Figure 3-17. Diag. up/down control and clock generation 68
Figure 3-18. Diagonal coefficient circuit implementation – state machine. 68
Figure 3-19. Approximation coefficient simulated results. 69
Figure 3-20. Horizontal coefficient simulated results. 70
Figure 3-21. Diagonal coefficient simulated results. 71
Figure 3-22. Haar node outputs with colliding spikes. 73
Figure 3-23. Integral of the approximation coefficient power consumption. 74
Figure 3-24. Approximation coefficient alternative implementation. 75
Figure 3-25. Horizontal, vertical, and diagonal coefficient alternative implementation. 76
Figure 3-26. Horizontal, vertical, and diagonal coefficient alternative layout comparison. . . . 76
Figure 3-27. Lightweight c_H collision intolerance 77
Figure 3-28. Quadtree decimation circuit. 79
Figure 3-29. Quadtree output pulse generation circuit. 79
Figure 3-30. Quadtree output address generation circuit. 80
Figure 3-31. Quadtree output address driver circuit. 81
Figure 3-32. Quadtree simulated results for the lowest hierarchical level. 82
Figure 3-33. Quadtree simulated results with saturated input. 83
Figure 3-34. Integral of quadtree power consumption. 84
Figure 3-35. Haar node layout. 85
Figure 3-36. 2×2 layout. 86
Figure 3-37. Quadtree layout. 87
Figure 3-38. 4×4 pixel layout. 88
Figure 3-39. 8×8 layout. 88
Figure 3-40. 16×16 layout. 89
Figure 3-41. Power and pixel pitch scaling. 91

Figure 4-1. Pixels, Haar, and quadtree levels 93
Figure 4-2. Estimation of f_w from a scene. 98
Figure 4-3. Power use as pixel spike rate varies 102
Figure 4-4. Power use as CMOS node varies 103

Figure 5-1. Preprocessing 105
Figure 5-2. Spike Distribution 106
Figure 5-3. Distribution Comparison 107
Figure 5-4. Buffer at time of event detection. 112
Figure 5-5. Probability of detection 117
Figure 5-6. ToA estimation error targets at 2% of background 118
Figure 5-7. ToA estimation for a signal that is 1.5% of background 119

Figure 5-8. An example of tail trigger. 120
Figure 5-9. Simulated star paths for a tracking experiment 121
Figure 5-10. Detection output for tracking 122
Figure 5-11. Output from multiple hypothesis tracker (MHT) 123
Figure 5-12. Image of the earth. 125
Figure 5-13. Percent error for each decimation as a function of spike count. 126
Figure 5-14. Image reconstructions for the low decimation simulations. 127
Figure 5-15. Image reconstructions for the high decimation simulations. 128

Figure 6-1. Evolution of a reconstructed image 131

Figure C-1. Matched filter decimation 153
Figure C-2. Filters are put in row-major order 153
Figure C-3. Differentiator decimation 153

Figure E-1. Mean of estimators 162
Figure E-2. Percent error of estimators. 163
Figure E-3. Variances of the maximum likelihood estimators (MLES) 163
Figure E-4. Probability of detection as a function of signal level 166
Figure E-5. Probability of detection as a function of signal level. 167
Figure E-6. Probability of false alarm as a function of signal level. 168
Figure E-7. Probability of detection vs. the number of photon arrivals 169
Figure E-8. Probability of detection vs. the number of photon arrivals 170
Figure E-9. Probability of detection vs. background photon arrival rate 170
Figure E-10. Signal reconstruction on a target 10× background 171
Figure E-11. Signal reconstruction on a target 1× background 171
Figure E-12. Signal reconstruction on target 0.04× background 172

Figure F-1. An integrate-and-fire (IAF) neuron. This figure was taken reference [22]. 173
Figure F-2. TDM reconstruction. This figure was taken from reference [22]. 175

LIST OF TABLES

Table 1-1. Example data sequence 16
Table 1-2. Some choices of Spiking/Processing Array (SPARR) layers 17

Table 2-1. Pixel labels for a 2×2 focal plane array (FPA). 31
Table 2-2. Parameters used for the `sparrow` simulation. 32
Table 2-3. The theoretical and empirical positive and negative Haar spike means. 50

Table 3-1. 2×2 array orientation. 62
Table 3-2. Approximation coefficient state machine operation. 63
Table 3-3. Horizontal and vertical state machine operation. 65
Table 3-4. Diagonal coefficient state machine operation. 67
Table 3-5. Haar node power consumption. 75
Table 3-6. Quadtree power consumption. 84
Table 3-7. Haar node and quadtree power comparison. 87

Table 4-1. Selected CMOS process nodes 99
Table 4-2. Notional parameters for analysis, from 90 nm process node. 100
Table 4-3. Power estimates for a specific case 101

Table 5-1. Thresholds used for each algorithm. 116
Table 5-2. Algorithm parameters. 116
Table 5-3. Number of spikes used for the image reconstructions. 129

1. INTRODUCTION

The Spiking/Processing Array (SPARR) is an optical focal plane array with signal processing embedded in the integrated circuitry. Signal processing has been embedded in focal planes at least since Carver Mead [24] or Buchanan *et al* in 1994 [4]. Embedded signal processing continues to more recent designs like MIT Lincoln Lab's Digital Focal Plane Array [32], Sandia's Ultra-High Dynamic Range Focal Plane Array [30] and Hyper-Temporal Sensor [13] or the Event-Based Sensor/Silicon Retina [6].

In its most abstract form, SPARR is an array of independent photo transducers and a net of interconnected signal processing elements. The pixels are asynchronous (clockless) and are called asynchronous self-resetting pixels (ASPS). The signal processing elements are digital though they also operate without an external clock and are called *digital asynchronous processing elements* (DAPES). ASPS and DAPES are joined in a network or reticulum. The propagation of spikes from pixels through the reticulum of DAPES performs signal processing. In other words, the topology of the DAPES configures the system.

A SPARR pixel has a unique and defining property: it generates an electrical spike when it has satisfied some internal condition. The most straightforward design is the integrate and fire rate converter. Interestingly, Louis Lapicque published on the integrate-and-reset or integrate-and-fire model in 1907 [18], so the underlying concept is far from new. Figure 1-1 shows an idealized integrate-and-fire circuit schematic where a photodiode delivers photoelectrons to an integrating capacitor. The integrating capacitor's voltage rises as more charge accumulates until the voltage exceeds a reference. Upon exceeding the reference, the circuit emits a short-duration electrical spike which also empties the integrating capacitor. Section 3.2 provides a more thorough treatment of this kind of pixel.

An ASP must produce a spike as a result of the incident optical signal. Beyond that, a great deal of flexibility is possible. For example, a Geiger mode single-photon avalanche photodiode (SPAD) might produce a spike when struck by a single photon thus making SPARR a single-photon spike processing system. At the other extreme the photocurrent could partially quench the quantum efficiency causing the spike rate to be proportional to, for example, the logarithm of the incident signal. Some analog processing could be performed in the pixel, such as a deliberate leakage resistance added to the integration capacitor. With that design we get a leaky integrate and fire response, essentially turning a spike into a high-pass filter and threshold operation. Myriad designs are possible, but all ASPS emit spikes into the processing reticulum without an external clock. Unless otherwise specified, this report assumes that pixels are linear-response integrate-and-fire circuits.

Spikes from ASPS feed DAPES. A DAPE is a digital circuit, meaning its processing elements are bistable – representable as zeros and ones. This is important, as the SPARR design eschews analog

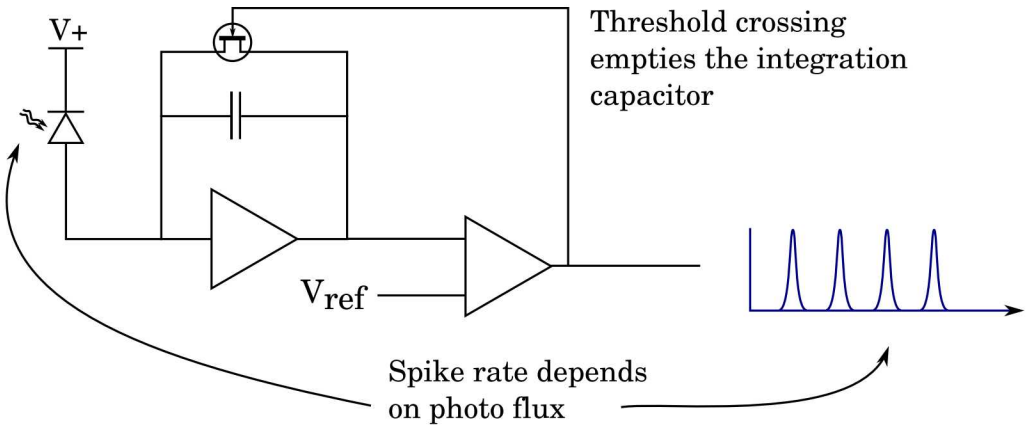

Figure 1-1. A simple ASP can be an integrate and fire circuit, essentially a photon flux to spike rate converter.

application-specific integrated circuit (ASIC) concepts to minimize fabrication variability. A DAPE is a state machine with an arbitrary number of input and output ports. However, most of the DAPE designs developed in this report are two-input and two-output as shown in figure 1-2. All electrical spikes in SPARR are essentially identical, there is no concept of a *positive* or *negative* spike defined by its electrical characteristics. Nevertheless, we assign the spike a sign or value by its location. That is, a spike on the positive input of a DAPE is thought of as a positive spike while one on the negative input is thought of as negative. These sign values are strictly semantic since they are electrically indistinguishable.

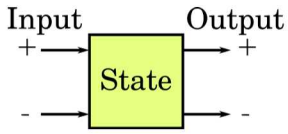

Figure 1-2. Two-input/two-output DAPE.

A DAPE state machine transitions to a different state depending on the input it receives. It may generate an output spike as well. Figure 1-3 depicts the behavior for a three-state machine with two inputs, and two outputs. If the DAPE is in state +1 and receives a positive spike then it will emit a spike on its + line and maintain its internal state. Instead of returning to the initial state, an alternative design would have the DAPE emit a spike and then go to, for example, state 0.

The DAPE shown by figure 1-3 performs signal processing operations related to its two input lines. If we think of the spike rate on the positive line as r_+ and the rate on the negative line as r_-, and the positive and negative output rates as ρ_+ and ρ_- then the approximate behavior of the circuit is

$$\rho_+ \approx \begin{cases} r_+ - r_- & r_+ > r_- \\ 0 & \text{otherwise} \end{cases}, \tag{1.1}$$

$$\rho_- \approx \begin{cases} r_- - r_+ & r_+ < r_- \\ 0 & \text{otherwise} \end{cases}. \tag{1.2}$$

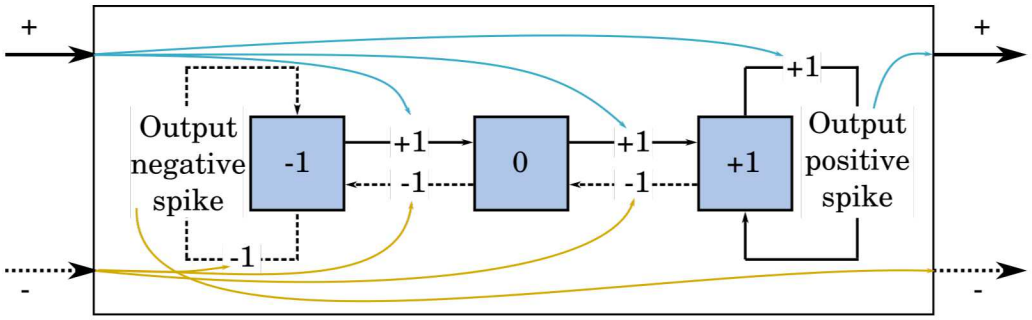

Figure 1-3. Example two-input/two-output state machine with three internal states. In state -1 or state +1 an incoming spike may cause the DAPE to emit a spike and return to the same state.

If the two input rates are approximately equal ($r_+ \approx r_-$) then the DAPE will emit few spikes, even if both rates are quite high. Similarly, if one input has rate near zero the DAPE will essentially forward the non-zero line (see section 2.3). The preceding argument assumes SPARR is a rate-coded process. It is not, at least not entirely since the DAPEs depend on the exact order of the input spikes, not just the rate. Nevertheless, the rate-coded simplification is often adequate when upstream processing elements have no dependence on time.

Although digital, the DAPE retains one key analog characteristic: time. Because it is asynchronous, the relative sequence of arriving spikes completely dictates its output. For example, a spike arriving on the positive input line moments before a spike arriving on the negative line may lead to an output spike; reversing the spike order may not. A conventional digital system would treat those spikes sequences identically provided they occurred within a single clock cycle. Mathematical operations are evidently possible, at least approximately, with DAPEs. To perform useful signal processing, though, multiple DAPEs and ASPs must cooperate. Chapter 2 discusses signal processing operations in detail, while the design of the circuits to implement that processing is the topic of chapter 3.

The spikes or information about them (such as a spike count from each DAPE) must move off the array. We have considered several approaches, including detecting and reading out all the DAPE counts from the most active overall row in the array and the quadtree. Most of this report assumes the quadtree readout, which minimizes the chance of spike collisions, provides unbiased readout, and whose flow control is dynamically adjustable. The quadtree readout uses quadtree DAPEs stacked in a pyramid topology so that the base of the pyramid attaches to the output of the signal processing layer. Each quadtree DAPE has four spiking inputs, one spiking output, and an appropriate number of address lines to reflect the location in the focal plane that originated the spike. For example, at the base of the quadtree pyramid, four spike lines enter each quadtree DAPE and the output of is a spike line and a two-bit address line. The second layer in the quadtree receives the spike line and address lines from four first-layer quadtree DAPEs. In addition to forwarding the incoming address bits, the second layer quadtree DAPE must include two additional address bits. Once enough low level pixels and DAPEs are aggregated, the quadtree terminates at an apex node (the tip of a pyramid). Since the Haar transform emits spikes on both positive and negative lines, the design includes two separate quadtrees, independently throttled, one for

negative-line outputs and another for positive-line outputs.

Mathematical operations are evidently possible, at least approximately, with DAPES. To perform useful signal processing, though, multiple DAPES and ASPS must cooperate. Sections 2.3 discusses signal processing operations in detail.

The apex of the quadtree emits an asynchronous sequence of addresses and an associated spike. Each spike captures the time from a local clock at the edge of the FPA before exiting the array. Time stamping transitions the spike from the asynchronous electronic domain to the synchronous (clock-driven) domain and is a precursor to conventional processing. We imagine these asynchronous addresses are pushed into a dual port circular memory buffer that is drained by high-speed serial lines. The fullness of the buffer is a measure of the spike rate, and if the buffer is filling faster than it can be drained the quadtree is commanded to decimate spikes (or throttle). Conversely, if the buffer is emptying faster than it can fill, the quadtree DAPES reduce throttling. Table 1-1 shows a caricature of the sequence resulting from the quadtree readout. The amount of throttling dictates how many pixel spikes are read out, and how many are correspondingly lost.

Table 1-1. Example data sequence from readout of a SPARR with a positive quadtree, negative quadtree, 2 GHz clock, and 1024Œ1024 pixels. A global throttle control setting is needed to interpret these results.

Tree	Address (row, col)		Clock (ns)
–	0428	0992	1538889.5
+	0207	0322	1538992.0
+	0569	0512	1539088.5

The clock frequency controls SPARR's time resolution, but not its fastest spike rate. The fastest spike rate is dictated by the ASP and DAPES, which in the 90 nm complementary metal-oxide-semiconductor (CMOS) process allows for distinct spikes separated by about 0.25 ns. Smaller process nodes would cycle more quickly. The entire array cannot spike that fast *and read the data out*, but the spike rate still justifies the coarsest precision to be about 0.5 ns. Finer resolution may be beneficial when considering SPARR as a time encoding sensor in addition to a rate encoding sensor. Other readout technologies are possible, such as counting each DAPE's spikes and then reading the counts; however, the quadtree approach has a integrated throttling and provides very fine time resolution.

SPARR may be thought of as three technology layers assembled. A transduction layer (ASPS) converts light into spikes which feed a processing layer made of DAPES to perform some kind of signal processing, and these in turn supply a readout system which may also be composed of DAPES. Table 1-2 shows examples from each layer, where one component would be taken from each layer to produce a complete SPARR.

The unusual data supplied by SPARR challenges detection and time-of-arrival (TOA) estimation familiar from framing readout sensors. Although we are confident that direct operations using the time of spikes can perform detection and TOA estimation, we have addressed the problem by first counting spikes into time bins and then performing detection and estimation on the resulting

Table 1-2. Some choices of SPARR technology layers. A design needs to choose one from each column. Most of this report uses a linear integrate-and-fire ASP, Haar spatial wavelet transform, and quadtree readout. Most of this paper discusses the chosen elements highlighted in green text.

Pixel (ASP)	In-Array Processing	Readout
Linear response integrate and fire in CMOS with microlens		
Linear response integrate and fire with hybridized indium silicon detector		
Hybridized alternative material, e.g. III-V (InSb)	Whitened Haar spatial wavelet transform	Quadtree
Geiger-mode single photon avalanche photodiode	Unwhitened Haar spatial wavelet transform	Maximum activity row
Linear-mode avalanche photodiode integrate and fire	Spatial matched filter	
Non-linear response integrate and fire		
Leaky integrate and fire		

sequences. This work, the topic of section 5.1, should give the reader confidence that SPARR can perform the work of conventional imagers in spite of its unusual function. In fact, we believe SPARR can perform operations that are not possible with classical framing sensors. Such capability growth operations include spike-time exact TOA estimation, sub-frame motion compensation, and support of spiking neuromorphic hardware for applications like autonomous vehicles.

1.1. HOW SPARR WORKS

The preceding sections have explained how the parts of SPARR work, but only loosely connected. To connect these pieces, it is helpful to compose ASPS and DAPES together to perform a function, and explain a little of why that function matters. The specific details are illuminated in detail later in this report; here we use a linear integrate-and-fire ASP array, connected to a whitening Haar spatial wavelet transform, which is subsequently connected to a quadtree readout.

Assume the sensor is viewing the image in figure 1-4, a full sunlit earth. The brighter a pixel is, for example the clouds over South America in the center of the image, the faster its capacitor fills with electrons causing it to spike. The dark pixels, beyond the edge of the earth, only spike because of the eventual accumulation of dark current or scattered light.

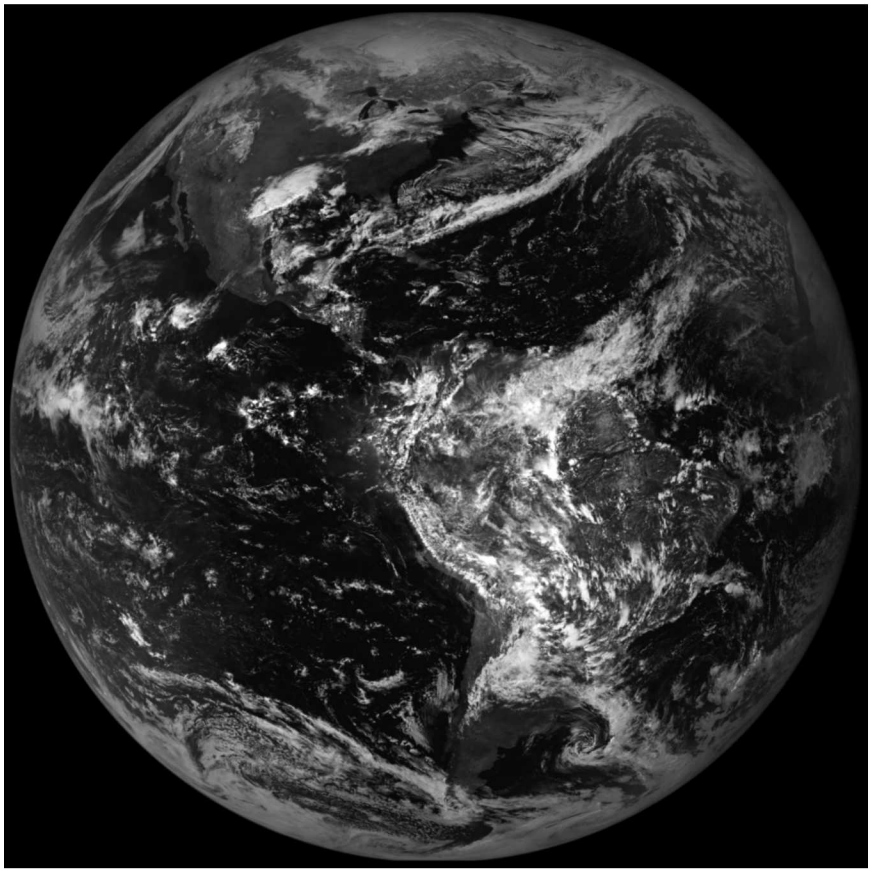

Figure 1-4. A full sunlit earth viewed by the sensor.

The Haar wavelet transform layer, a carefully connected network of DAPES, converts this image to coefficients. It is worth exploring why we chose that process. Consider biological processes where the human optic nerve transmits information content of about 875 000 bits/s [16]. There are 6.4 million cones in the human eye – think of these as the eye's "pixels" [17]. Each of these, on average, has an integration interval of 100 ms [1]. If we a assume a time and amplitude precision of about 10 bit/spike[1] then the retina is *generating information at* 6.4×10^8 *bit/s*. Pushing this information down the optic nerve's 8.75×10^5 bit/s capacity means compressing the retina's information about 700:1. Even allowing for considerable error in the estimate of the retina's information generation capacity, it is clear that the optic processing in the eye is performing (lossy) compression.

Practical FPAS also struggle with the readout bottleneck; it is readily possible to generate more information in the focal plane than any readout can support. SPARR borrows inspiration from biological systems and performs compression using the Haar wavelet transform. Biological systems may use a Gabor-like transform in vision systems [23]. The Gabor transform has the properties of emphasizing edges in multiple orientations, operating across a wide range of scales, and executing compression [7]. Like the Gabor transform, the Haar transform emphasizes edges with multiple orientations (horizontal, vertical, and diagonal), it works over many scales or feature sizes, and as we show shortly, it can execute compression. The Gabor transform is not easily implemented in digital electronics, but the Haar transform is, which is why it was chosen.

The Haar spatial wavelet transform works by applying four convolutional kernels to the image, and then decimating the image 2:1 in both rows and columns. Figure 1-5 shows the unity coefficients and the first level of the transform.

Inside SPARR, the wavelet transform applies spike decimation at each level so that each scale and orientation within the scene receives the same emphasis and readout bandwidth. The left part of figure 1-6 shows only the first level of the decomposition. Although the coefficients from the first level of the decomposition make of 75% of the pixels, it only represents about 12% of the total energy in the image. Kern [15] found that natural scenes tend to have a consistent power-law spatial power spectral density. In other words, natural scenes have a predictable fraction of their energy at each level of the 2D wavelet transform. SPARR exploits this property of natural scenes to scale each level of the Haar wavelet transform so that the coefficients all have approximately the same amount of energy. That is, the scene has been spectrally whitened.

As figure 1-7 shows, successive application of the transform using the c_A coefficients creates a hierarchy. Spectral whitening by scaling each wavelet transform level in the hierarchy using spike decimation performs a kind of lossy compression. It is lossy in the sense that decimating spikes introduces an effective quantization error. However, the information redundancy at later levels of the wavelet transform means that loss is of little practical consequence.

The calculated coefficients, represented by spikes emerging from DAPES, exist only inside the FPA. To be useful, the spikes must leave the array through a readout. The quadtree readout, depicted schematically by figure 1-8, funnels the spikes to a single readout stream where they are time tagged. At the left edge, four coefficients' spikes enter a quadtree element. The element forwards

[1] It is difficult to estimate the information content in an integration interval for an eye's cone; however the combination of time coding and place coding in reference [16] suggests about 10 bit/spike along the optic nerve.

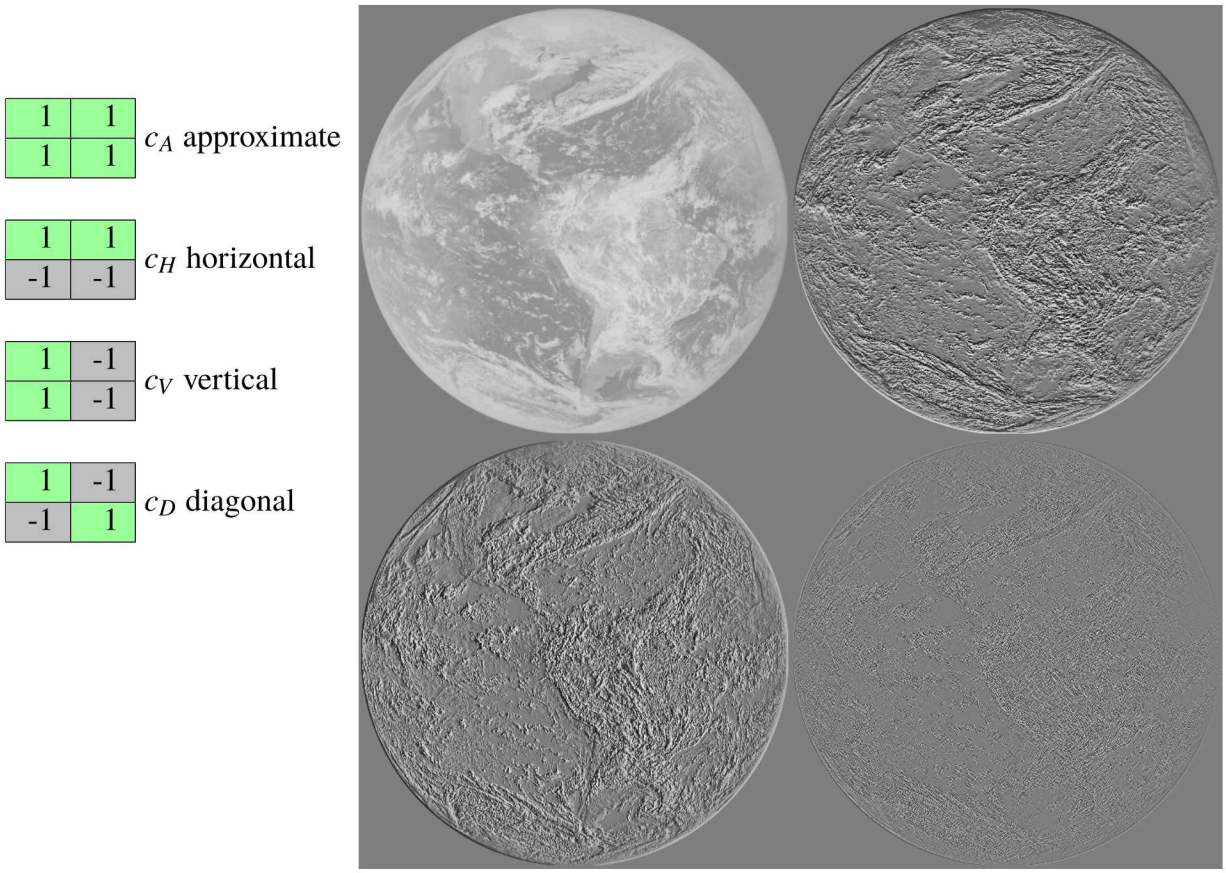

Figure 1-5. Haar transform for features and orientation. Gray intensity is scaled with the hyperbolic arcsine (arcsinh) function to reveal contrast. The upper right quadrant is the result of applying the c_H kernel and downsampling 2:1 in horizontal and vertical directions. The lower left the c_V kernel, lower right the c_D kernel. The upper left values are from the c_A kernel, and have been scaled by 0.05 prior to applying the arcsinh for visibility.

the spike to its output, along with the two bit address indicating which of the four inputs spiked. Suppose that the entering coefficients are spiking very quickly, near the maximum operational spike rate (around 1 GHz for the 90 nm CMOS process). In this case, the output of the quadtree element would have to spike four times as fast, a rate it cannot sustain. To keep the quadtree aggregated spike rate manageable when the internal of the FPA are spiking very actively, each quadtree element can decimate its inputs by 2:1 or 4:1. In 4:1 decimation, the quadtree element only passes every fourth input spike to its output. Each quadtree layer's decimation is independently controlled, and is consistent for all elements in the tree.

The quadtree coalesces all lower level spikes until they reach an apex node. Figure 1-9 shows how spikes from SPARR's wavelet transform layer flow into an ever smaller channel. Each quadtree layer has one fourth as many elements as the previous layer, and two more address bits.

The linear spiking pixel, whitening Haar spatial wavelet transform, and quadtree readout is this

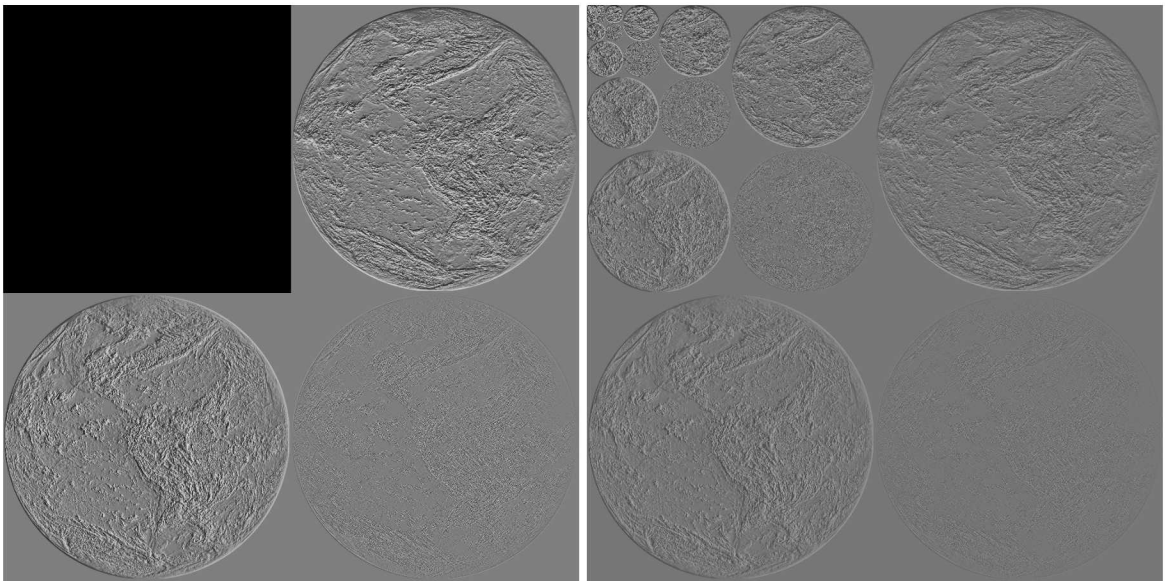

Figure 1-6. The approximation coefficients are scaled by a predetermined constant and passed through the Haar transform again. This is repeated successively until the only remaining data has been through c_H, c_D, or c_V convolution, resulting in values represented by the right figure.

report's primary focus. Some designs considered included readout based on most-active-row first, and spatial differentiation to emphasize point-like features in the scene. Numerous other designs are possible, and fit under the SPARR umbrella.

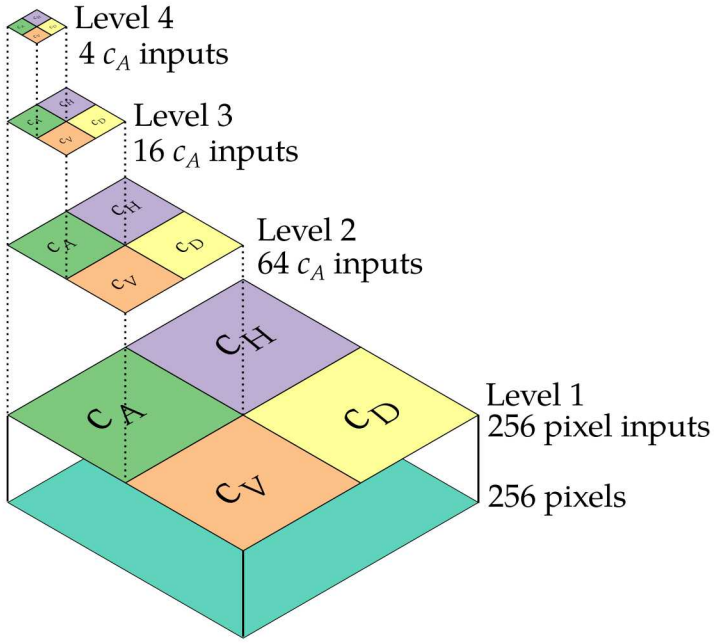

Figure 1-7. A 256 × 256 array example of the levels in the Haar transform. The c_A coefficients pass through the Haar transform again.

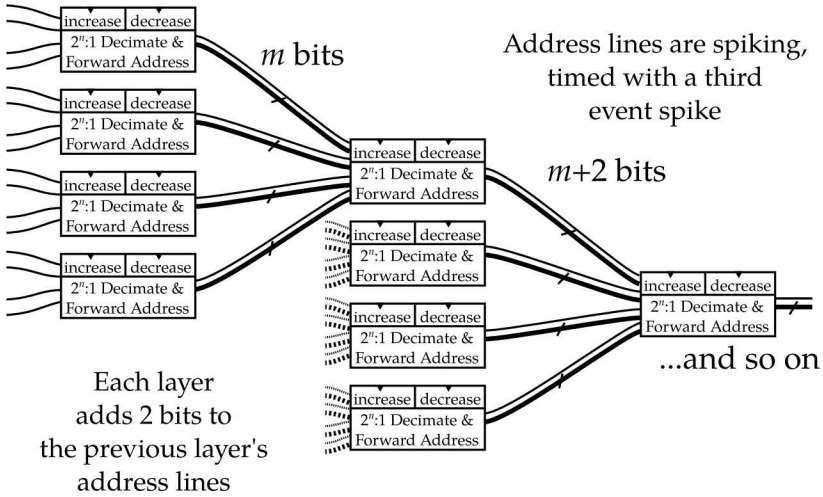

Figure 1-8. Haar coefficient spikes enter the quadtree readout on the left, four coefficients into each quadtree element. The entering spikes exit, possibly after decimation, with the input address. This 4:1 element aggregation occurs at multiple levels, adding two address bits until the spike reaches the apex node, is timestamped, and read out.

Time (μs)	Row	Col
1.10773	48	19
1.32928	21	11
1.55082	53	17
1.77237	12	3
1.99392	25	32

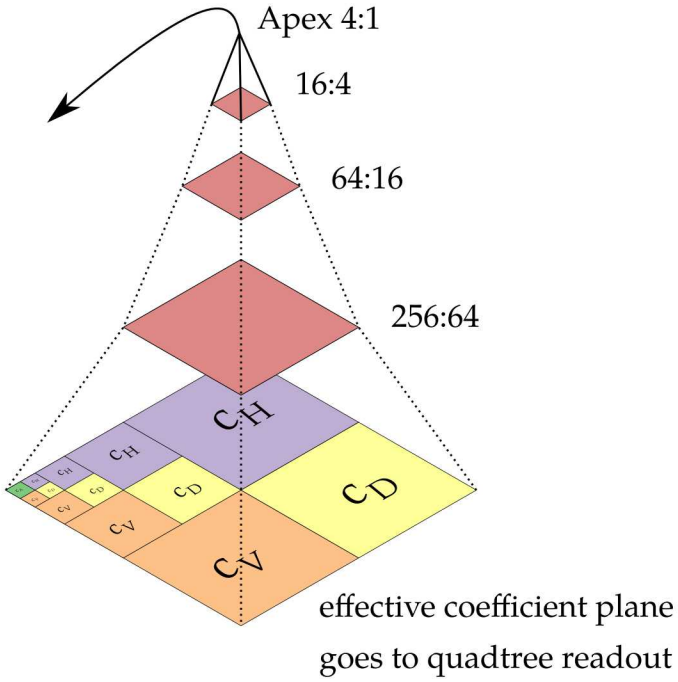

Figure 1-9. Quadtree levels and example readout. The table, at left, is a simplified representation of the stream of output produced by SPARR from the quadtree readout process (right).

1.2. ADVANTAGES OVER OTHER FOCAL PLANE ARRAY DESIGNS

As with any design, there is a trade-space to iterate and optimize over. However, an ideal focal plane would provide

- Dynamic range to accurately measure optical signals from very small values to very large values to support a large class of signals,
- High spatial resolution to distinguish closely spaced objects or accurate measure angle to target,
- High measurement rate to record fast-changing or short-lived signals,
- Precise time to support time-of-flight sensing,
- Data rate large enough to deliver all the measured data to a processing consumer,
- Low latency or delay between the measurements and that measurement's availability off-array to support fast reaction,
- Low power dissipation to facilitate low-temperature detectors and minimize impact on focal plane hosts.

These engineering objectives are often in conflict. To read off all data, the supported readout rate (bit/s) must exceed the product of the dynamic range (bit/sample), measurement rate (sample/s/pixel), and spatial resolution (pixel). For example, a 1024Œ1024 pixel sensor with 100 kHz sample rate and 12 bit/sample resolution produces 0.14 TiB/s. For context, a Xilinx 16-lane GTH transceiver supports 0.03 TiB/s [39]. While the specs on this hypothetical array seem impressive, 100 kHz provides temporal resolution of 10 s and speed-of-light range precision of 3 km. While neither latency nor power dissipation are easy to assess, they are likely both poor.

To address the off-chip bottleneck, digital focal planes make trade-offs. Some common trade-offs include

- Recording full-frame data for only a short interval instead of continuously, typically while providing high-capacity cooling, as in the Phantom v2640 which supports 0.9 Mpx at 19 690 frames/s for 15.4 s [28],
- Processing pixel time-series in the focal plane to select pixels for readout based on rapid signal amplitude changes, or based on small command-selected regions as in the Hyper-Temporal Sensor [13],
- Pre-processing images in the focal plane to, for example, stabilize images or remove backgrounds thus providing useful information in a slower readout as in MIT Lincoln Lab's Digital Focal Plane Array [32],
- Exploiting scene redundancy to compress the data (lossy) while retaining high temporal resolution, as in SPARR.

As will be shown later in this report, the spatially whitened Haar transform used by SPARR relies on the statistical characteristics of typical scenes to perform data compression. The whitening Haar transform reduces the total data, as measured by spike counts, by 80% to 90% for typical natural scenes. In cases where this compression still results in readout rates exceeding the FPA's capacity, SPARR reacts by elastically throttling the spike rate through the quadtree. Throttling, or discarding every *n*th spike, is an unbiased data reduction technique. The net result is that for a range of average ASP spike rates from zero up to about five times the readout capacity there is negligible scene measurement loss since there is no spike decimation throttling. At higher average rates the amount of decimation throttling increases, to fit the propagated spike rate within the readout capacity. Consequently, SPARR has elastic amplitude resolution, finer for small signals and coarser for large. Correspondingly, its power dissipation is extremely low at low spike rates, gradually approaching an upper limit where both maximum spike rate and maximum throttling are occurring. This gives SPARR a wide operational dynamic range.

Finally, SPARR differs from other sensors because its output, at the edge of the array, is spikes. These spikes are naturally adapted to spiking neuromorphic processing, with technologies like Intel's Loihi processor [8] or IBM's TrueNorth [25]. We are aware of no other large-format optical sensors that can natively provide spiking data.

1.3. POTENTIAL APPLICATIONS

No short prescription will contain every possible application of a technology. A few to keep in mind, though, include transient detection, imaging from unstable platforms, and power-sensitive applications.

A transient optical event's intensity varies over time, typically it is not emitting light, starts emitting, and then stops. How the emitter's intensity fluctuates or modulates can be important information for the sensor operator. A variety of natural events are optically transient, think of lightning as an example. Lightning is challenging to observe because, most of the time, it is *not happening*. Detecting such transients typically demands continuous observation; there is no way to anticipate when the event will occur. In support of this, SPARR provides constant streaming of data instead of a finite cache with readout. Some applications use the time of the transient directly. For example, LIDAR uses time-of-flight to measure the range between the sensor and reflectors. SPARR's continuous streaming and fine time resolution are essential to these transient monitoring applications. Section 5.1 shows example of transient detection using a SPARR with linear-response integrate-and-fire pixels feeding spiking Haar wavelet transform which subsequently feeds a quadtree readout.

Imaging, the taking of photos, usually depends on stability during the capture integration interval. That is, usually users would like to assume that the scene has not changed between when the shutter opens and when it closes. Well-lit conditions without significant platform motion satisfy this objective; however, many sensing platforms are not stable over the integration interval. Instability can be due to slow motion with long integration intervals for low light observation, or bright conditions with fast moving platforms like imaging from a shaking car or aerial vehicle. A

spike inside SPARR is conceptually similar to a single count in other digital imagers. In other words, SPARR's integration interval is not constant – it lasts as long as it takes to acquire a single count. This can often be orders of magnitude faster than a typical framing sensor. Although not explored thoroughly in this project, reconstruction of a motion-compensated image is possible from the spike train and can effectively increase the frame rate of the sensor dramatically.

Finally, almost any optical sensor application must face the necessary evil of powering the optical sensing device. Power supply to a sensor means its host must be sized appropriately. Whether power is generated by motors, fuel cells, solar panels, or batteries all would grow to support more demanding sensors. The other side of power supply is heat dissipation. The greater the heat dissipated in the sensor, the larger the cooling capacity needed. For room-temperature sensors in the atmosphere greater cooling may mean larger heat sinks and fans. On orbiting platforms heat dissipation requires more careful design to balance radiators. For cryogenic temperature sensors typical of infrared applications, like astronomy, the cryogenic thermal management efficiency typically results in *much larger* and more expensive coolers. Chapter 4 shows that most of SPARR's power dissipation is dynamic, meaning it depends on the scene. A star field in the dark generates few spikes, consequently the power dissipation for a 256×256 pixel array made with 90 nm CMOS process is predicted to be as low as 36 mW. A larger 1024×1024 array on a bright scene generating 6 million spikes per pixel per second may use 6.7 W.

1.4. SPARROW INTRODUCTION

1.4.1. Overview

The SPARR simulator, `sparrow` is designed to model the SPARR array to enable performance analysis of this concept. Modeling a clockless architecture in a timely fashion is inherently difficult since there is no basis for synchronization among the different elements. The major simulation challenges for this design included correctly modeling photon arrival statistics, modeling the processing elements used for the Haar transform and the quadtree readout, and ensuring that the simulation ran fast enough that performance could be assessed in a reasonable timeframe.

This type of simulation is typically known as a discrete event simulation since the simulation state only changes at discrete points in time and the state is constant between those points. For SPARR, the external influences that cause a change in state are photons arriving at the sensor. Internal states can change as spikes propagate through the system to the readout.

1.4.2. Modeling a Pixel

Typical scene simulation codes use radiometry to compute the number of photons arriving at a pixel in a given integration period. Since SPARR uses an integrate and fire model for a pixel, this physics model no longer works. Instead, `sparrow` uses a likelihood function that describes the probability that a photon will arrive in a given period, or the time until N photons have arrived at a pixel, where N is the number of photons that will cause a pixel to output a spike and reset.

Multiple iterations of this process were developed and evaluated until the final solution supporting moving, time-varying functions in a computationally tractable fashion was developed and implemented. Pixel effects, such as non-uniformity in offset and gain can also be simulated.

1.4.3. Modeling the Haar Transform

The pixels in the array are connected to Haar processing elements following the logical Haar transform layout that will be described elsewhere. Each Haar element models effects due to processing delays through the element, the time it takes for an element to process another spike, and spike collisions on the inputs.

1.4.4. Modeling the Quadtree Readout

The output from the Haar transform feeds two quadtree readout structures, one for positive spikes, and one for negative spikes. These quadtrees also model the effects of element processing time, reset time, and spike collisions. Each element, or level in the quadtree, is capable of apply a single, double, or quadruple decimation to the spike output rate from that element. There is a top-level monitor that decides whether to increase or decrease the global throttling rate for the entire quadtree, and it adjusts the rate to try and keep the number of spikes output within an upper and a lower bound.

1.5. HOW TO READ THIS REPORT

The introduction, so far, provided an overview of the pixel or ASP which converts light into spikes. These spikes supply the processing units (DAPES) which perform a Haar spatial wavelet transform. In turn, the results of the wavelet transform supply the quadtree readout and thus takes the spikes off the array. With this overview in mind, the organization of the remainder of the report is:

Chapter 2 uses a rigorous mathematical analysis to determine important implementation details for the Haar transform, such as deciding how much decimation needed between transform levels. As part of its statistical assessment, it validates the system-level `sparrow` simulator used for other SPARR work with a clear picture of what must be implemented in the circuitry of SPARR.

Chapter 3 designs circuits for the ASP, including discussions of trade-offs between spectral responsivity, interconnection capacitance, and the resulting threshold voltage effects. The detailed specification from the previous chapter is used to define the Haar transform circuit elements, as is the looser specification for the quadtree elements. These elements are realized in simulated, detailed, circuit layouts. The non-ideal circuit behavior (run-length effects) with parasitic extraction, and then SPICE simulation provides reset time, collision conditions, leakage power, and per-spike energy estimates. These estimates are used in a chapter on system power consumption.

Chapter 4 develops a model for power consumption both from a theoretical framework and which can be used to assess power consumption in SPARROW calculations. The model shows how most power in SPARR is dynamic (depends on spike rate) and how power scales with both CMOS process node and with spike rate. Realistic quadtree throttling is represented. With the actual SPARR model in place, it only remains to demonstrate its utility.

Chapter 5 connects SPARR to the kinds of signal processing operations likely to be familiar to many readers, namely detection of transient events and estimation of the TOA of those events. Detection of signals as small as 1% of background is demonstrated, along with negligible TOA error for signal-to-noise ratios better than about 2.

Chapter 6 discusses some of the remaining work. SPARR produces an unusual type of data and although chapter 5 demonstrate utility, SPARR presents opportunities that have not yet been thoughtfully considered.

Appendices provide greater detail on elements that may not be of interest to the average reader. Some of these, like the SPARROW user's guide represent the considerable body of work required to actually simulate SPARR. Additional background that proved exceptionally useful to the SPARR team is included, along with an explanation of methods of signal time-series reconstruction.

While any chapter may be read without the others, the argument for the realized creation, fundamental performance, and application will make the most sense read sequentially.

2. STATISTICS FOR A NOMINAL EARTH SCENE

2.1. INTRODUCTION

The basic information that the SPARR sensor produces is spikes, and the arrival of photons on the SPARR sensor is a probabilistic process. Therefore, if we can understand how photons arrive on the sensor under typical conditions (i.e. no event of interest is occurring), then we can compute the probability of spike production from all of the image processing elements since they work in deterministic ways. Understanding the SPARR output under typical conditions allows us to better analyze what the sensor does when an event of interest does occur. Further, computing the probability of spike production allows us to verify the results we obtain from the SPARR simulator, `sparrow`.

The image processing units referred to above are the actual pixels that produce the spikes, the throttled Haar transform, and the quadtree. The pixels, of course, are the elements that actually gather the energy deposited by the photons on the array. The throttled Haar transform is used to throw away a lot of the unwanted information. It takes the pixel spikes and inputs them into a series of state machines that emulate a Haar transform with the exception that the Haar coefficients are altered or "throttled." Essentially, it is a spatial whitener so that anomalies or events of interest are more apparent. Finally, the quadtree allows us to read out the throttled Haar spikes in a single stream of data. This stream will give us the time, address, and weighting for each Haar spike. A weighting is needed because there is a finite amount of time for the hardware to register a spike. If all spikes were put into a single readout, depending on the photon arrival rate, SPARR would try to register two or more spikes in too short a time interval. This would cause us to lose all but one spike during this time period, which could severely alter the final output. Hence, the quadtree decimates the Haar spikes in a deterministic way so that we can properly assign a weight to a spike in the readout.

We will go through each of the image processing elements in turn, describing how they would work under ideal conditions and the underlying probabilities for spikes to come out of each element. These theoretical probabilities are then compared to a `sparrow` simulation to show how much agreement exists between theory and practice.

2.2. PIXEL SPIKES

Pixel spikes are driven by the photon arrivals at the sensor, the pixel capacitor size, and the quantum efficiency. For the analysis below, we will ignore the effects of quantum efficiency, which means we will assume all optical photons affect the system equally. Hence, we model each pixel as an energy accumulator where the energy is stored in a capacitor. Each optical photon that arrives at a pixel is assumed to deposit an electron in the capacitor. When a pixel's capacitor is full, it dumps its contents, which creates a pixel spike. We will assume an ideal capacitor without thermal noise. That is, if the capacitor can store N electrons, we will assume it will dump its contents when it reaches exactly N electrons. Further, we will assume it will dump all of its charge so that it is left with 0 electrons immediately after an electron dump.

2.2.1. Photon Arrivals

Throughout this section, we will make much use of the Poisson distribution. The Poisson PMF is given by

$$f(n;\lambda) = \begin{cases} \dfrac{e^{-\lambda}\lambda^n}{n!}, & n \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{2.1}$$

where λ is the mean number of some event per unit time. Equation (2.1) gives the probability of some discrete event occurring n times in a time interval Δt. The mean and variance of $f(n;\lambda)$ are both λ. The Poisson cumulative mass function (CMF) is given by

$$F(n;\lambda) = \sum_{i=0}^{n} f(i;\lambda). \tag{2.2}$$

Here, we use the standard convention in statistics of using lowercase letters for PMFs and capital letters for CMFs. Finally, it is also useful to note that the probability of some discrete event occurring between n_1 and n_2 times with rate λ is given by

$$\sum_{i=n_1}^{n_2} f(i;\lambda) = F(n_2;\lambda) - F(n_1 - 1;\lambda). \tag{2.3}$$

The arrivals of photons on a given pixel is assumed to be a stationary Poisson event in the absense of an event of interest. That is, the probability of K photon arrivals in some fixed interval of time Δt is given by

$$\Pr(K \text{ photon arrivals}) = f(K;\lambda) \tag{2.4}$$

where λ is the mean rate in units of photons per Δt time interval.

2.2.2. Pixel Spike Probability

A pixel spikes when its capacitor contains N electrons. This can be viewed as a decimation of the photon arrivals, that is, for every N photons that arrive one spike is emitted. The case of no decimation would be when $N = 1$. Regardless, we will assume there is a one-to-one correspondence between a photon arriving at a pixel and an electron entering its capacitor. Thus, if one photon arrives at a pixel, the corresponding capacitor will obtain one electron. However, it is important to note that

$$\Pr(K \text{ pixel spikes}) \neq \sum_{i=KN}^{(K+1)N-1} f(i;\lambda) \tag{2.5}$$

because at the start of the time interval, there may already be some number of electrons in the capacitor. In particular, there could be anywhere between 0 and $N-1$ electrons already in the capacitor. If we suppose that the number of electrons in the capacitor at the beginning of the time interval follows a discrete uniform distribution ranging from 0 to $N-1$, then the PMF for pixel spikes is given by

$$\Pr(K \text{ pixel spikes}) = \frac{1}{N}\sum_{j=0}^{N-1} \sum_{i=KN-j}^{(K+1)N-j-1} f(i;\lambda) \tag{2.6}$$

$$= \frac{1}{N}\sum_{j=0}^{N-1} \Big[F\big((K+1)N-j-1;\lambda\big) - F\big(KN-j-1;\lambda\big)\Big] \tag{2.7}$$

$$\equiv g(K;N,\lambda). \tag{2.8}$$

2.2.3. Pixel Spike Count Simulation

In order to demonstrate equation (2.8), we ran a `sparrow` simulation with a 2×2 FPA. We labeled the pixels according to table 2-1 and used the parameters given in table 2-2. The mean photon arrival rate λ for a pixel is given by the product of the photon base rate, the background likelihood, and the time interval. For example, pixel P has a mean photon arrival rate of $\lambda_P = 15.3$ Hz (photons per second). The capacitor well size is $N = 5$, and the simulation lasted 5 seconds. Since the time interval was 0.001 seconds, this gives 5000 time bins over which to count the number of pixel spikes to obtain the necessary statistics.

Table 2-1. Pixel labels for a 2×2 FPA.

P	Q
R	S

Equation (2.8) generated a theoretical PMF for each pixel, and the simulation yielded empirical PMFs. The result is shown in figure 2-1, and we notice two things. The first is that there is good agreement between the theoretical and empirical PMFs. The second is that there are many theoretical data points corresponding to small probabilities for which there is no empirical data

Table 2-2. Parameters used for the `sparrow` simulation.

photon base rate (Hz)	100000.00
background likelihoods	P: 0.153, Q: 0.224, R: 0.779, S: 0.207
capacitor well size (electrons)	5
time interval (s)	1.00e-03
simulation time (s)	5.00

point. The reason for the dearth of empirical data is that the theoretical probabilities are not zero for any $K \geq 0$, but the probabilities are so small they are not realized in our finite simulation. The theoretical data points, of course, also are not realized for every $K \geq 0$ since there is a limit to computer precision. Overall, however, the close agreement between the theoretical and empirical data points shows that we can predict the pixel spike count outputs.

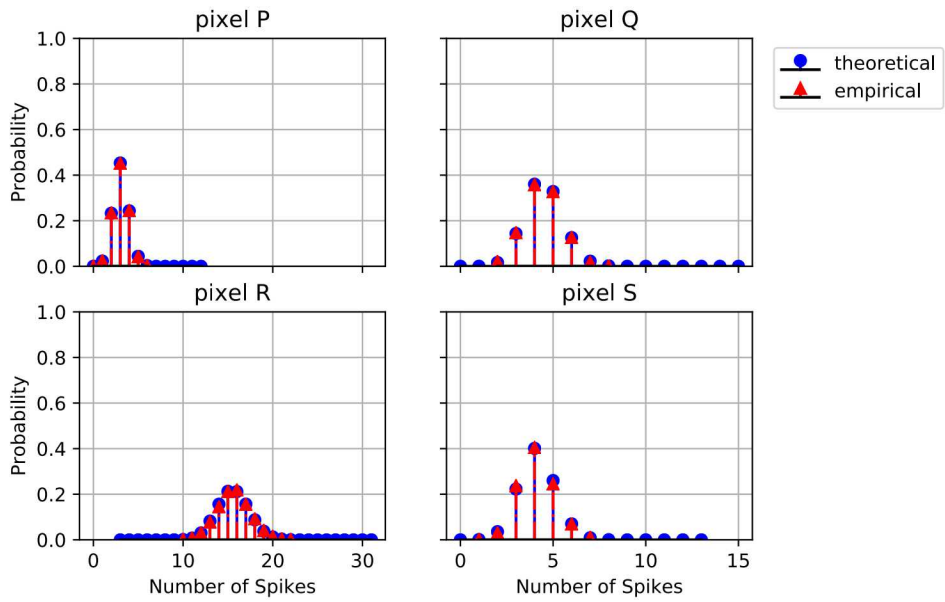

Figure 2-1. The theoretical and empirical PMFs for the probability of pixel spikes.

2.3. HAAR SPIKES

The purpose of including a Haar transform in the signal processing is to spatially whiten an earth background and minimize the decimation required by both the positive and negative quadtrees. Whitening, in this context, means making the spatial power spectral density approximately uniform, that is, the spectrum has the same power at all spatial frequencies and is therefore *white*. The Haar transform itself is a type of discrete wavelet transform (DWT). Specifically, it is the simplest DWT, and we make use of it because its simplicity allows us to implement it in hardware. We use a specialized "throttled" Haar transform, which is a Haar transform where each Haar

element is scaled (or throttled) in such a way that the mean magnitude of each element is roughly equal. This ensures that the background spatial spectrum appears as white noise upon reconstruction. As we will see, this Haar transform is implemented with state machines that only approximate the actual transform, and there are positive and negative spike lines for each Haar element. Hence, we want the number of positive spikes minus the number of negative spikes to be roughly constant across the Haar plane. Further, in order to minimize the decimation that both the positive and negative quadtrees perform, we would like the number of positive and negative spikes to be as small as possible while still providing information about the scene. Ideally, this would mean that for each element, either the positive or the negative line would not contain any spikes in a static scene. That is, there would be no wasted spikes. For instance, if the positive line for a Haar element contains 1000 spikes in some interval of time and the negative line contains 900 spikes, the net number of spikes is 100. This would mean there were 1800 spikes that did not provide any further information. In this case, we would like the positive line to contain 100 spikes and the negative line to contain 0 spikes. Of course, this is not realistic, so the goal is to minimize the number of wasted spikes.

2.3.1. The Haar Transform in a Static Domain

In a static domain (i.e. a still image), a Haar transform is simply the application of a low-pass and high-pass filter to each of the rows and columns of the pixel values. For a simple 2×2 array, this is shown schematically in figure 2-2. Here, the pixel values are P, Q, R, and S. The transformation formulas are given by

$$A = \frac{1}{M_A}\left[(P+Q)+(R+S)\right], \tag{2.9}$$

$$H = \frac{1}{M_H}\left[(P+Q)-(R+S)\right], \tag{2.10}$$

$$V = \frac{1}{M_V}\left[(P-Q)+(R-S)\right], \tag{2.11}$$

$$D = \frac{1}{M_D}\left[(P-Q)-(R-S)\right]. \tag{2.12}$$

The Haar coefficients are often referred to as "approximation", "horizontal", "vertical", and "diagonal," which is the origin of A, H, V, and D.[1] The $1/M$ factor in each coefficient definition is the throttling factor. This will be discussed in more detail later in this section. Finally, figure 2-2 shows how the coefficients are arranged in the Haar plane.

For larger arrays, a Haar transform is similar, but it can be carried out further. Figure 2-3 shows a schematic for a Haar transform on a 4×4 array. To explain, each contiguous 2×2 subarray of the pixel array is transformed according to equations (2.9)–(2.12). The approximation coefficients are put in the top left quadrant, the horizontal coefficients are put in the top right quadrant, the vertical coefficients are put in the bottom left quadrant, and the diagonal coefficients are put in the bottom

[1] Typically, the coefficients are labeled c_A, c_H, c_V, and c_D. For simplicity, we drop that notation here.

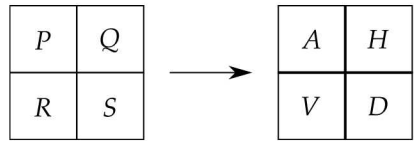

Figure 2-2. Schematic of a Haar transform on a 2×2 array. The pixel array values transform into the Haar coefficients.

right quadrant. To be clear, the first two blocks of figure 2-3 use

$$A_{1,n} = \frac{1}{M_A}\left[(P_n + Q_n) + (R_n + S_n)\right], \tag{2.13}$$

$$H_{1,n} = \frac{1}{M_H}\left[(P_n + Q_n) - (R_n + S_n)\right], \tag{2.14}$$

$$V_{1,n} = \frac{1}{M_V}\left[(P_n - Q_n) + (R_n - S_n)\right], \tag{2.15}$$

$$D_{1,n} = \frac{1}{M_D}\left[(P_n - Q_n) - (R_n - S_n)\right] \tag{2.16}$$

for $n = 0,1,2,3$. The "1" subscript on each Haar coefficient denotes that they are in the first Haar level. Subsequent to the first level of the transform, we can perform a similar operation on the approximation coefficients, which yields the second level of Haar coefficients. In figure 2-3, the block on the right shows the second level of the Haar transform. Only the top-left 2×2 portion is affected. Specifically,

$$A_{2,0} = \frac{1}{M_A}\left[(A_{1,0} + A_{1,1}) + (A_{1,2} + A_{1,3})\right], \tag{2.17}$$

$$H_{2,0} = \frac{1}{M_H}\left[(A_{1,0} + A_{1,1}) - (A_{1,2} + A_{1,3})\right], \tag{2.18}$$

$$V_{2,0} = \frac{1}{M_V}\left[(A_{1,0} - A_{1,1}) + (A_{1,2} - A_{1,3})\right], \tag{2.19}$$

$$D_{2,0} = \frac{1}{M_D}\left[(A_{1,0} - A_{1,1}) - (A_{1,2} - A_{1,3})\right] \tag{2.20}$$

where the "2" subscript denotes the second Haar level.

For larger arrays, this process can continue as long as there is another 2×2 subarray of approximation coefficients. For a pixel array with dimensions of $2^L \times 2^L$, the total number of possible Haar levels is L.

2.3.2. Throttling for a Typical Earth Scene

The DWT naturally puts more of the signal energy in the approximation coefficients. This is usually a desirable feature, but within the SPARR sensor, we want to use the DWT to spatially whiten the signal. If we were using a Fourier transform, this would imply that all Fourier coefficients would have equal magnitude but not necessarily sign. The same is true for the DWT. Hence, we

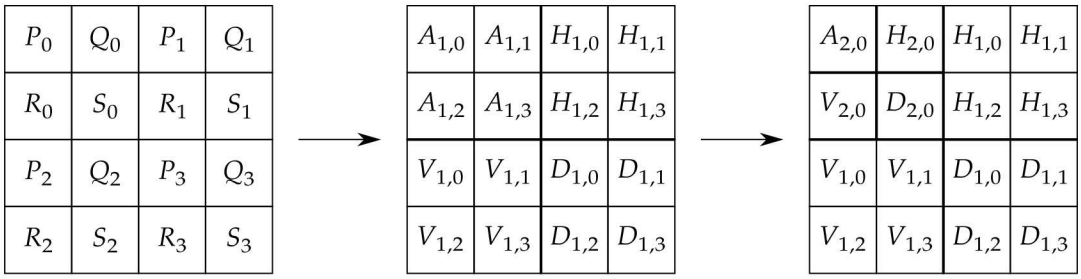

Figure 2-3. Schematic of a Haar transform on a 4×4 array. The pixel array values transform into the Haar coefficients at level 1. The approximation coefficients at level 1 transform into the Haar coefficients at level 2.

want to "throttle" the DWT coefficients so that the magnitudes of all coefficients are roughly the same. By throttle, we mean that we want to mutliply the approximation, horizontal, vertical, and diagonal coefficients by some number between 0 and 1 so that each element at every level is roughly equal in magnitude to all others (i.e. find suitable values for M_A, M_H, M_V, and M_D in equations (2.9)–(2.12)). For SPARR, this is equivalent to decimating the Haar spike outputs. This has the added benefits of using less power (each spike consumes a certain amount of power, so fewer spikes results in less power) and fewer collisions between spikes in the quadtree elements.

To determine the optimal throttling factors for each Haar element, we can create random synthetic earth scenes (see Kern [15]). For each scene, we can try different combinations of throttling factors and compute the average Haar element magnitude at each level. As stated above, we know that the approximation coefficients will be the largest and will need to be throttled. Hence, we begin by trying different throttling values for the approximation coefficients only. The result of this is shown in figure 2-4 for a 256×256 FPA. Each point is the mean of 1000 randomly generated synthetic earth scenes with 95% confidence intervals. The confidence intervals are too small to see at this scale. Different approximation throttling values are used for each plot. The throttling values are shown in pairs where the first value is the multiplier on each approximation coefficient as the DWT is computed. The second number is the additional apex approximation throttling, which will be discussed shortly. The throttling values for the horizontal, vertical, and diagonal detail coefficients are all unity.

Figure 2-4 shows that the the approximation coefficient throttling should be 0.25, which corresponds to a 4:1 decimation for SPARR. However, this does not exactly give us what we want because the apex approximation coefficient is still too large. Since it is only a single coefficient, it does not add much hardware to do an additional throttling on the apex approximation coefficient. We performed a similar test to the one described above to determine the optimal apex throttling value. The result of this is shown in figure 2-5. Again, each point is the mean of 1000 randomly generated synthetic earth scenes with 95% confidence intervals, some of which are visible this time. The approximation throttling value is held constant at 0.25, but the apex throttling is varied (the horizonal, vertical, and diagonal throttling values are still unity). The apex throttling can be viewed as an additional throttling on the final approximation coefficient. The multiplier on the final approximation coefficient is the product of the normal approximation factor and the apex

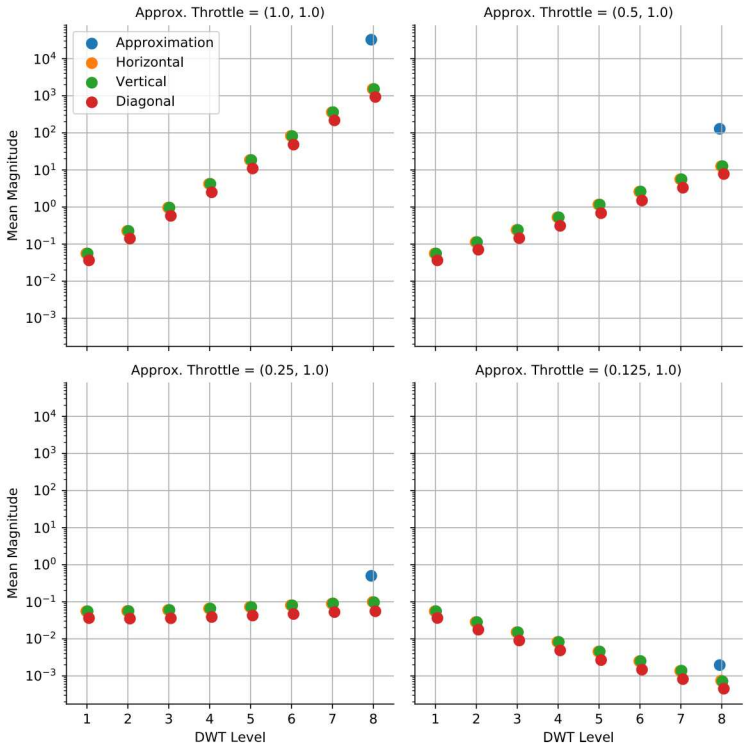

Figure 2-4. The mean coefficient magnitude for each Haar element and DWT level is shown for a 256×256 FPA for various approximation throttling values. See the text for a complete description.

factor.

Figure 2-5 shows two things. The first is that an additional apex throttling of 0.125 (a total of 0.03125 on the final approximation coefficient) is optimal. For SPARR, this is an additional 8:1 decimation for a total of 32:1 decimation on the final approximation coefficient (all other approximation coefficients only have the normal 4:1 decimation). The second thing to notice is that there is some disparity between the diagonal coefficient mean magnitudes and the horizontal and vertical coefficient mean magnitudes. The horizontal and vertical coefficient mean magnitudes are roughly equal, so the throttling value used on one should be used on the other. Another experiment was run to determine this throttling value. The results are shown in figure 2-6. We see that the optimal horizontal/vertical throttling value is 0.5. This corresponds to a 2:1 decimation for SPARR.

All of the above experiments were done assuming the FPA was 256×256. To see whether this scales to other FPA sizes, we used the throttling values determined above (0.25 approximation throttling, 0.125 additional apex throttling, 0.5 horizontal throttling, and 0.5 vertical throttling) on FPAs with sizes 16×16, 32×32, 64×64, and 128×128. This is shown in figure 2-7. We see that the optimal throttling values are invariant under these FPA sizes.

This means that we can update the formulas for the Haar transform. For all DWT levels except for

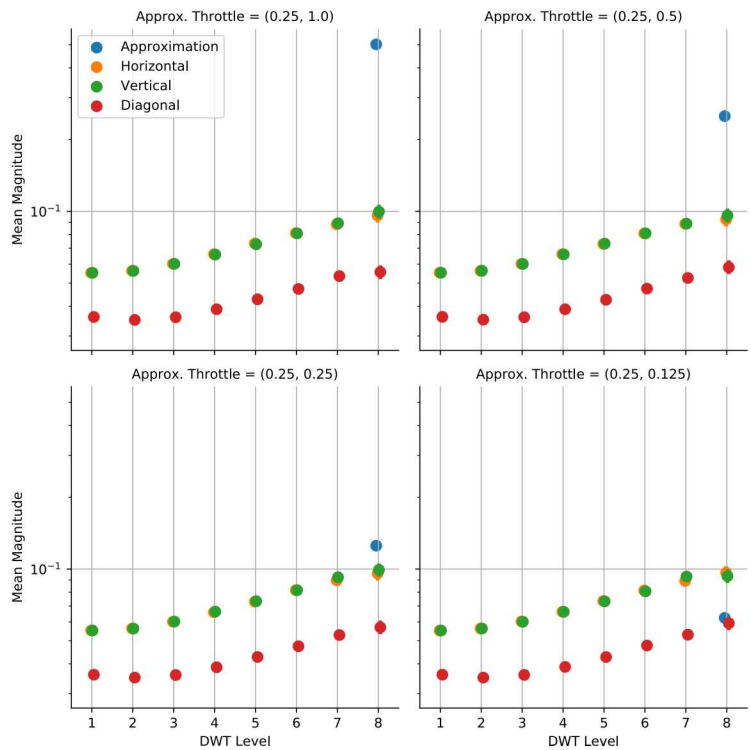

Figure 2-5. The mean coefficient magnitude for each Haar element and DWT level is shown for a 256×256 FPA for various apex throttling values. See the text for a complete description.

the apex, it is given by

$$A = \frac{1}{4}\left[(P+Q)+(R+S)\right], \tag{2.21}$$

$$H = \frac{1}{2}\left[(P+Q)-(R+S)\right], \tag{2.22}$$

$$V = \frac{1}{2}\left[(P-Q)+(R-S)\right], \tag{2.23}$$

$$D = \frac{1}{1}\left[(P-Q)-(R-S)\right]. \tag{2.24}$$

Hence, $M_A = 4$, $M_H = M_V = 2$, and $M_D = 1$. The apex level of the throttled Haar transform contains identical formulas for the horizontal, vertical, and diagonal coefficients. The apex approximation coefficient is given by

$$A_{\text{apex}} = \frac{1}{32}\left[(P+Q)+(R+S)\right], \tag{2.25}$$

which means that $M_{A,\text{apex}} = 32$.

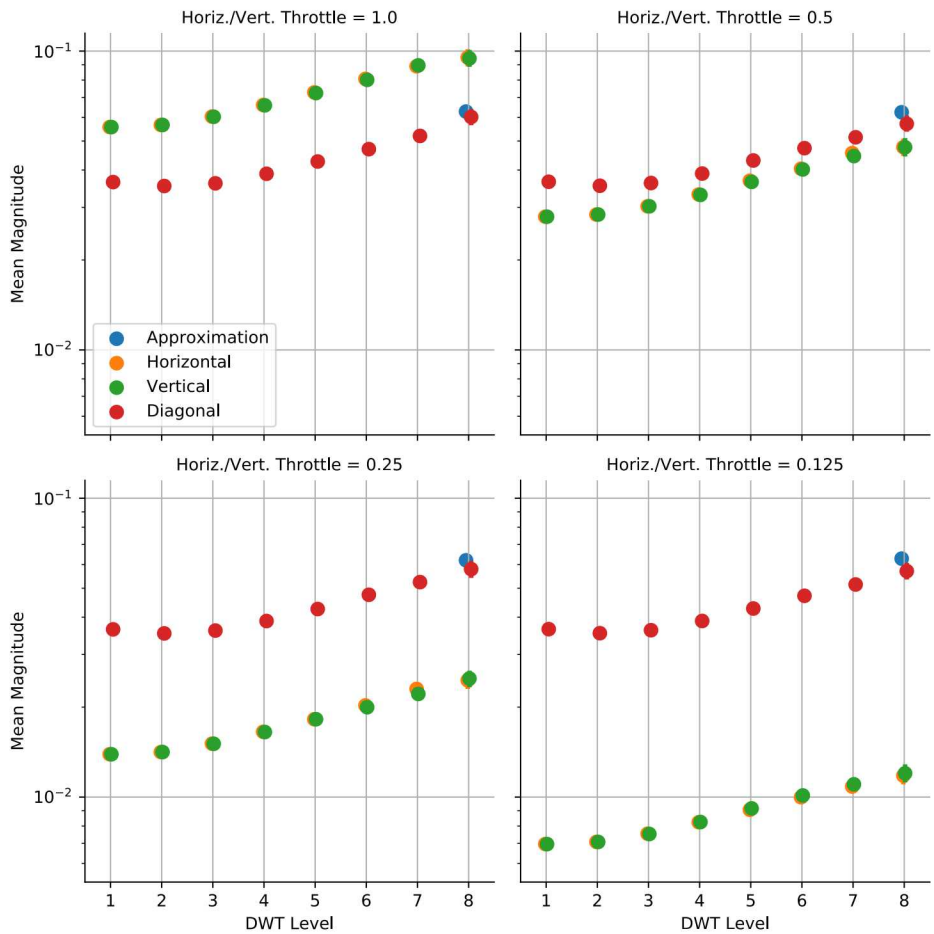

Figure 2-6. The mean coefficient magnitude for each Haar element and DWT level is shown for a 256×256 FPA for various horizontal/vertical throttling values.

2.3.3. The Haar Transform in a Dynamic Domain

All of the analysis thus far assumes that the throttled Haar transform used by SPARR conforms to the definition given above. However, we do not have a static image. Instead, each pixel outputs a spike train that depends on the incoming rate of photons. Hence, we cannot directly apply equations (2.21)–(2.25). Instead, we will use state machines to emulate the throttled Haar transform for a dynamic (i.e. spike) domain.

The approximation coefficient state machine is the simplest because it does not have to treat any of the incoming pixel spike trains as negative. That is, according to equation (2.21), four different spike trains get aggregated into a single spike train and only every fourth spike is retained. This results in a single spike train that contains a quarter of the number of spikes as the four input spike trains. This is shown schematically in figure 2-8. Here, the boxes represent the current state of the machine (in this case, it can be in state 0, 1, 2, or 3.) An input spike changes the state of the machine according to the arrows, and if the machine is in state 3, then the state machine outputs a

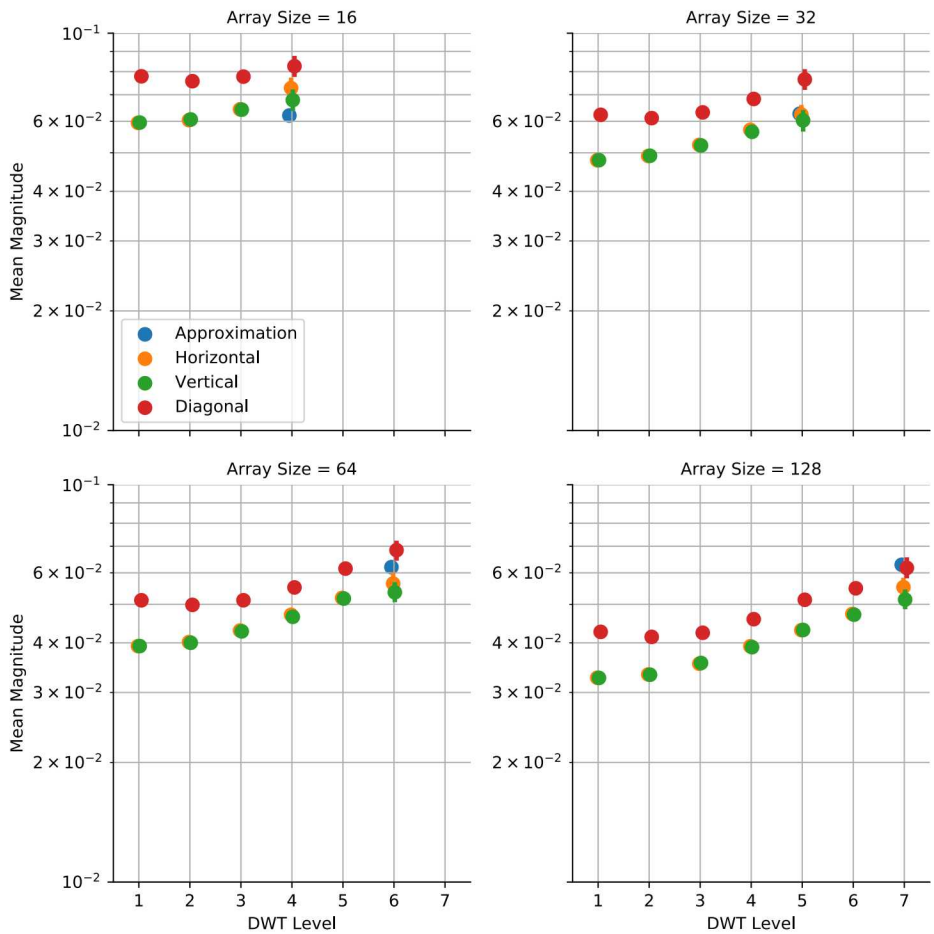

Figure 2-7. The mean coefficient magnitude for each Haar element and DWT level is shown for various FPA sizes. The orange (horizontal) and blue (approximation) dots are mostly hidden beneath the green (vertical) dots.

single spike before it resets to state 0. It should be clear that this machine exactly represents equation (2.21). We also show the state machine corresponding to equation (2.25) in figure 2-9. It is very similar except the number of states has been increased to 32 from 4.

The three detail state machines are more difficult because they involve subtraction. According to equations (2.22)–(2.24), there are two positive input spike trains and two negative input spike trains for each state machine. Since equations (2.22) and (2.23) are the same, the horizontal and vertical state machines are identical. One possibility for this machine is shown in figure 2-10. The diagram is read in much the same way as the approximation state machines. The difference is that there are negative input spikes, which are represented by the dashed lines (the positive input spikes are still represented with the solid lines). A similar diagram for one possibility of the diagonal state machine is shown in figure 2-11. The diagonal state machine is the same as the horizontal and vertical state machines except that there is no decimation. That is, whenever a spike is output for the horizontal and vertical state machines, the state resets either up one for a negative output

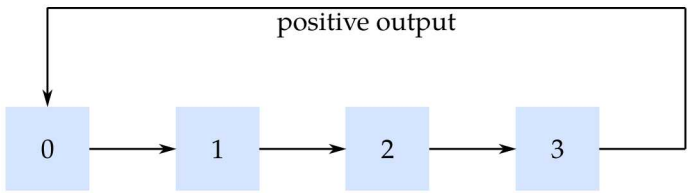

Figure 2-8. The approximation state machine for all DWT levels except the apex.

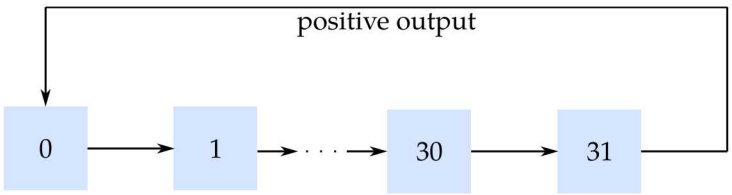

Figure 2-9. The approximation state machine for the apex DWT level.

spike or down one for a positive output spike whereas for the diagonal state machine, the state stays the same upon output spikes. One last thing to point out about both figures 2-10 and 2-11 is that there are both positive and negative output spike trains. In order to find the net number of spikes in some interval of time, you simply subtract the number of negative spikes from the number of positive spikes.

From our analysis in subsection 2.3.2, these state machines should spatially whiten a typical earth background. To see this, we created the synthetic earth scene shown in figure 2-12, and ran it through the state machines given in figures 2-8–2-11 via a `sparrow` simulation. The net output throttled Haar coefficients are given in figure 2-13. The reconstruction is shown in figure 2-14, showing that it is qualitatively spatially whitened.

As stated earlier, whitening the reconstruction is one goal, but we also wish to minimize the number of wasted spikes. In order to determine if this is happening, we plotted the positive and negative throttled Haar outputs separately. These are shown in figures 2-15 and 2-16. In both images, we see the average value is close to zero, which means the number of wasted spikes is small. This reduces the amount of decimation necessary in both the positive and negative quadtrees because fewer spikes need to be processed.

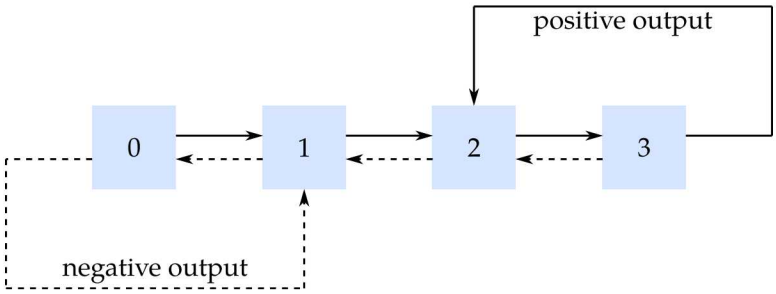

Figure 2-10. The horizontal and vertical state machine for all DWT levels.

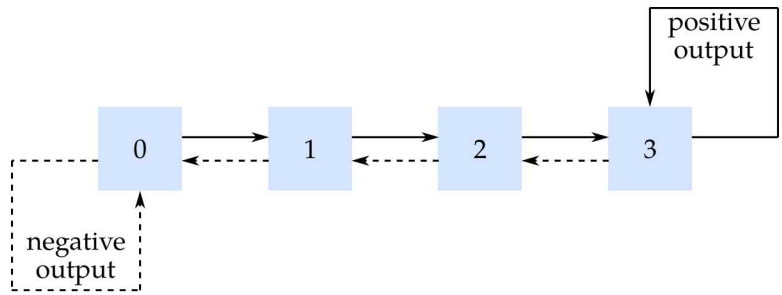

Figure 2-11. The diagonal state machine for all DWT levels.

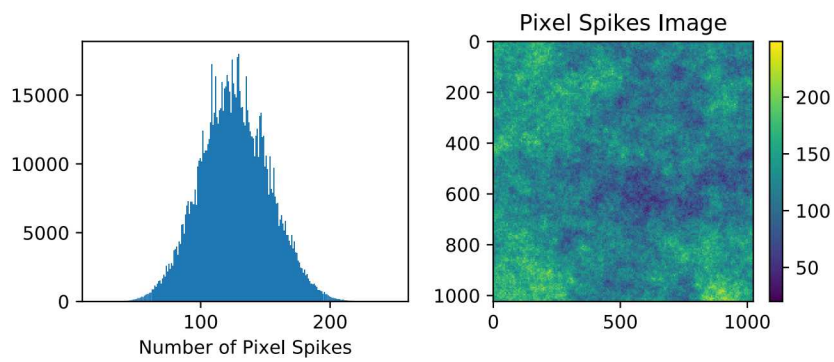

Figure 2-12. A synthetic earth scene based on a Kern spectrum and a histogram for the number of spikes in a given pixel over some interval of time. The right-hand image is in units of pixels and represents a 1024×1024 pixel image.

2.3.4. Haar Net Spike Probability

As explained in subsection 2.3.1, the throttled Haar transform initially deals with 2×2 blocks of pixels. The transform is given by equations (2.9)–(2.12), where P, Q, R, and S are interpreted as pixel values in some 2×2 block of the array. However, since we are not in a static domain, we have to interpret them as the number of spikes out of those same pixels in some interval of time Δt. Then, A, H, V, and D are interpreted as the net number of spikes out of the approximation, horizontal, vertical, and diagonal Haar elements in the same interval of time.

These equations, however, cannot be used as is because we are dealing with probability distributions. That is, for pixels P, Q, R, and S, equation (2.8) gives us

$$\Pr(K \text{ pixel spikes from pixel P}) = g(K;N,\lambda_P) \tag{2.26}$$
$$= g_P(K), \tag{2.27}$$
$$\Pr(K \text{ pixel spikes from pixel Q}) = g(K;N,\lambda_Q) \tag{2.28}$$
$$= g_Q(K), \tag{2.29}$$
$$\Pr(K \text{ pixel spikes from pixel R}) = g(K;N,\lambda_R) \tag{2.30}$$
$$= g_R(K), \tag{2.31}$$

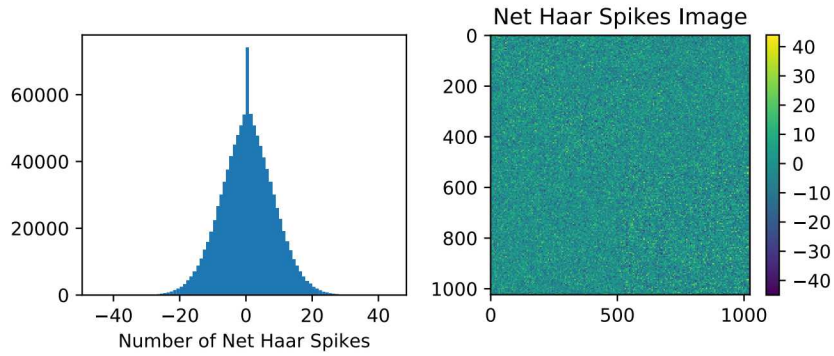

Figure 2-13. The throttled Haar plane from a `sparrow` simulation using the new state machines in figures 2-10 and 2-11 and a histogram for the number of spikes in a given element over some interval of time.

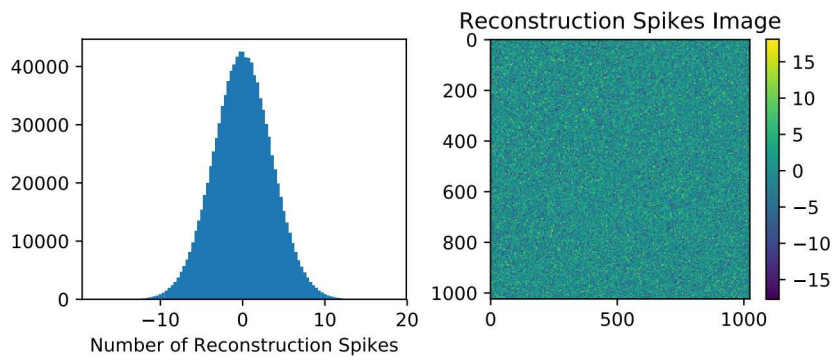

Figure 2-14. The reconstruction of figure 2-13 and a histogram for the number of spikes in a given pixel over some interval of time.

$$\Pr(K \text{ pixel spikes from pixel S}) = g(K; N, \lambda_S) \tag{2.32}$$
$$= g_S(K). \tag{2.33}$$

It is well-known that if X and Y are independent random variables with PMFs $f_X(n)$ and $f_Y(n)$, then the PMF for $X+Y$ is given by the convolution

$$f_{X+Y}(n) = (f_X * f_Y)(n) \tag{2.34}$$
$$= \sum_{m=-\infty}^{\infty} f_X(m) f_Y(n-m). \tag{2.35}$$

Similarly, it is well-known that the PMF for $X-Y$ is given by the cross-correlation[2]

$$f_{X-Y}(n) = (f_Y \star f_X)(n) \tag{2.36}$$

[2]Equation (2.37) only holds if f_X is real-valued. Since this is true in the case of pixel spikes, we will use (2.37) as the definition of the difference of independent random variables. In general, cross-correlation uses the complex conjugate of f_X in its definition.

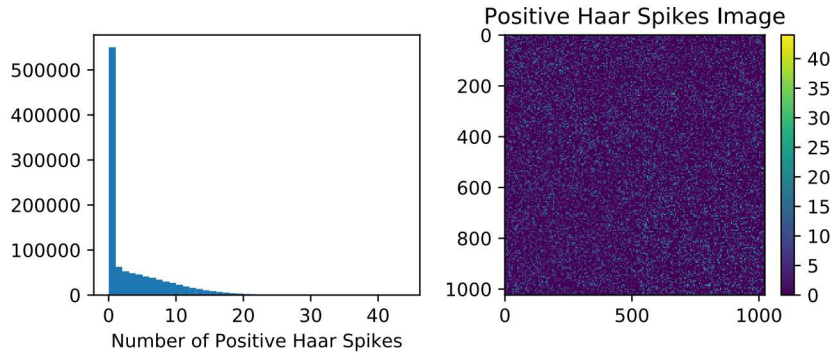

Figure 2-15. Positive portion of the throttled Haar plane from a `sparrow` simulation.

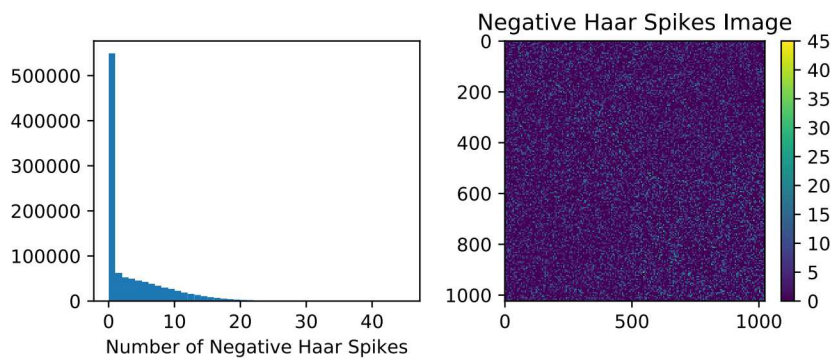

Figure 2-16. The negative portion of the throttled Haar plane from a `sparrow` simulation.

$$= \sum_{m=-\infty}^{\infty} f_Y(m) f_X(m+n). \tag{2.37}$$

Let the reader note the difference in symbols for convolution ($*$) and cross-correlation ($\star$) as they are easy to confuse.

In equation (2.9), the factor $[(P+Q)+(R+S)]$ translates to

$$g_A(K) = \big((g_P * g_Q) * (g_R * g_S)\big)(K), \tag{2.38}$$

which is the probability of K approximation spikes prior to the throttling decimation. To include the throttling, we use logic similar to what was done in section 2.2.2. We find

$$\Pr(K \text{ Haar approx. spikes}) = \frac{1}{M_A} \sum_{j=0}^{M_A-1} \sum_{i=KM_A-j}^{(K+1)M_A-j-1} \big((g_P * g_Q) * (g_R * g_S)\big)(i) \tag{2.39}$$

$$= \frac{1}{M_A} \sum_{j=0}^{M_A-1} \sum_{i=KM_A-j}^{(K+1)M_A-j-1} g_A(i) \tag{2.40}$$

$$= \frac{1}{M_A} \sum_{j=0}^{M_A-1} \Big[G_A\big((K+1)M_A - j - 1\big) - G_A\big(KM_A - j - 1\big) \Big] \tag{2.41}$$

$$\equiv h_A(K) \tag{2.42}$$

where $G_A(K)$ is the CMF of $g_A(K)$ and is given by

$$G_A(K) = \sum_{n=0}^{K} g_A(n). \tag{2.43}$$

Finding the PMFs for the horizontal, vertical, and diagonal spikes is similar to the PMF for the approximation spikes, so we simply give the result. It is

$$\begin{aligned}
\Pr(K \text{ net Haar horiz. spikes}) &= \frac{1}{M_H} \sum_{j=0}^{M_H-1} \sum_{i=KM_H-j}^{(K+1)M_H-j-1} \big((g_R * g_S) \star (g_P * g_Q)\big)(i) && (2.44)\\
&= \frac{1}{M_H} \sum_{j=0}^{M_H-1} \sum_{i=KM_H-j}^{(K+1)M_H-j-1} g_H(i) && (2.45)\\
&= \frac{1}{M_H} \sum_{j=0}^{M_H-1} \Big[G_H\big((K+1)M_H - j - 1\big) - G_H\big(KM_H - j - 1\big)\Big] && (2.46)\\
&\equiv h_H(K), && (2.47)\\
\Pr(K \text{ net Haar vert. spikes}) &= \frac{1}{M_V} \sum_{j=0}^{M_V-1} \sum_{i=KM_V-j}^{(K+1)M_V-j-1} \big((g_Q \star g_P) * (g_S \star g_R)\big)(i) && (2.48)\\
&= \frac{1}{M_V} \sum_{j=0}^{M_V-1} \sum_{i=KM_V-j}^{(K+1)M_V-j-1} g_V(i) && (2.49)\\
&= \frac{1}{M_V} \sum_{j=0}^{M_V-1} \Big[G_V\big((K+1)M_V - j - 1\big) - G_V\big(KM_V - j - 1\big)\Big] && (2.50)\\
&\equiv h_V(K), && (2.51)\\
\Pr(K \text{ net Haar diag. spikes}) &= \frac{1}{M_D} \sum_{j=0}^{M_D-1} \sum_{i=KM_D-j}^{(K+1)M_D-j-1} \big((g_S \star g_R) \star (g_Q \star g_P)\big)(i) && (2.52)\\
&= \frac{1}{M_D} \sum_{j=0}^{M_D-1} \sum_{i=KM_D-j}^{(K+1)M_D-j-1} g_D(i) && (2.53)\\
&= \frac{1}{M_D} \sum_{j=0}^{M_D-1} \Big[G_D\big((K+1)M_D - j - 1\big) - G_D\big(KM_D - j - 1\big)\Big] && (2.54)\\
&\equiv h_D(K). && (2.55)
\end{aligned}$$

Up until this point, the number of spikes has always been positive. However, a standard Haar transform allows for negative coefficients, so we allow K to be negative in equations (2.47), (2.51), and (2.55).

Further levels of the DWT can be treated similarly. The only difference is that the labels P, Q, R, and S now correspond to 2×2 blocks of the Haar approximation coefficients. This process can be repeated up until the apex node where a different value of M_A is needed for the final approximation coefficient.

2.3.5. Haar Net Spike Count Simulation

In order to demonstrate the theoretical equations for the net number of Haar spikes, we counted the Haar outputs from the `sparrow` simulation described in section 2.2.3. That is, equations (2.42), (2.47), (2.51), and (2.55) were used to compute theoretical PMFs and the simulation generated empirical PMFs. The result of doing this is shown in figure 2-17. Not only do we see close agreement between the theoretical and empirical distributions, we also see that the net number of spikes can be negative for some of the Haar outputs. This agreement confirms the theory and analysis shown above.

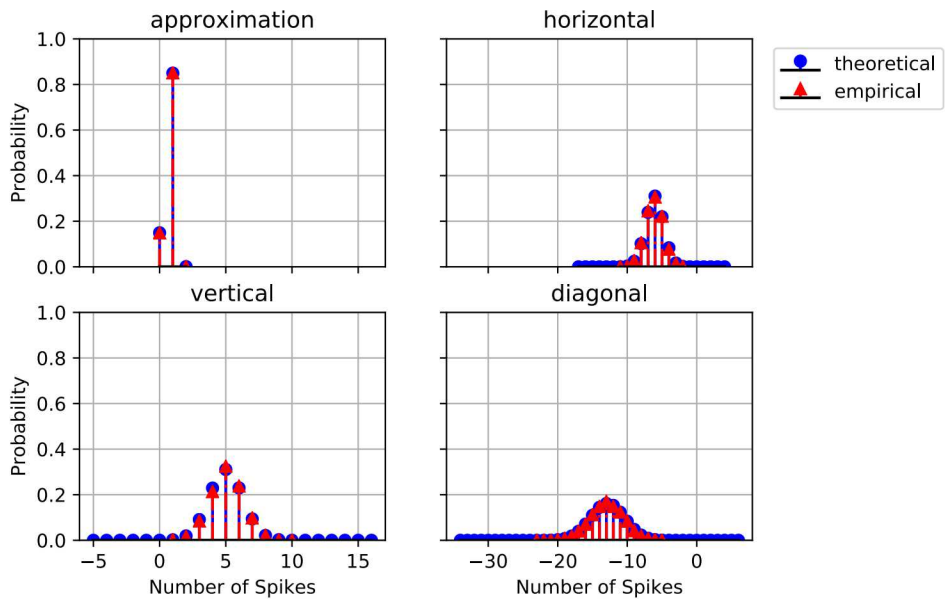

Figure 2-17. The theoretical and empirical PMFs for the probability of net Haar spikes.

2.3.6. Positive and Negative Haar Spike Means

The approximation state machines, shown in figures 2-8 and 2-9, only have positive output spikes. However, the horizontal, vertical, and diagonal state machines, shown in figures 2-10 and 2-11, have both positive and negative output spikes. Equations (2.47), (2.51), and (2.55) give the PMFs for the net number of output spikes during some time interval (i.e. positive minus negative), but SPARR processes the positive and negative spikes separately in the quadtree portion of the signal processing. Hence, we would like to have a way to compute the PMFs for the positive and negative output spikes individually.

This is not a simple task, though. It would be nice if we could apply Markov chain theory to the state machines to find an answer. At first glance it may appear that we could use a continuous time Markov chain (CTMC) to compute the PMFS. However, CTMC requires the inputs to have exponentially distributed waiting times. For SPARR, this is only true in the case where a photon arrival results in a pixel spike (the $N = 1$ case) since the photon arrivals are Poisson distributed (a Poisson distributed process necessarily has exponentially distributed waiting times). We can, however, make use of CTMC theory if we enlarge the state space. To explain, figures 2-10 and 2-11 have four states where the inputs are pixel spikes. If we enlarge the state space by a factor of N^4, then we can use photon arrivals as inputs, which allows us to use CTMC theory.

Let us consider the simplest case of $N = 2$ (i.e. a pixel spike occurs after two photons arrive at a pixel). In order for a photon to cause a transition, each state shown in figures 2-10 and 2-11 (hereafter referred to as major states) enlarges to $N^4 = 16$ minor states, which gives $4N^4 = 64$ total states. Figure 2-18 shows the 16 states for any one of the major states. The states are labeled in binary where the last digit corresponds to the number of P photons received, the second to last digit to the number of Q photons, the third to last digit to the number of R photons, and the fourth to last (i.e. the first) digit to the number of S photons. For example, suppose the figure was for major state 3 of the horizontal state machine, and it was in minor state 0110 (call this state (3,0110)). If the next photon was through pixel P, then the machine would transition to (3,0111). If the next photon was instead through pixel Q, then the machine would yield a positive output spike and transition to state (2,0100). That is, it moves to minor state 0100 within major state 2. If the next photon was instead through pixel R, the machine would transition to state (2,0010). Finally, if the next photon was instead through pixel S, then the machine would transition to state (3,1110). The process can be extended to any positive N. For instance, if we set $N = 3$, then figure 2-18 would need to be enlared to have 81 states for each minor state, and each minor state would be labeled in base 3 instead of base 2.

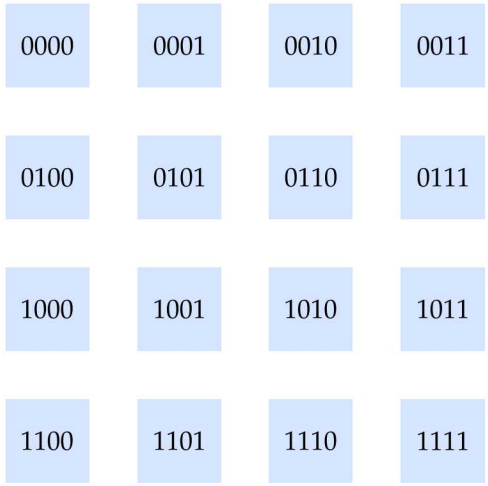

Figure 2-18. The enlarged state space for any of the major states. See the text for a full explanation. The arrows showing transitions are not drawn since there are four transitions out of each state, which would clutter up the drawing.

We can now analyze this enlarged state space with a CTMC. Start by relabeling each state. Above, we labeled a major state of m and minor state of b (b should be converted to decimal) with (m,b). Let the relabeled state be $mN^4 + b$. Using our above example, state (3,0110) becomes state $54 = 3 \cdot 2^4 + 6$ and state (2,0100) becomes state $36 = 2 \cdot 2^4 + 4$. With the relabeled (and ordered) states, we can now populate a transition rate matrix $\mathbf{T}$.[3] Since we know the photon arrival rates for each pixel (λ_P, λ_Q, λ_R, and λ_S), we only need to determine how each state transitions to populate $\mathbf{T}$.

Let us define

$$k = \begin{cases} 0, & \text{photon arrival at pixel P} \\ 1, & \text{photon arrival at pixel Q} \\ 2, & \text{photon arrival at pixel R} \\ 3, & \text{photon arrival at pixel S} \end{cases}, \tag{2.56}$$

$$M = \begin{cases} M_H, & \text{horizontal state machine} \\ M_V, & \text{vertical state machine} \\ M_D, & \text{diagonal state machine} \end{cases}, \tag{2.57}$$

and

$$t(k) = \begin{cases} t_H(k), & \text{horizontal state machine} \\ t_V(k), & \text{vertical state machine} \\ t_D(k), & \text{diagonal state machine} \end{cases} \tag{2.58}$$

depending on which state machine we are considering. Here,

$$t_H(k) = \begin{cases} +1, & k = 0,1 \\ -1, & k = 2,3 \end{cases} \tag{2.59}$$

$$t_V(k) = \begin{cases} +1, & k = 0,2 \\ -1, & k = 1,3 \end{cases} \tag{2.60}$$

$$t_D(k) = \begin{cases} +1, & k = 0,3 \\ -1, & k = 1,2 \end{cases} \tag{2.61}$$

$$\tag{2.62}$$

in accordance with equations (2.10)–(2.12). We now note that when a photon arrives at a pixel, one of three things happens: either the pixel does not produce a spike, the pixel does produce a spike but the state machine does not produce a Haar spike (either positive or negative), or the pixel produces a spike and the state machine also produces a Haar spike (either positive or negative).

The rule for a pixel spike is as follows: if the digit in the N^k's place of b in base N is less than $N-1$, then no pixel spike is produced. By extension, no Haar spike is produced either. In this case, the state advances by N^k. If, instead, the digit in the N^k's place of b in base N is equal to $N-1$, then a pixel spike is produced, but we still need to determine if a Haar spike is produced. Assuming no Haar spike is produced, the state advances by $t(k)N^4 - (N-1)N^k$. At this point, if

[3] The transition rate matrix is often called $\mathbf{Q}$ in the literature, but to avoid confusion with pixel Q, we will call it $\mathbf{T}$.

this new computed state is less than 0, then a negative Haar spike is also produced. If the new computed state instead is greater than or equal to $4N^4$, then a positive Haar spike is produced. Regardless of whether the Haar spike is positive or negative, the state now decreases by $t(k)MN^4$.

This is a bit confusing, so let us consider the example with the horizontal state machine and $N = 2$. Let the current state be denoted X_{current} and let this state be (3,0110) or $X_{\text{current}} = 54$ as determined earlier. If the next photon arrival is through pixel P ($k = 0$), then the digit in the $N^k = 2^0 = 1$'s place of $b = 0110$ (which is already in base $N = 2$) is 0. This is less than $N - 1 = 1$, so no pixel spike is produced. In this case, the state advances by $N^k = 2^0 = 1$, so the next state $X_{\text{next}} = 55$. If the photon had instead arrived at pixel Q ($k = 1$), then the digit in the $N^k = 2^1 = 2$'s place of $b = 0110$ is 1. This is equal to $N - 1 = 1$, so a pixel spike is produced. If we try to advance the state by $t(k)N^4 - (N-1)N^k = +1 \cdot 2^4 - (2-1) \cdot 2^1 = 14$, we end up with $X_{\text{next}} = 68$. However, this is greater than or equal to $4N^4 = 64$, which means we have to decrease the next state by $t(k)MN^4 = +1 \cdot 2 \cdot 2^4 = 32$. This gives $X_{\text{next}} = 36$. We can go through the same procedure to determine that if the next photon arrival was instead through pixel R, then there would be a pixel spike, but no Haar spike. The next state would be 34 in this case. Similarly, if the next photon arrival was instead through pixel S, then there would be no pixel or Haar spike and the next state would be 56.

Returning to the topic of populating the transition rate matrix $\mathbf{T}$, let us define

$$\lambda(k) = \begin{cases} \lambda_P, & k = 0 \\ \lambda_Q, & k = 1 \\ \lambda_R, & k = 2 \\ \lambda_S, & k = 3 \end{cases}. \tag{2.63}$$

Now, if a 0-indexing scheme is used for the elements of $\mathbf{T}$, then the element at row X_{current} and column X_{next} is given by $\lambda(k)$. The diagonal elements are all given by $-(\lambda_P + \lambda_Q + \lambda_R + \lambda_S)$. Hence, there are exactly five non-zero elements in each row of $\mathbf{T}$, and the sum of each row is 0. The algorithm for forming $\mathbf{T}$ is summarized in algorithm 1.

Now that the transition rate matrix is formed, we can compute the limiting probabilities $\mathbf{a}$ by solving

$$\mathbf{a}^\intercal \mathbf{T} = \mathbf{0}^\intercal \text{ subject to } \mathbf{a}^\intercal \mathbf{1} = 1 \tag{2.64}$$

where $\mathbf{0}$ is a vector of zeros and $\mathbf{1}$ is a vector of ones. Numerically, this is equivalent to finding the null space of $\mathbf{T}^\intercal$ and normalizing the result. The null space is simple to find after computing the singular value decomposition. The limiting probabilities are the fraction of time the system spends in each state.

Once the limiting probabilities are found, we can compute an expectation for the number of times a positive or negative Haar spike occurs in some time interval Δt. To do this, multiply λ_P by the sum of the limiting probabilities for states that produce a positive Haar spike from a photon arrival at pixel P. Do the analogous calculation for the other three pixels and add the results together. For the negative Haar spikes, a similar procedure can be used. The result is the mean number of positive and negative Haar spikes in some time interval.

Algorithm 1 Populate the transition rate matrix.

```
T = O
for i ∈ [0,4N^4) do
    T_{i,i} = −(λ_P + λ_Q + λ_R + λ_S)
    for k ∈ [0,4) do
        if (digit in the N^k's place of b_N) ≠ N − 1 then
            j = i + N^k
        else
            j = i + t(k)N^4 − (N − 1)N^k
            if j < 0 or j ≥ 4N^4 then
                j = i − (M − 1)t(k)N^4 − (N − 1)N^k
            end if
        end if
        T_{i,j} = λ(k)
    end for
end for
```

It may be possible to avoid enlarging the state space by using the theory of semi-Markov processes. This is an advanced topic in statistics, but it may provide new insight into the SPARR system if it was understood. It may also be able to provide more than simply the mean for the positive and negative Haar spikes. We have reason to believe that it could also provide the PMFS for the positive and negative Haar spikes just as we can compute the net number of Haar spikes. If, in addition to this, it can also give the probability density functions (PDFS) for the waiting times between positive and negative Haar spikes, then it could provide an alternative method for simulation within `sparrow`. If the calculation could be done efficiently, the speedup could be significant. However, this avenue has not yet been explored, so none of these claims can yet be verified.

2.3.7. Haar Positive and Negative Spike Mean Simulation

To demonstrate that the method in algorithm 1 successfully computes the Haar positive and negative spike means, we again used the `sparrow` simulation described in section 2.2.3 to obtain empirical means. To be specific, we computed the theoretical spike count means for the positive and negative horizontal, vertical, and diagonal state machines using algorithm 1 and obtained corresponding means from the `sparrow` simulation. The results are shown in table 2-3. We see that for each Haar element and each sign, there is agreement between the theoretical mean and the empirical mean. This, again, shows the effectiveness of the above theory in predicting spike outputs.

Table 2-3. The theoretical and empirical positive and negative Haar spike means.

element	sign	theoretical	empirical
horizontal	+	0.010	0.010
horizontal	−	6.094	6.132
vertical	+	5.040	5.091
vertical	−	0.026	0.027
diagonal	+	0.000	0.000
diagonal	−	12.860	12.923

2.4. QUADTREE SPIKES

The two quadtrees put the positive and negative Haar spikes into two spike trains for readout. The quadtrees carry the address of each Haar spike to their apexes, where they add a add a time-tag so that time and location information is not lost. Unfortunately, the quadtree spike statistics require that we separately have the positive and negative Haar spike PMFS. Since we don't currently have that information, the best we can do is specify how to compute the quadtree spike count PMFS if we did.

That is, suppose we had

$$\Pr(K \text{ pos. Haar horiz. spikes}) \equiv h_H^+(K), \tag{2.65}$$
$$\Pr(K \text{ neg. Haar horiz. spikes}) \equiv h_H^-(K), \tag{2.66}$$
$$\Pr(K \text{ pos. Haar vert. spikes}) \equiv h_V^+(K), \tag{2.67}$$
$$\Pr(K \text{ neg. Haar vert. spikes}) \equiv h_V^-(K), \tag{2.68}$$
$$\Pr(K \text{ pos. Haar diag. spikes}) \equiv h_D^+(K), \tag{2.69}$$
$$\Pr(K \text{ neg. Haar diag. spikes}) \equiv h_D^-(K) \tag{2.70}$$

for each level of the Haar transform instead of simply $h_H(K)$, $h_V(K)$, and $h_D(K)$ from equations (2.47), (2.51), and (2.55). Then, for each level of each quadtree, we simply apply the corresponding decimation $M_{Q,\ell}^{\pm}$. Since each level of each quadtree can have its own decimation, it is possible to have $M_{Q,1}^{\pm}$ for level 1, $M_{Q,2}^{\pm}$ for level 2, and so on through level L. If we just consider the positive quadtree, this means in general that $M_{Q,1}^{+} \neq M_{Q,2}^{+} \neq \cdots \neq M_{Q,L}^{+}$ with a similar statement for the negative quadtree. Also note that generally $M_{Q,\ell}^{+} \neq M_{Q,\ell}^{-}$.

For each quadtree, the total decimation is

$$M_Q^+ = \prod_{\ell=1}^{L} M_{Q,\ell}^+, \tag{2.71}$$
$$M_Q^- = \prod_{\ell=1}^{L} M_{Q,\ell}^-, \tag{2.72}$$

and applying this decimation is straightforward. Let $u(K)$ represent any of the PMFS in equations (2.65)–(2.70), and let $U(K)$ be the corresponding CMF. Then,

$$\Pr(K \text{ quadtree spikes}) = \frac{1}{M_Q} \sum_{j=0}^{M_Q-1} \sum_{i=KM_Q-j}^{(K+1)M_Q-j-1} u(i) \tag{2.73}$$

$$= \frac{1}{M_Q} \sum_{j=0}^{M_Q-1} \Big[U\big((K+1)M_Q - j - 1\big) - U\big(KM_Q - j - 1\big) \Big] \tag{2.74}$$

$$\equiv v(K) \tag{2.75}$$

where M_Q is either M_Q^+ or M_Q^-, depending on what $u(K)$ represents. It is important to note that the above equations assume that M_Q is not changing, which is not generally true. The decimation levels dynamically change in each quadtree depending on the input spike rate. However, `sparrow` only adjusts the decimation levels at a fixed period, so within any of these periods, the above analysis holds for the current decimation factors M_Q^+ and M_Q^-.

2.5. CONCLUSION

We have definitively shown how to compute the the PMFS for the number of pixel spikes per time interval for any pixel as well as the number of net Haar spikes for any Haar element and that we can qualitatively spatially whiten the output of the Haar layer through throttling. We showed through simulation that these PMFS are followed very closely in practice, thus providing the theoretical foundation for designing a SPARR system. This means that we learned how to perform signal processing asynchronously on the pixel spikes.

However, the Haar layer gives outputs on a positive line and a negative line, and computing the PMFS for the number spikes per time interval for the individual lines rather than the net result proved more challenging. We were able to compute the theoretical means for the number of spikes per time interval for the first level of the Haar layer only. Further, due to limitations in computing power and time, the pixel capacitors must be limited to $N \leq 25$ or so, which is not realistic. We suspect that analyzing the Haar state machines as a semi-Markov process will yield the sought-after PMFS, but this has not yet been explored.

If we somehow obtain the PMFS for the positive and negative Haar outputs, then we showed that it is fairly straightforward to compute the PMFS for the number of spikes per time interval from the quadtree for each Haar coefficient. The quadtree PMFS are useful for a few reasons. The first is that it gives us a system that is statistically understood, at least in the absence of thermal noise and spike collisions. This allows for better design and faster design iteration. The second reason is that it allows us a way to verify that our `sparrow` simulations are working as expected and may allow for certain optimizations. To be specific, in the same way that `sparrow` can skip generating each photon and instead generate each pixel spike by knowing the theoretical distribution, knowing the Haar and quadtree PMFS may allow us to directly generate those spikes and skip the pixel spike generation processing. Some further work may be needed to statistically understand thermal noise and spike collisions, but it would provide an optimization at least in certain circumstances.

A third reason the quadtree PMFS are useful is that they allow us to characterize the probability of detection and false alarm for signals of interest. That is, if we are interested in signals of a certain brightness, we can design SPARR to have a certain probability of detection and false alarm for that signal, but the quadtree PMFS also allow us to compute the probability of detection and false alarm for signals of any brightness. Of course, there may be unforeseen benefits of accurate quadtree PMFS, but the above reasons are sufficient to continue pursuing these PMFS.

3. CIRCUIT DESIGN

3.1. CIRCUITS INTRODUCTION

In implementing the Haar transform based FPA, there are three main circuit elements to consider in the signal processing chain. The first circuit is the ASP which creates the spike train that will drive the susequent signal processing elements. The ASP will be described in section 3.2.

The second circuit implemented is what we call the Haar node. The Haar node will take the spikes coming from the ASP as inputs and will have several outputs that are representative of the Haar transform of the input spikes. These output spikes will either go into another Haar node for further processing or enter into a readout system.

The total number of spikes being output from the Haar nodes can be significantly higher than what can actually be physically read from the FPA. Due to this, the output will be decimated using a tree structure which we are calling the quadtree. The quadtree is the third circuit element presented.

In section 3.3, the Haar node circuit implementation will be described, and in section 3.4 we will show simulation results and provide a performance summary for the Haar node. For the purpose of reducing power, we present an alternative Haar node implementation design in section 3.5. For the quadtree, the implementation will be described in section 3.6 with simulation and performance results in section 3.7. In section 3.8, we will discuss the full signal processing chain by combining the Haar node and quadtree elements to implement an FPA. The layout of the array will be shown along with simulation results which include effects due to the parasitic resistance and capacitance introduced during the layout. Finally, in section 3.9, we will discuss how a process node can be chosen to achieve the optimum power consumption for the design.

3.2. ASYNCHRONOUS SELF-RESETTING PIXEL

3.2.1. How the Pixel Works

The SPARR concept relies on each individual pixel producing a digital spike train, the frequency of which is proportional to the illumination intensity incident on the pixel. Each pixel operates asynchronously from all others. This function can be achieved by integrating the photo-charge generated by a pixel and resetting this charge when a particular threshold is reached. The reset signal is then used as the spike output from the pixel. This concept is shown schematically in

figure 3-1. For a given illumination flux Φ, the output frequency will depend upon the sensitivity of the input cell as given by

$$f_{\text{out}} = \frac{1}{\dfrac{C_{\text{int}} \cdot V_{\text{thresh}}}{\Phi \cdot A_{\text{pix}} \cdot QE \cdot q} + t_{\text{reset}}} \tag{3.1}$$

where f_{out} is the output frequency, C_{int} is the integration capacitance, V_{thresh} is the comparison voltage at which to reset the integration capacitance, A_{pix} is the pixel area, QE is the pixel quantum efficiency, q is the electronic charge, and t_{reset} is the duration of the reset pulse.

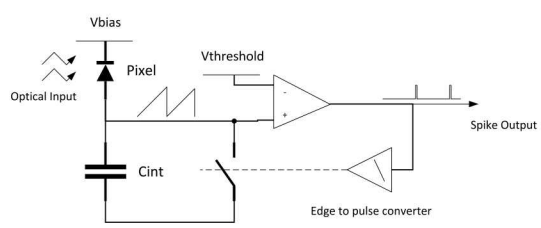

Figure 3-1. asynchronous self-resetting pixel (ASP) conceptual schematic.

The implementation details of any particular pixel design will rely heavily on the desired

characteristics of the FPA. In general, the trade among a completely monolithic FPA, either front-side or back-side illuminated, or a hybrid FPA configuration in which the pixel photosensitive layer is separate from the pixel must first be determined. The hybrid FPA could also be implemented in one of two sub-configurations: (1) The pixel photosensitive layer and pixel ASP spike circuitry are monolithically integrated into the thinned detector layer, or (2) only the pixel photodiodes are implemented in the detector layer – the readout integrated circuit (IC) would implement both the ASP circuitry and the spike processing circuitry. These options are shown in schematic cross-section in figure 3-2, and the merits and drawbacks are discussed below.

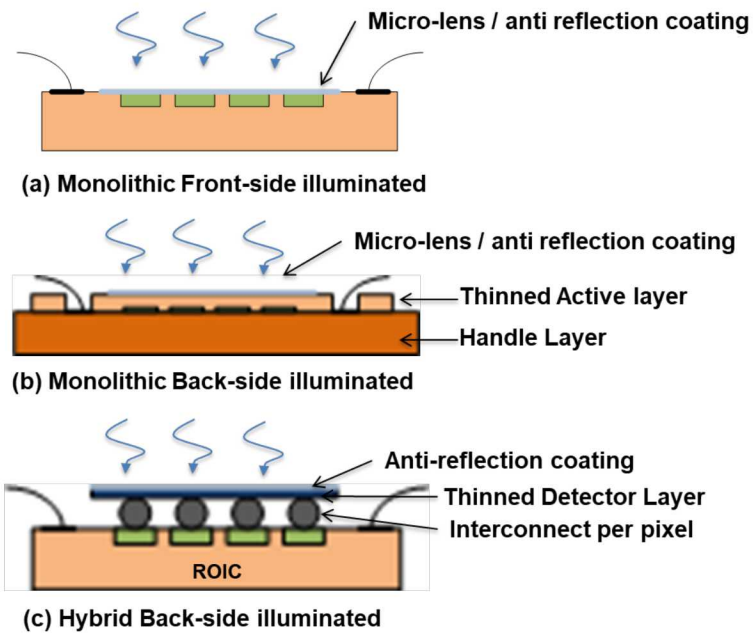

Figure 3-2. Focal plane array configurations.

3.2.2. Monolithic Front-Side Illuminated FPA

Benefits

The monolithic approach is very attractive because the entire FPA could be fabricated as a single chip without the need for 3D integration to any other layer. A monolithic FPA has the potential to be the most sensitive of the configurations because the effective integration capacitance can be kept low due to the fact that the pixel implant can be connected directly to the input comparator directly in the first level metal of the process. If this is a deep sub-micron process, the first level wiring can also be narrow, again minimizing capacitance. This configuration may be able to achieve pixel capacitance as low as a few fempto-farads (fF). Assuming a 100 mV threshold to reset, this would result in a spike output for about every 1000 integrated electrons.

Drawbacks

There are unfortunately a number of drawbacks associated with the purely monolithic configuration. First, the spectral response of the pixel is limited to about 400 nm to 650 nm in

wavelength for a bulk process, or 350 nm to 450 nm for a silicon-on-insulator (SOI) process. The limited spectral response range results from the relatively thin epitaxial layer (about 2 µm thickness) used in the base silicon wafer, or about 200 nm or thinner used in fully depleted SOI processes. Most process nodes at 30 nm or below are implemented on SOI wafers. For a bulk process, the pixel reverse bias will not be sufficient to fully deplete the pixel junction, resulting in carrier diffusion as a contributor to charge transport and collection. This will result in higher pixel cross-talk and slower response times compared with a backside illuminated pixel under high reverse bias. Small process geometry nodes are optimal for SPARR digital signal processing because this will provide improvements in power consumption and allow us to maximize spike rate; however, modern small process nodes are typically SOI which, due to small silicon thickness, are poor absorbers of light. This makes the small process nodes a poor choice for pixel implementation.

The number of metal layers is likely to be 8 or greater on process nodes of 90 nm or smaller. This is good for routing the digital spike processing layers, but it also means that for front-side illumination, the pixel must be illuminated through about 10 µm of silicon dioxide. This can also limit the pixel field of view (FOV) and coupling to an optically fast lens due to the presence of metal traces well above the optically sensitive substrate. It can also result in spectral response oscillation due to interference resulting from reflections from the top and bottom of a dielectric that is many wavelength multiples thick.

Finally, the pixel photosensitive area must share space with ASP circuitry and SPARR spike processing circuitry, limiting the optical fill factor of the pixel. This can be mitigated in part by integration of micro-lenses on the front surface, although this will likely need to be done after wafers are delivered from the silicon foundry. Most deep sub-micron processes, unless specifically designed for imager technology, will not include a micro lens option directly from the manufacturer.

3.2.3. Monolithic Back-Side Illuminated FPA

Benefits

The benefits of the back-side illuminated monolithic FPA are similar to the front-side illuminated configuration discussed above, but with the addition of a higher fill factor and complete control over the dielectric anti-reflection coating thickness and materials. The higher fill factor is a result of no metal lines to obscure optical illumination.

Drawbacks

Back-side configurations require significant process beyond the front-side illuminated architectures. Devices need to be bonded to a handle wafer in order to be thinned. Next, anti-reflective coatings and micro lenses are added. Finally, the bond pads need to be revealed from the backside.

3.2.4. Hybrid FPA Configuration 1

The first hybrid configuration we will consider would implement the photosensitive pixel and immediate ASP circuitry in a layer separate from the spike processing circuitry. The benefits and drawbacks of this approach are discussed below.

Benefits
This approach would allow for a relatively high sensitivity pixel because as with the monolithic configuration, the pixel capacitance can be kept small because the pixel diffusion can be connected to the comparator circuitry using first level metallization. The process used could be an 180 nm mixed-signal process on bulk wafers. The interface to the digital process layer would be the spike train generated by the ASP design. In this way, the digital processing layer could be implemented in a very deep sub-micron node without compromising pixel performance. The fill factor of the pixel could be high in that the photosensitive diffusion needs to only compete for area with the ASP circuitry; all spike processing circuitry would be implemented in the digital processing layer.

Drawbacks
Some of the drawbacks of the monolithic configuration also apply to this implementation. Depending upon the process chosen, the bulk spectral response range of 450 nm to 650 nm may still apply.

With process customization of this layer, it may be possible to implement a low resistivity substrate to increase the reverse bias across the pixel. This would need careful investigation to understand the interactions with CMOS circuitry such as back-gate effect and latch-up.

Of course the complications associated with a hybrid configuration would be realized, such as the need for indium bump or direct bond interface processes to mechanically and electrically connect the two layers.

3.2.5. Hybrid FPA Configuration 2

The second hybrid configuration we will consider would implement the photosensitive pixel only in the pixel hybrid layer. The pixel ASP circuitry and spike processing circuitry would reside on the digital processing layer.

Benefits
This approach allows for the best implementation of pixel photosensitivity. The pixel layer needs only to implement the conversion of photons to electrons, so no consideration of transistor implementation is needed. For a silicon photodiode detector, excellent response can be achieved in the spectral band between 350 nm to 900 nm with 100% fill factor, full depletion, and low cross-talk. Other detector materials such as III-V or II-VI compound semiconductors can be used rather than silicon to extend spectral response into the infrared.

Drawbacks
A hybrid detector configuration's primary drawback is low sensitivity as measured by electrons collected per spike. This is because the parasitic capacitance associated with the metal layers

connecting the pixel threshold circuitry to the top of the readout IC, the top metal indium pad, and indium bump result in a combined capacitance of about 50 fF. With a 100 mV threshold comparison to reset the pixel, about 30,000 electrons are needed. If a simple logic gate is used as the threshold comparison, sensitivity is about 150,000 electrons per spike.

3.2.6. Example ASP Circuit

Figure 3-3 shows an example of a simple ASP circuit comprised of five inverters, a D flip-flop, and two transistors. In a 90 nm process, the circuitry would consume about 30 μm^2 of area. The threshold is determined by the switching point of an inverter, nominally half the power supply voltage (600 mV for 90 nm technology). The threshold can be lowered by increasing the width/length gate ratio of n-channel to p-channel compared to a standard inverter. At the expense of area, power, and maximum frequency of operation, a differential threshold comparator was added to lower the switch point and hence decrease the electrons per spike.

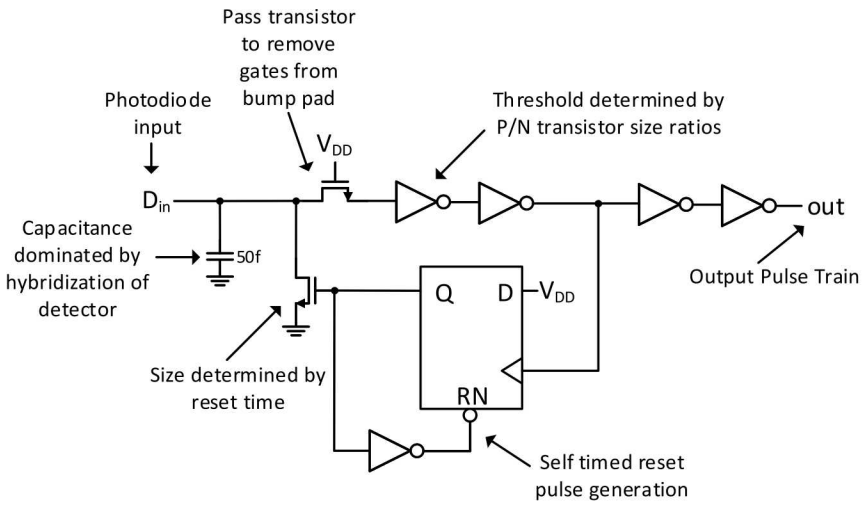

Figure 3-3. Example of a simple asynchronous self-resetting pixel (ASP) circuit.

Figure 3-4 shows the output of a simulation for 3 μA photocurrent feeding the circuit input assuming 50 fF of detector capacitance. The sawtooth waveform displays the integration and resetting functionality of the circuit. The voltage of the spike output is seen in the bottom trace. Figure 3-5 shows a zoom-in of the spike signal around the reset time. The spike output is about 150 ps. The spike width can be increased by adding additional delay between the output of the inverter and the D flip-flop reset pin.

The output frequency and power consumed as a function of input photocurrent are shown in figure 3-6. The output frequency is linear with photocurrent until about 100 MHz output frequency, then it begins to roll over and compress high end signals.

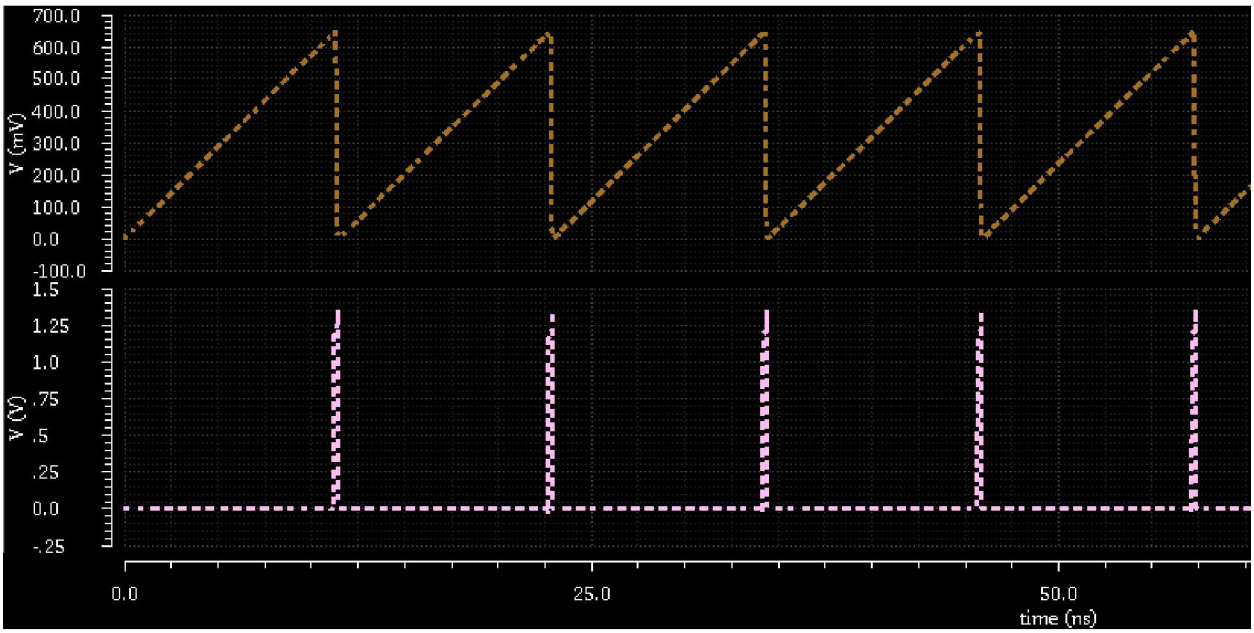

Figure 3-4. Circuit simulation output for the asynchronous self-resetting pixel (ASP) with 3 µA photocurrent. The top plot shows the capacitor voltage increasing as more current is integrated, the bottom plot is the voltage of the spikes emitted by the ASP.

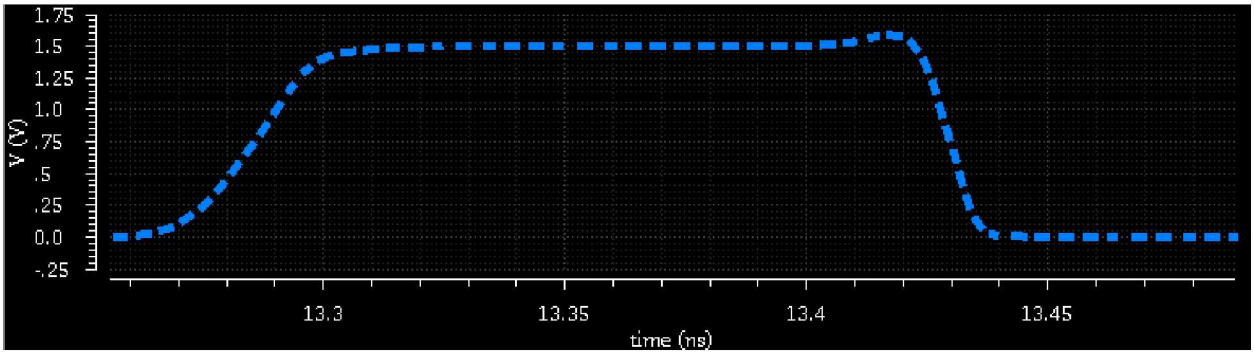

Figure 3-5. Zoom-in of the simulation of the asynchronous self-resetting pixel (ASP) around the pixel reset time.

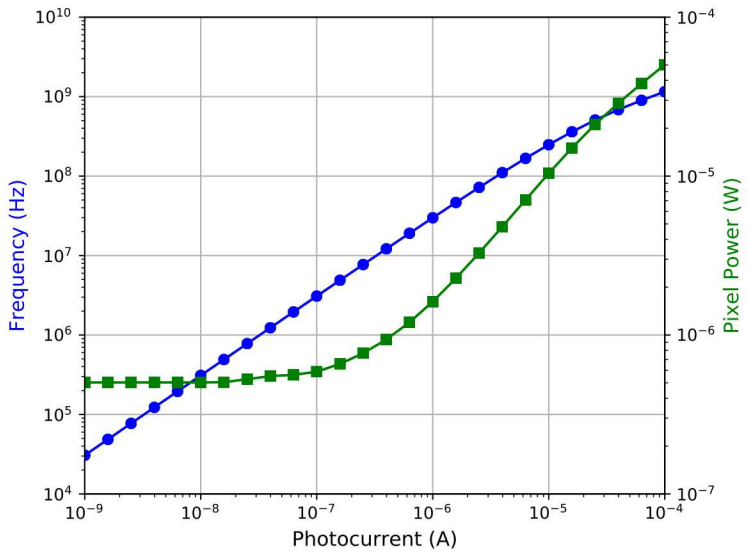

Figure 3-6. Baseline pixel output frequency and power dissipation as a function of input photocurrent.

3.2.7. Fixed Pattern Noise and Manufacturing Variability

Due to manufacturing variability across the FPA, the pulse output frequency for each pixel as a function of illumination level will vary stochastically about a mean response. The response variation for the circuit shown in figure 3-3 as fabricated in Global Foundries' 90 nm process was estimated using a Monte Carlo simulation using foundry provided across chip variations in doping and geometry mismatch. A histogram of frequency response for 100 pixels with expected across-chip process variations at 4 μA of photocurrent is shown in figure 3-7. The ratio of standard deviation to mean frequency is 3.7%.

One method to reduce pixel to pixel variations is to increase the area of the critical frequency determining components. For this design, the input inverter and its associated switching threshold is the critical component and parameter. For the circuit shown in figure 3-3, the input inverter has been increased to four times its original minimum size. The resulting spread in output frequencies has been reduced to 2.4% standard deviation to mean ratio for only a 1 μm^2 increase in area, and an associated increase in low frequency power consumption as seen in figure 3-8 and figure 3-9.

This demonstrates that pixel to pixel output non-uniformity can be decreased by increasing component area and power dissipation and should be implemented. The right trade for pixel circuit complexity for a given application must consider the specific sensitivity, pixel pitch, power requirements, and FPA configuration.

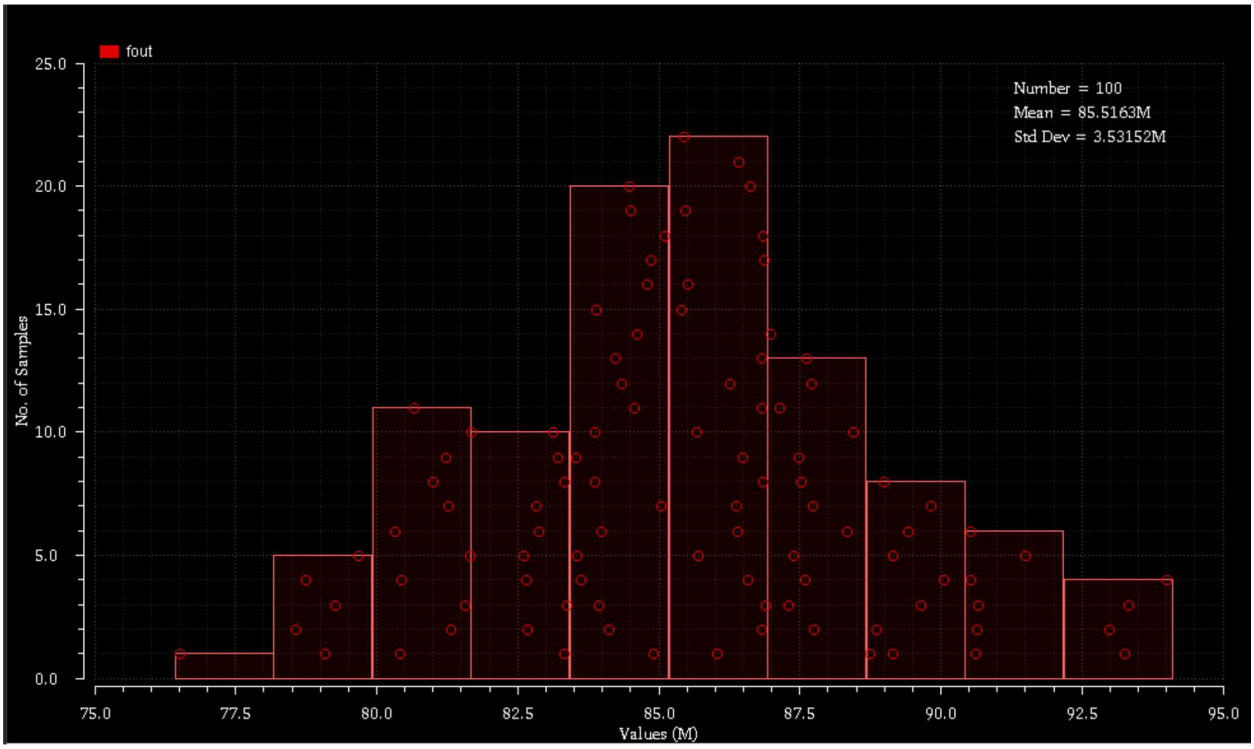

Figure 3-7. Expected variation in output frequency for the baseline pixel due to across-chip manufacturing variations.

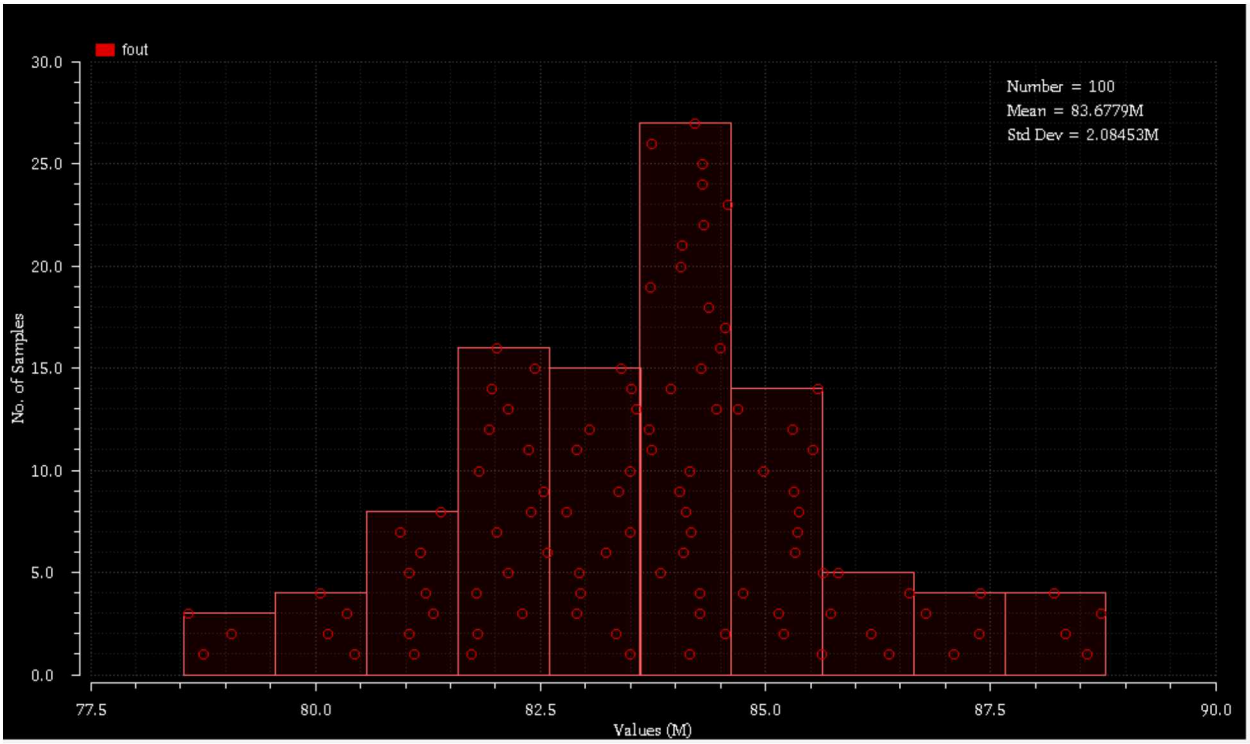

Figure 3-8. Pixel output pulse frequency variation with the input inverter sized to four times the baseline design.

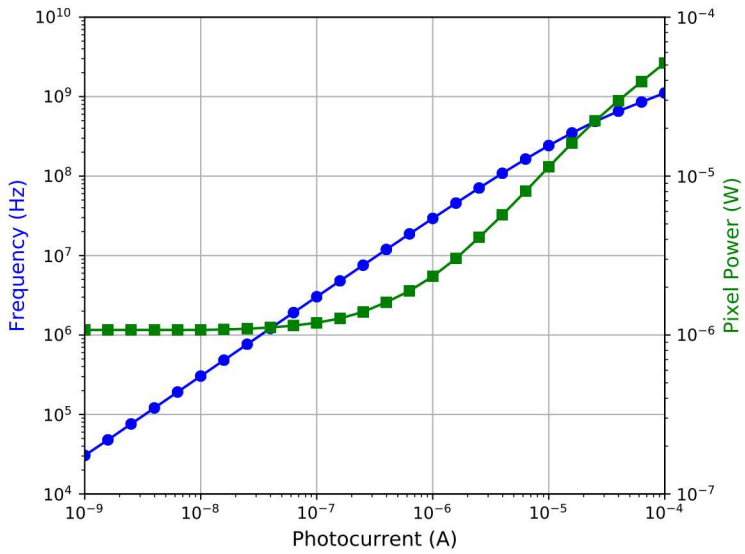

Figure 3-9. Pixel output pulse frequency and power dissipation for the input inverter sized to four times the baseline area.

3.3. HAAR NODE IMPLEMENTATION

In this design, the Haar transform is implemented by processing pixel spike data. At the lowest level in the hierarchy, the spike inputs to the Haar node will be the spikes produced by the ASP. At higher levels in the hierarchy, the Haar nodes will take as inputs one of the outputs of the Haar nodes from a lower level. In all cases, the Haar nodes will operate in a 2×2 array based architecture. For example, the lowest level Haar node will take as its inputs the ASP outputs from a 2×2 set of pixels. Since the input order of the 2×2 array matters in the implementation of the Haar node, it is useful to have a simple nomenclature defining the 2×2 array. We will use the letters P, Q, R, and S to define the inputs to any Haar node as coming from one of the four quadrants in a 2×2 array as shown in table 3-1. There are four Haar transform coefficients that the Haar nodes needs to compute. They are called the approximation, horizontal, vertical, and diagonal coefficients and are respectively denoted c_A, c_H, c_V, and c_D. Next, we will look at each of these coefficients individually and discuss their circuit implementations.

Table 3-1. 2×2 array orientation.

P	Q
R	S

3.3.1. Approximation Coefficient Implementation

The approximation coefficient implementation is the simplest of the four coefficients. This block will simply take each input spike coming from the P, Q, R, and S blocks and output one spike for

every four input spikes, giving a decimation rate of four. This can be implemented according to the block diagram shown in figure 3-10. There are four states that the circuit can operate in: 00, 01, 10, and 11. When the FPA is powered on, a reset signal, RN, can be applied to force the starting state to be 00. On any spike from the P, Q, R, and S inputs, the state machine will move to the next state, following the arrows. When the state machine is in state 01, an input spike will produce an output spike and change the state to 11.

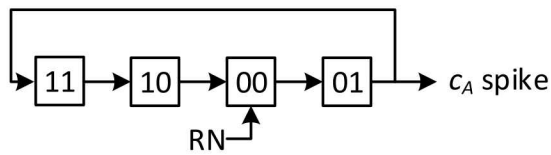

Figure 3-10. Approximation coefficient block diagram.

The state machine operation is summarized in table 3-2. From this table, it can be derived that the next state most significant bit (MSB) can be defined as

$$NS_{MSB} = CS_{LSB} \tag{3.2}$$

while the next state least significant bit (LSB) can be defined as

$$NS_{LSB} = \overline{CS_{MSB}}. \tag{3.3}$$

Table 3-2. Approximation coefficient state machine operation.

Current State (CS)	Next State (NS)	Output
00	01	0
01	11	1
10	00	0
11	10	0

The circuit implementation for the c_A block is shown in figure 3-11. The four-input OR-gate takes the P, Q, R, and S spikes as inputs. Whenever there is a spike on any of these inputs, the OR-gate will output a spike which will be used as a clock signal to operate the state machine. The two D flip-flops hold the current state. The left flip-flop controls the MSB and the right the LSB. Their inputs are the values needed to create the next state according to equations (3.2) and (3.3).

To create the output spike, we use the right flip-flop. When the current state is 01, the output of the AND-gate will be a logic 1. On the next spike input, the Q output of the flip-flop will go high. This signal is buffered and applied to the output. Meanwhile, when the Q signal goes high, the QN signal goes low. This is passed through a delay used to define our spike width. When the output of the delay goes low, the flip-flop is reset – Q and c_A go low and QN goes back to logic high.

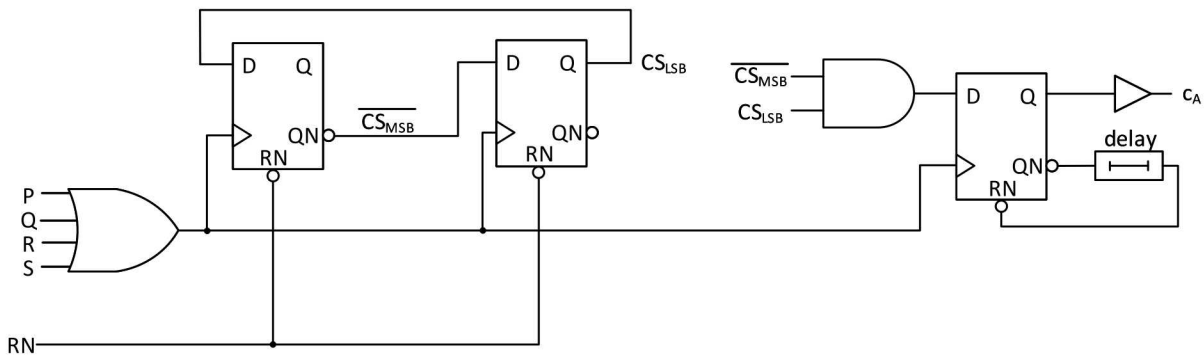

Figure 3-11. Approximation coefficient circuit implementation.

3.3.2. Horizontal and Vertical Coefficient Implementation

The horizontal and vertical coefficients are both implemented with the same circuit topology; the only difference between the two is the order of the inputs. Both will be described here. Figure 3-12 shows the block diagram for the c_H and c_V element. Unlike the c_A element, the c_H and c_V cells have two outputs – a positive and negative spike. Depending on which input has a spike, the state machine will either increment or decrement. If the state machine is in the 01 state and there is an input spike on one of the positive inputs (up), there will be a spike on the positive output and the state machine will then move to the 00 state. Likewise, if the state machine is in the 11 state and there is a spike on one of the negative inputs (down or dn), there will be a spike on the negative output, and the state machine will transition to the 10 state.

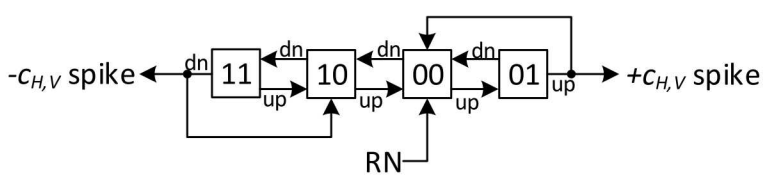

Figure 3-12. Horizontal and vertical coefficient block diagram.

The c_H cell implements the spiking version of

$$c_H = \frac{1}{2}\left[(P+Q)-(R+S)\right] \tag{3.4}$$

while the c_V cell implements

$$c_V = \frac{1}{2}\left[(P-Q)+(R-S)\right]. \tag{3.5}$$

According to equation (3.4), the P and Q inputs will be the positive inputs for the c_H cell which cause the state machine to increment while the R and S inputs are the negative inputs which

decrement the count in the state machine. Similarly, from equation (3.5), P and R increment the c_V state machine while Q and S decrement for the c_V coefficient.

The state machine operation is defined in table 3-3. From this table, it can be derived that the next state MSB can be defined as

$$NS_{MSB} = CS_{MSB} \cdot CS_{LSB} + \overline{\text{in}} \cdot \overline{CS_{LSB}} \tag{3.6}$$

while the next state LSB can be defined as

$$NS_{LSB} = \overline{CS_{MSB}} \cdot \overline{CS_{LSB}} \cdot \text{in} + CS_{MSB} \cdot \overline{CS_{LSB}} \cdot \overline{\text{in}}. \tag{3.7}$$

Table 3-3. Horizontal and vertical state machine operation.

Current State (CS)	Input	Next State (NS)	Output
00	0	10	0
00	1	01	0
01	0	00	0
01	1	00	Positive Spike
10	0	11	0
10	1	00	0
11	0	10	Negative Spike
11	1	10	0

The circuit implementation for the c_H and c_V coefficients is more complicated than that of the c_A coefficient, so it will be explained in smaller pieces. Figure 3-13 shows the first part of the circuit. The OR-gates on the left add the spikes together. If there is a spike on the up OR-gate, there will be a spike on in+. Similarly, if there is a spike on one of the down inputs, there will be a spike on in-. These spikes will then control the SR-latch that follows the input OR-gates. If there is a spike on in+, lat_in+ will be held to a logic high; if there is a spike on in-, lat_in+ will be a logic low. The other output of the SR-latch, lat_in- is the logical complement of lat_in+.

The right half of figure 3-13 shows how the local clock is generated to control the state machine. When there is a spike on in+ or in- (i.e., there's a spike on any input), this OR-gate will create a spike. It is then delayed to allow the state machine to process the next state logic due to lat_in+ and lat_in- potentially changing state. After the delay, there is the single-pulse clock signal, clk, which will be applied to the state machine to load the next state.

The implementation of the state machine is shown in figure 3-14. The logic gates to the left of each flip-flop implement the required next state logic functions given in equations (3.6) and (3.7). The single-pulse clock signal used here, clk, is generated as previously shown in figure 3-13.

The output driver for the c_H and c_V coefficients is similar to that of the c_A coefficient except that there are now two of them as shown in figure 3-15. For the positive output, when the current state is 01, the left AND-gate output will be high. If the state machine is in this state, according to table

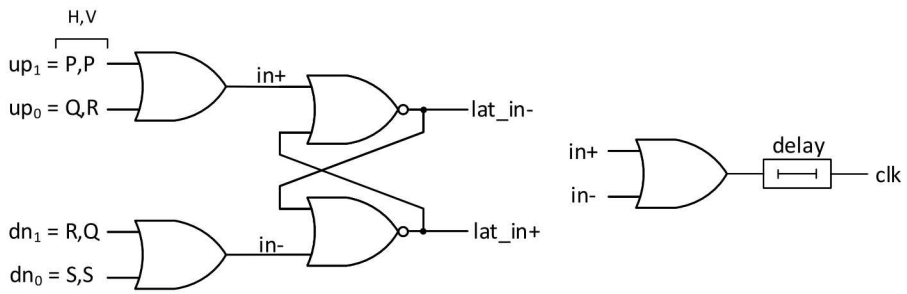

Figure 3-13. Horizontal and vertical coefficient circuit implementation – up/down control and clock generation.

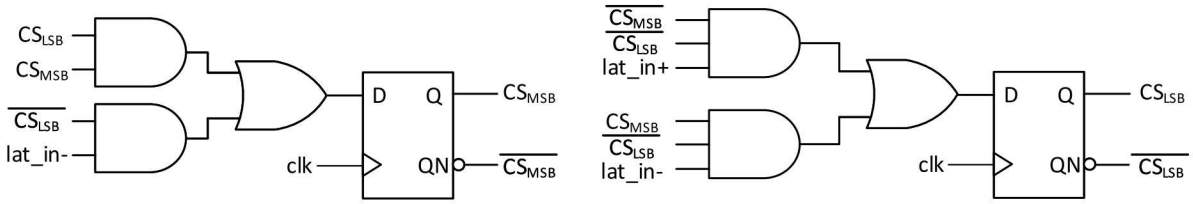

Figure 3-14. Horizontal and vertical coefficient circuit implementation – state machine.

3-3, there should be a positive output spike if the circuit gets a spike on one of the two up-inputs. If this is the case, there will be a spike on in+ which will produce an output spike on out+. The width of the spike is controlled by the delay element in the same fashion as previously described in the c_A section. Similarly, if the state machine is in the 11 state, according to table 3-3 there will be a negative spike either of the two down inputs spikes.

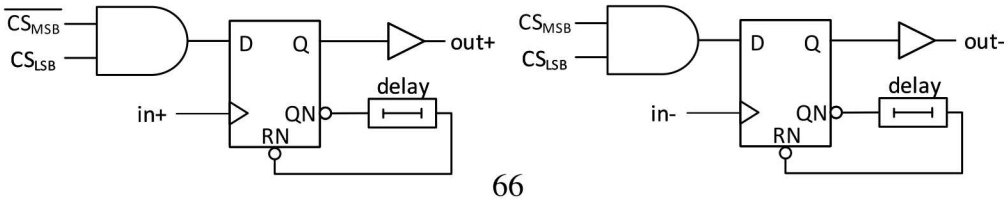

Figure 3-15. Horizontal and vertical coefficient circuit implementation – output pulse driver.

3.3.3. Diagonal Coefficient Implementation

The final element in implementing the Haar transform is the circuit for implementing the diagonal coefficient. The c_D spike output can be approximately described by the equation

$$c_D = (P - Q) - (R - S). \tag{3.8}$$

The c_D implementation is similar to that of the c_H and c_V implementations, and the block diagram of its operation is shown in figure 3-16. The difference between this implementation and that of the c_H and c_V is that the state machine will stay in its current state when an output pulse is generated.

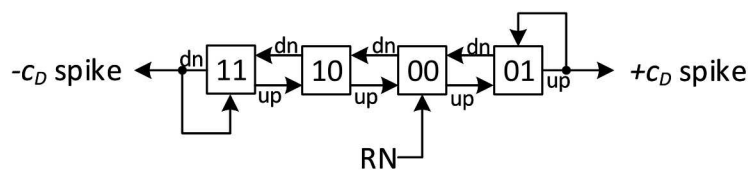

Figure 3-16. Diagonal coefficient block diagram.

The state machine operation for the c_D coefficient is defined in table 3-4. From this table, it can be derived that the next state MSB can be defined as

$$NS_{MSB} = CS_{MSB} \cdot CS_{LSB} + \overline{\text{in}} \cdot \overline{CS_{LSB}} \tag{3.9}$$

while the next state LSB can be defined as

$$NS_{LSB} = \overline{\text{in}} \cdot CS_{MSB} + \text{in} \cdot \overline{CS_{MSB}}. \tag{3.10}$$

Table 3-4. Diagonal coefficient state machine operation.

Current State (CS)	Input	Next State (NS)	Output
00	0	10	0
00	1	01	0
01	0	00	0
01	1	01	Positive Spike
10	0	11	0
10	1	00	0
11	0	11	Negative Spike
11	1	10	0

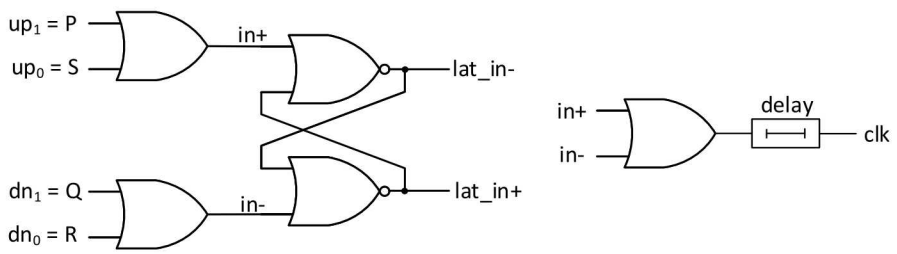

Figure 3-17. Diagonal coefficient circuit implementation – up/down control and clock generation.

The circuit implementation of the c_D element is based on that of the c_H and c_V elements. Figure 3-17 illustrates how the input spikes are converted to control signals similarly to how they were done for the c_H and c_V elements from figure 3-13. The only difference between these two circuits is the order of the P, Q, R, and S inputs.

The circuit implementation for the c_D element's state machine is shown in figure 3-18. Again, the c_D element operates similarly to that of the c_H and c_V state machines; the variation is in the combinational logic prior to the flip-flop inputs.

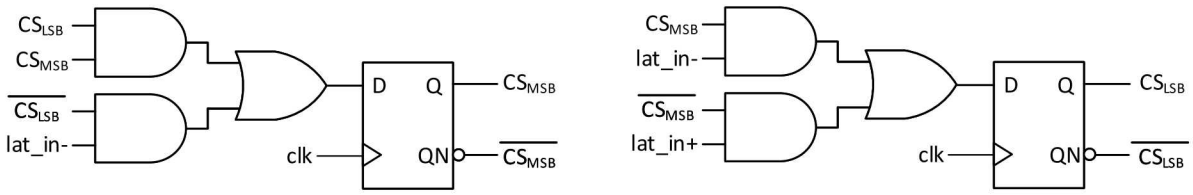

Figure 3-18. Diagonal coefficient circuit implementation – state machine.

The final circuit element is the output driver. For the c_D element, the operation is exactly the same as that of the c_H and c_V element so the same circuit of figure 3-15 is used.

3.4. HAAR NODE PERFORMANCE

Performance, from a circuit perspective, is concerned that the basic function of the circuit follows the behavior of the state tables when spikes are widely separated compared to the pulse width, follows reasonable behavior when spikes are densely spaced in time compared to the pulse width–colliding–and measures of energy consumption due to both quiescent non-spiking energy leakage and due to spike-driven dynamics. Section 3.4.1 demonstrates through simulation that the Haar node circuit elements describe the state table behaviors when spikes are not colliding. Section 3.4.2 shows how the behavior, while not mathematically accurate, is nevertheless stable

and avoids pathological failure in the presence of collision. Energy consumption is addressed by section 3.4.3.

3.4.1. Haar Node Simulations in Normal Operation

The approximation coefficient element was simulated with spikes randomly distributed between the four inputs. Sufficient time was given between spikes to ensure that there would be no issues with overlapping spikes. Figure 3-19 shows the results of this simulation where the circuit starts out in a middle state and there is an output spike after two spikes on the inputs. After this, the circuit emits one spike every four input spikes. This is operating in accordance to the state machine illustrated in figure 3-10.

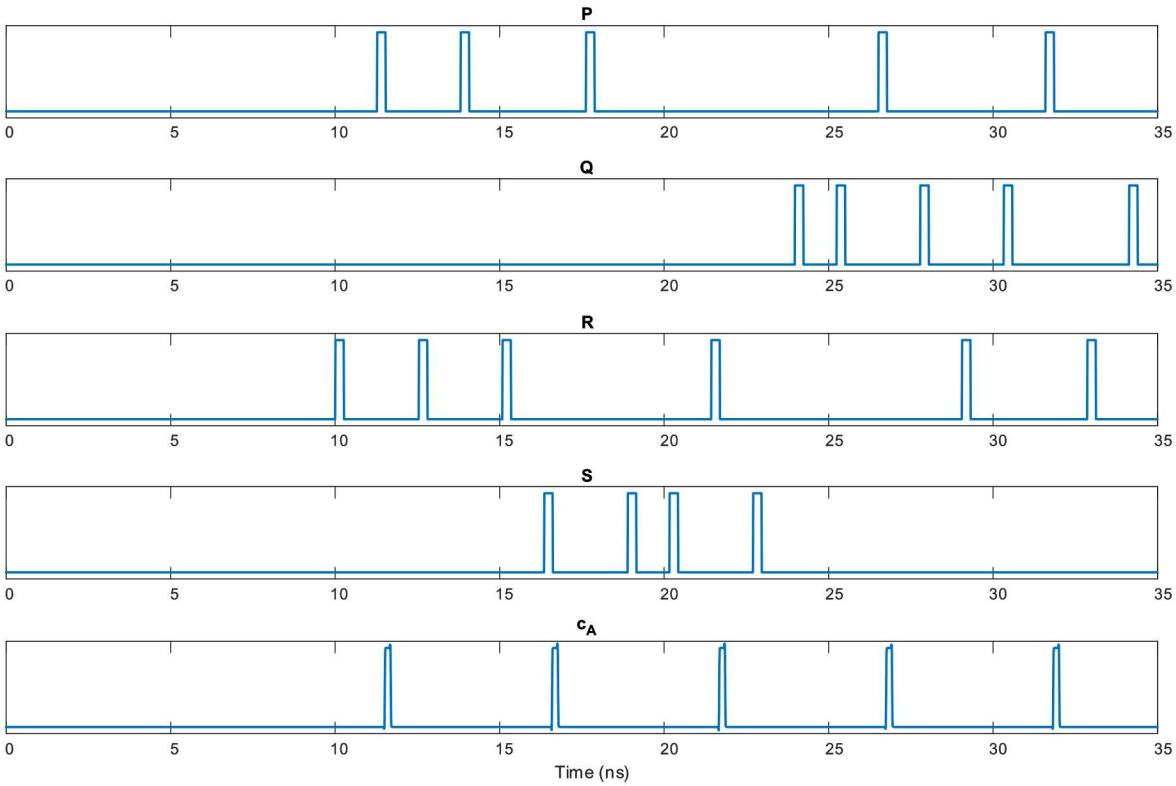

Figure 3-19. Approximation coefficient simulated results. The output c_A is a 4:1 decimation of the inputs once the state settles, around 13 ns

The same set of input spikes used for the c_A element was applied to the c_H element. Figure 3-20 shows a portion of these results. The c_H element has two outputs, c_H+ and c_H- corresponding to the block diagram in figure 3-12. For the c_H element, the P and Q inputs cause the state machine to increment while the R and S inputs cause it to decrement. The circuit starts in the 00 state. The sequence of inputs prior to the first output c_{H-} is R,P,R,P,R,S,P,S,S corresponding to internal state

transitioning 00+R→ 10+P→ 00+R→ 10+P→ 00+R→ 10+S→ 11+P→ 10+S→ 11+S→ 10 and emit a negative spike; the behavior shown by figure 3-20.

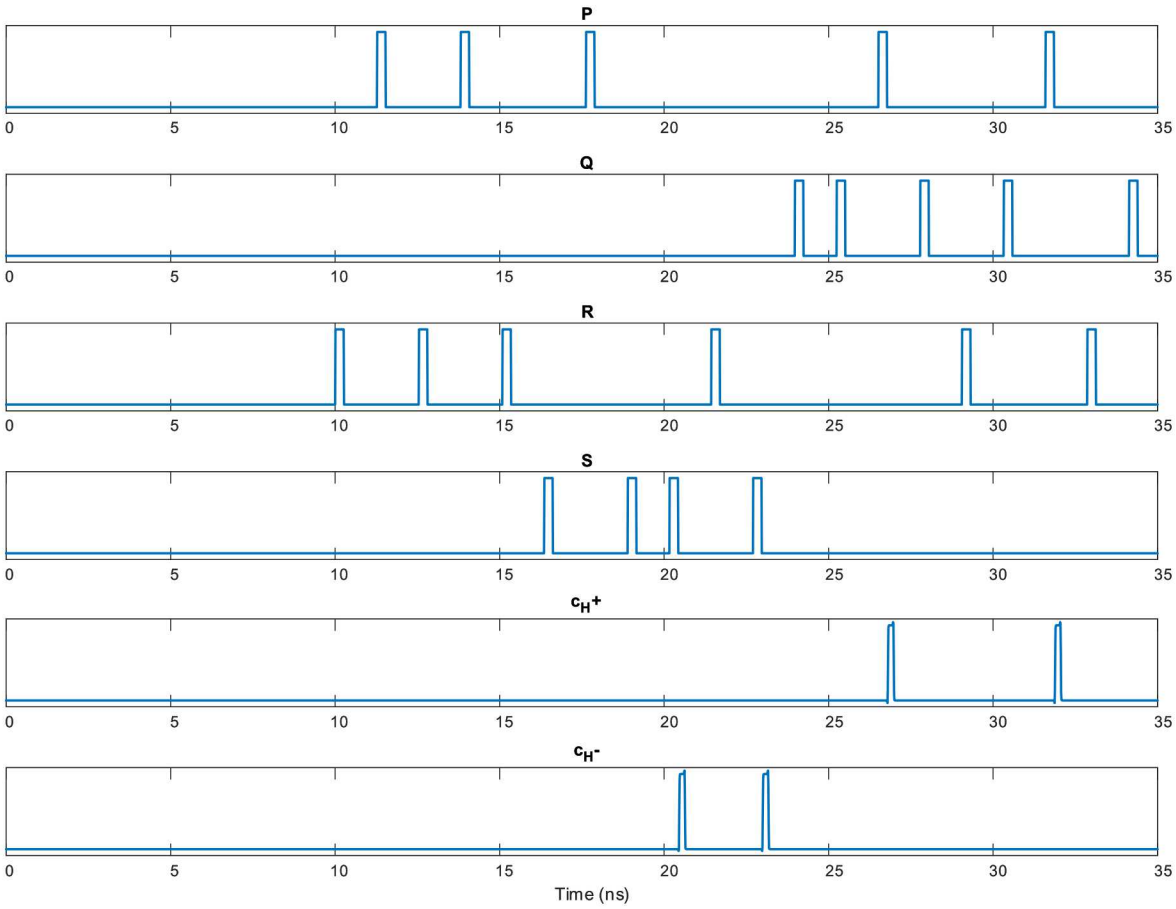

Figure 3-20. Horizontal coefficient simulated results.

The c_V element operating similarly to the c_H element; the only difference is that the P and R inputs cause the state machine to increment while the Q and S inputs cause the state machine to decrement. No figure is shown for this operation, but verification was performed separately on the c_V element to ensure its operation before integrating it into a larger FPA design.

The final block in the Haar node is the c_D element. The simulation results for this block are shown in figure 3-21. It also has two outputs and follows the signal flow in the state machine diagram of figure 3-16. The P and S inputs for this block cause the state machine to increment while the R and Q inputs cause it to decrement. The output spikes occur where expected according to state transition table 3-4

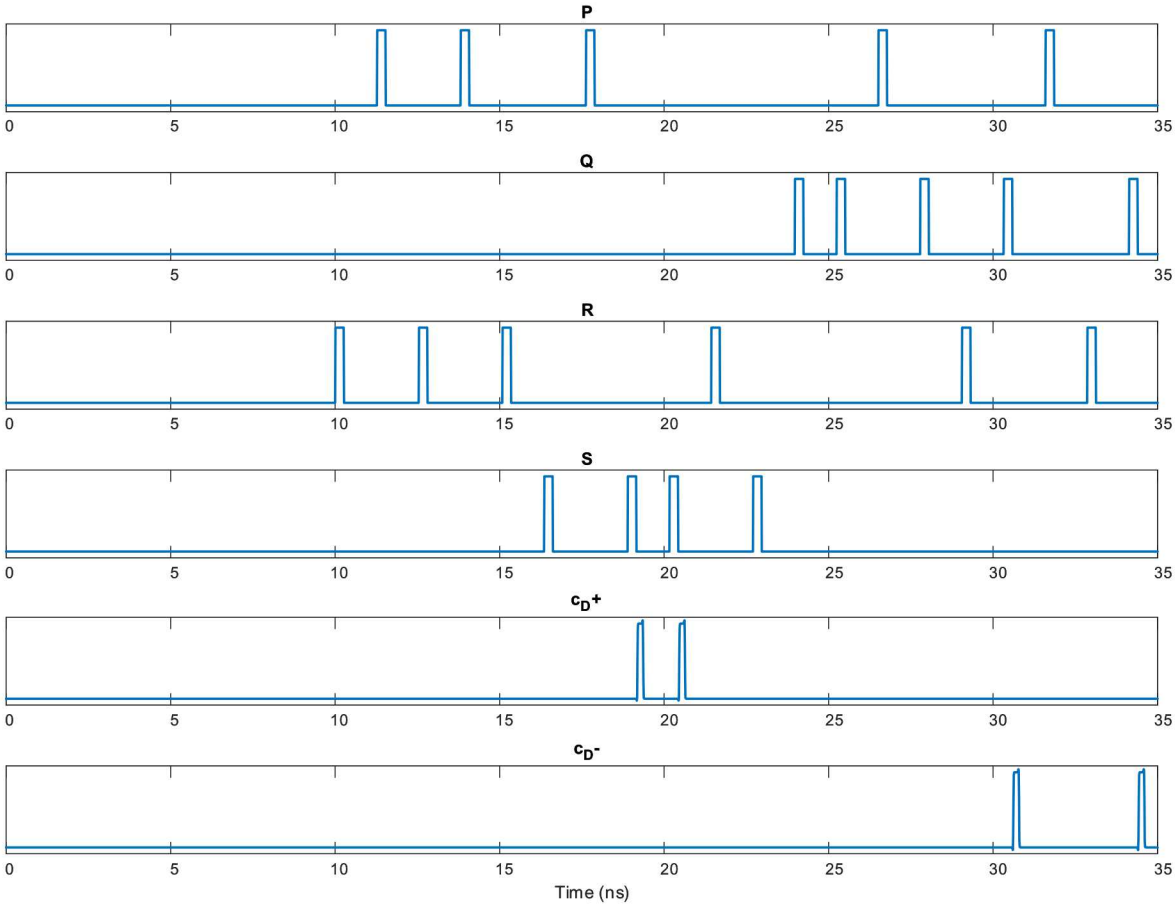

Figure 3-21. Diagonal coefficient simulated results.

3.4.2. Haar Node Simulations in Saturation

The previous simulations show the operation of the circuits in a normal operation mode where there is sufficient time between spikes for all the internal logic to operate as expected. As the spike rate increases, or if spikes become too close together or collide, the circuits will fail to operate as desired. This section describes how the circuit behaves in these undesired operating conditions.

Figure 3-22 shows the Haar node circuit outputs when receiving spikes simultaneously on the P and Q inputs. For this simulation the R and S inputs are grounded and therefore not receiving any spikes. Looking at the c_A output, it can be seen that output spike rate is half of what it should be. In this case, the spikes on the Q input are not being detected due to the spike overlap. Referring back to figure 3-11, the four-input OR gate needs to have a gap between spikes long enough for its output to go low between spikes. When the spikes overlap as they do in figure 3-22, the output of the OR gate is essentially just a single, wide pulse.

The output of the c_H element is having a similar failure to that of the c_A element. The output rate is half of what it should be. With spikes overlapping or too close to each other, the clock generation circuit of figure 3-13 creates only a single clock pulse, not two it would ideally generate.

The outputs of the c_V and c_D cells are as expected. However, this is only coincidence, the circuit is experiencing similar "blurred clock" failure as the previous two elements. The state machine logic is also not working correctly when the spikes are too close. In this case, the failure of the state machine and the failure of the clock negate each other to produce the correct output. That errors cancel to produce the correct behavior is not generally true, it occurs here by good fortune.

In general, there must be sufficient time between spikes for the logic to settle. For the current design implemented in 90 nm CMOS, that means 200 ps between the falling edge of the first spike and the rising edge of the second spike. For spacing closer than this, or for overlapping spikes, the logic circuits will begin to fail and the outputs can no longer be trusted to be representative of their inputs.

3.4.3. Haar Node Energy Consumption

One of the important concerns in this design is the power consumption. This section shows a full analysis of the c_A element to illustrate how a block's power consumption is determined. In our simulation, 100 spikes were applied randomly to the four inputs. Spikes were spaced 1 ns apart to allow all transients to fully settle.

Figure 3-23 shows the integral of the power being consumed by the c_A block. At the 10 ns point, the input spikes begin and the energy (integral of the power) increases with each spike. Once the 100 spikes finish, the energy curve becomes a nearly flat line since there is no switching in the logic circuits without spiking inputs.

Although the curve appears flat after spiking has ceased, it actually has a very small slope due to the direct current (DC) leakage current of the transistors. While the leakage is small, the

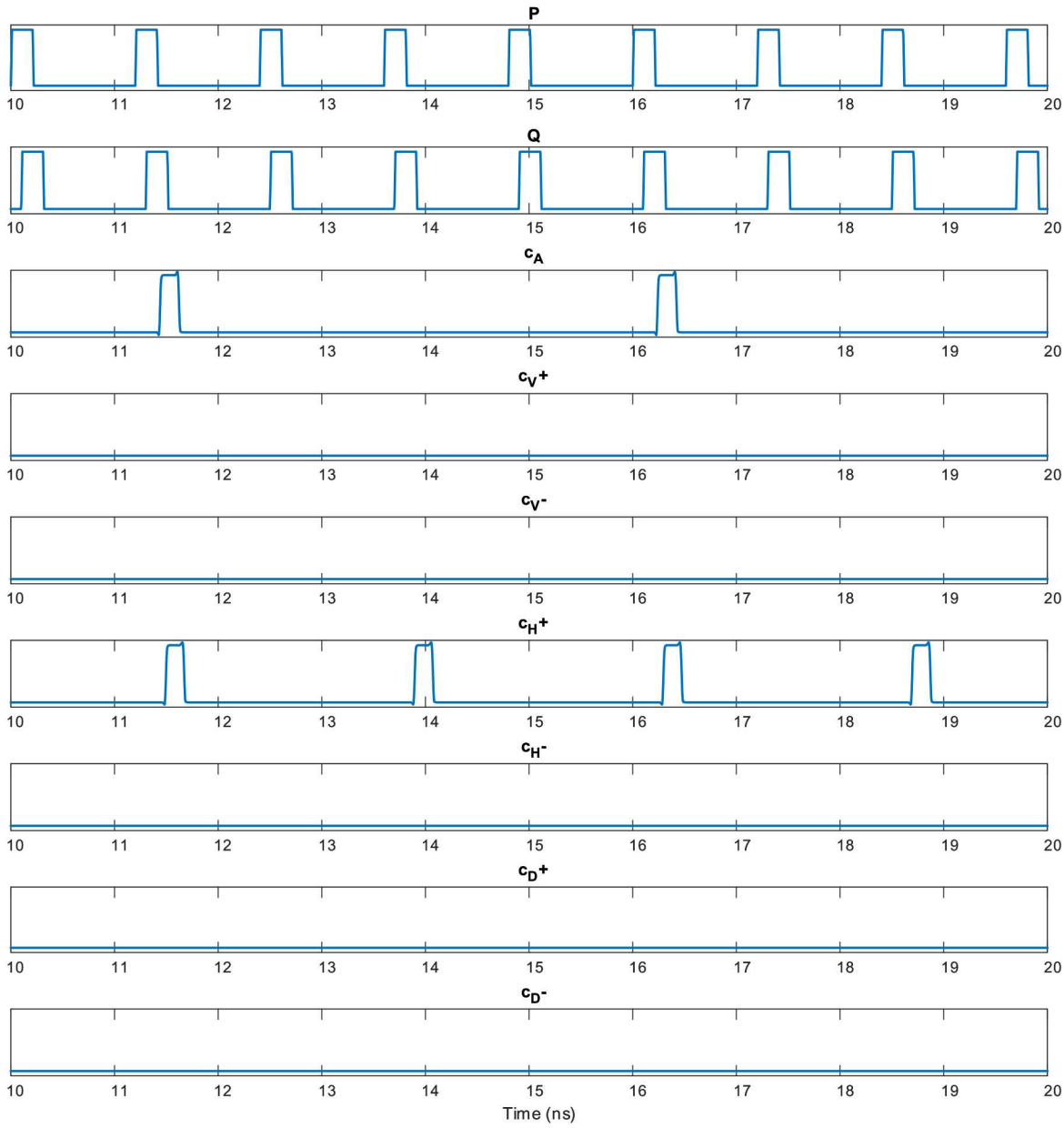

Figure 3-22. Haar node outputs with colliding spikes.

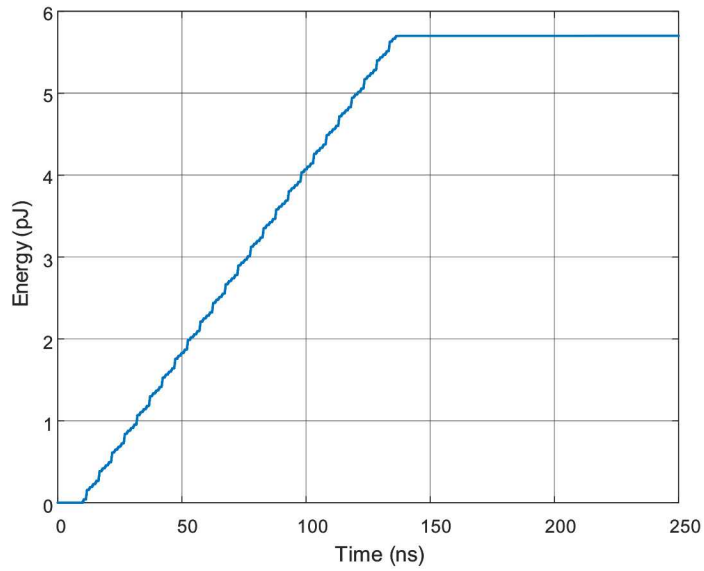

Figure 3-23. Integral of the approximation coefficient power consumption.

cumulative effect on a large array can be significant. Measuring the slope in the leakage region produces $m = 17.1$ nW. We assume this leakage is a constant DC power.

To calculate the energy per spike, we need two data points – one just before the first spike arrives and one immediately after the circuits settle subsequent to the last spike's arrival. These two data points can be read from the graph and are 197.316×10^{-18} J at 10 ns and 5.701 pJ at 136.8 ns. Taking the difference between these two data points gives 5.701 pJ and 126.801 ns. During the time difference of 126.801 ns, the circuit was also consuming power due to the leakage. The total energy consumed due to leakage is the leakage power times the duration

$$\Delta t \times m = 126.801 \times 10^{-9} \times 17.1 \times 10^{-9} = 2.168 \text{ fJ} \tag{3.11}$$

The energy due to the spikes will thus be

$$5.701 \text{ pJ} - 2.168 \text{ fJ} = 5.699 \text{ pJ}. \tag{3.12}$$

Dividing this by the number of spikes, 100, we have the c_A element consuming 57 fJ per input spike while having a DC leakage power of 17.1 nW. The same kind of analysis for the other circuit elements produces the results summarized in table 3-5.

3.5. HAAR NODE ALTERNATIVE IMPLEMENTATION

The Haar node implementation presented above is only one way to achieve the desired results. If timing and saturation error tolerance are relaxed, alternative implementations can consume less

Table 3-5. Haar node power consumption.

Element	Energy Per Spike (fJ)	Leakage Power (nW)
c_A	57	17
c_H and c_V	101	31
c_D	103	41
Total for Haar node	362	120

power and silicon area. This section presents our initial design which was modified to produce more predictable saturation behavior.

3.5.1. Alternative Approximation Coefficient Implementation

Figure 3-24 shows an alternative implementation for the approximation coefficient. In this implementation, the decimation by four is implemented using two divide by two in series (the first two D flip-flops). The output spike is generated in the same manner that was previously discussed and illustrated in figure 3-11.

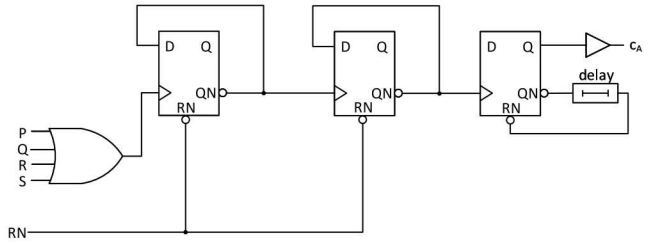

Figure 3-24. Approximation coefficient alternative implementation.

The primary fault with this approach is the ripple delay of having the two divide by two circuits in series. When compared to the c_H, c_V, and c_D coefficients, the delay from input spike to output spike will be approximately triple for this implementation due to the output of the first flip-flop being the clock signal to the second flip-flop who's output then clocks the third flip-flop. In order to improve the delay from the input to the output, the design of figure 3-11 was used which is a state machine based design. In this case, both flip-flops use the same clock signal which reduces the delay from input to output. The only additional circuitry in the more tolerant design is a single AND gate–a small increase in area and power. The chosen design is favored over this lightweight design since timing is improved at little cost.

3.5.2. Alternative Horizontal, Vertical, and Diagonal Coefficient Implementation

The horizontal, vertical, and diagonal lightweight designs use the same state machine logic; the difference is in generation of the output pulses. The left half of 3-25 shows the preferred design; the right half shows the alternative lightweight approach.

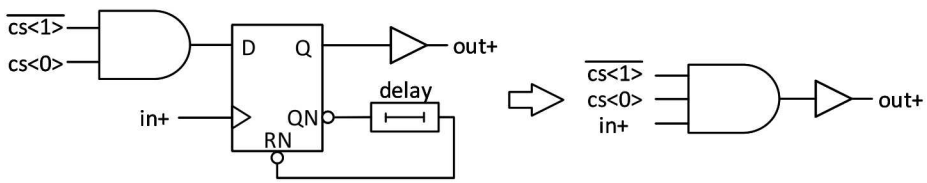

Figure 3-25. Horizontal, vertical, and diagonal coefficient alternative implementation.

In the left half, a D flip-flop generates the spike. When the state machine is in the desired state, 01 in the example shown, an output spike is generated if the next input spike is on one of the positive inputs. The delay element controls the width of the output spike. In the alternative approach (right side of figure 3-25), the output spike is generated through an AND gate. For the same positive output spike case, if the current state is 01, a spike on one of the positive inputs will create the output spike.

The benefit of lightweight approach is that a single AND gate replaces the D flip-flop and delay element. Figure 3-26 shows the area savings. The top half of the figure depicts the collision tolerant approach from section 3.3 while the bottom half shows the lightweight implementation. The lightweight implementation uses approximately half the area. The lightweight circuit reduces the area for one output by 27.9 μm^2. Considering that each element has two outputs and there are three total elements in each Haar node using this structure, the lightweight approach reduces total circuit are by 167.4 μm^2. For a 35 $\mu m \times 35$ μm pixel, this constitutes a 13.7% reduction in pixel area.

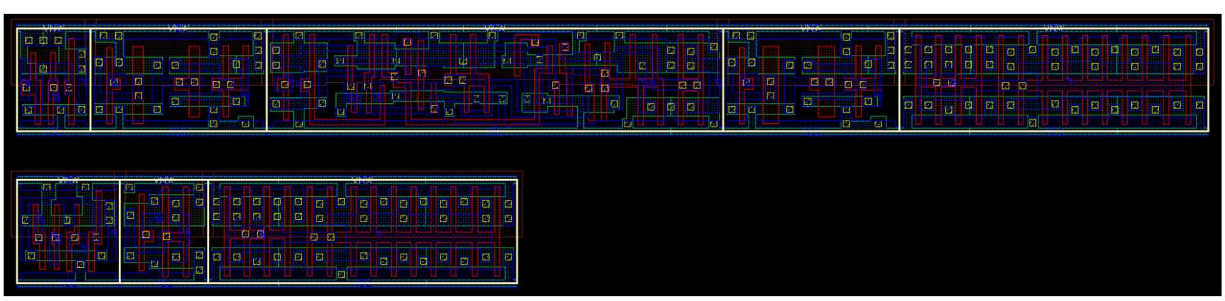

Figure 3-26. Horizontal, vertical, and diagonal coefficient alternative layout comparison.

The lightweight design also requires less power. The c_H, c_V, and c_D elements in the collision-tolerant design each consume about 100 fJ per input spike, while the lightweight approach only uses approximately 70 fJ per input spike.

While the area and power consumption are improved with this alternative approach, the greatest drawback is that circuit behavior at saturating spike rates becomes more erratic. Figure 3-27 shows the behavior of one of the outputs for an increasing spike rate. This simulation is showing the positive c_H output when the input spikes are alternating on the P and Q inputs with an even delay between spikes. The top waveform shows the outputs when the delay between spikes is 340 ps. In this case, the circuit is operating as desired. Reducing the time between spikes to 240 ps (second waveform) where the c_H block is saturating and there is an unexpected double-spiking behavior. Further decreasing the time between spikes to 140 ps gives (third waveform) produces output spikes faster than desired desired; this output rate is too high to be detected at the next level in the hierarchy since there is not enough time between spikes. Finally, if the spike rate becomes too high (bottom waveform), the output will saturate at a high level and no spikes will be detected.

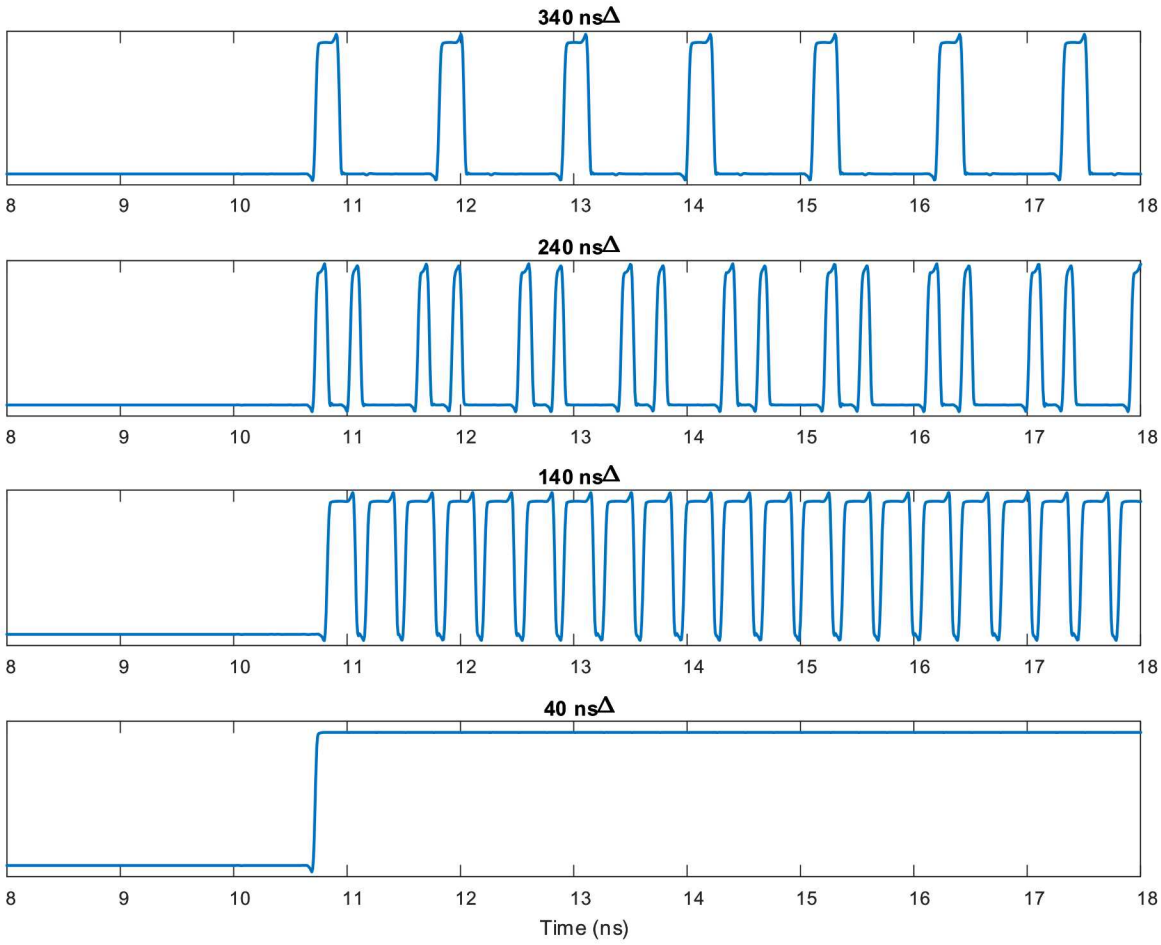

Figure 3-27. The lightweight c_H design output is pathological when inputs spikes are too close together. The top trace shows desired behavior, the second shows an unwanted double-spiking, the thirds output spike rate is too fast for downstream processing, and the bottom shows a single continuous output voltage.

Ultimately, we preferred the design from section 3.3. Although the chosen approach uses more power and area, the operation in saturation is better defined and is therefore better aligned with the `sparrow` simulator. By aligning the transistor-level behavior with the `sparrow` behavior we can more reliably predict total FPA operation. If the ultimate desire is to reduce the power consumption as much as possible, the alternative approach presented in this section should be used.

3.6. QUADTREE IMPLEMENTATION

The quadtree circuit must achieve two functions: decimate a set of input spikes and provide the address of the current pulse that is being output from the system. The quadtree implementation consists of two blocks, each block implements one of these two functions.

3.6.1. Quadtree Decimation and Output Spike Generation

The circuit in figure 3-28 decimates the input spikes. Four input spikes on the P, Q, R, and S lines are imposed on a four-input OR gate. In the Haar node discussion, P, Q, R, and S represented either pixel spike inputs (at the first level of the transform) or approximation coefficient spikes (at higher transform levels). In contrast, the inputs to the quadtree come from four Haar node output spikes. The Haar outputs entering a quadtree element can come from any part of the Haar transform, for example, they could be c_H spikes from four different levels in the transform. We do require, though, that the Haar node outputs have the same sign since there are two quadtrees – one for the positive Haar outputs and one for the negative Haar outputs. The output of the OR gate acts as a clock signal for the following circuitry which will pulse whenever there is a spike on any of the four inputs. A set of two D flip-flops follow the OR gate. In the configuration shown, with the QN output connected to the D input, the flip-flop acts as a divide-by-two circuit. The first flip-flop output will toggle after two spikes on the input while the second flip-flop output will toggle after four spikes on the input.

This system has three different outputs – the pulse coming from the OR-gate and the two flip-flop outputs. Each of these drive a tri-state buffer. The enabled tri-state buffer selects the decimation rate. To have decimation rates higher than four, more D flip-flops can be added to the signal chain with a corresponding tri-state buffer driving the output line.

While the output of the OR gate from figure 3-28 is a spike, the output of the D flip-flops is not; its output will resemble a square wave with a rising edge every other spike input on the first D flip-flop and every four input spikes on the second flip-flop. These longer duty cycle waveforms need to be converted to spikes and buffered to drive the next stage in the architecture. This is done using the circuit shown in figure 3-29.

The output of the decimation circuit drives the clock input on the D flip-flop. When there is a spike, the Q output toggles high while QN goes low. The QN line goes low, and after a delay drives the RN terminal of the D flip-flop low which in turn resets the Q output to a low value. The

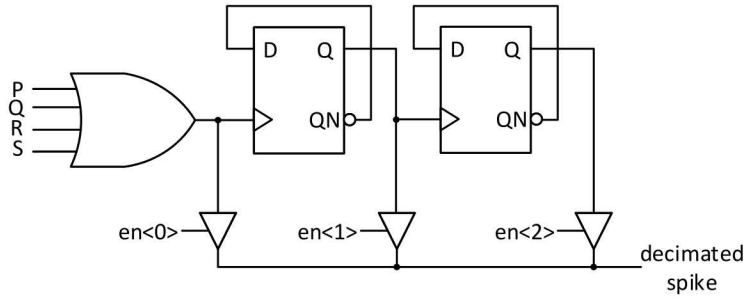

Figure 3-28. Quadtree decimation circuit.

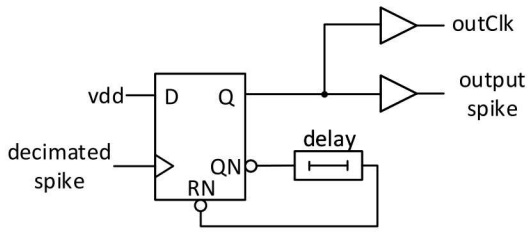

Figure 3-29. Quadtree output pulse generation circuit.

spike produced at the output can be made wider by increasing the amount of delay in the delay element.

There are two outputs shown in the figure, the main output spike which will go to the next level in the system and a single-pulse clock signal, outClk, which is used to drive the logic in some of the circuits discussed next.

3.6.2. Quadtree Address Output Generation

The second major piece of the quadtree element is the address generation circuit. In the address generation circuit, the output reflects which input (P, Q, R, or S) the spike came from. The inputs are coded as follows: P-00, Q-10, R-01, and S-11. For example, when a spike arrives on the R input while the decimation state is allowing an output, the address generation circuit will output 01.

At higher levels in the quadtree, each element receives address bits from the lower levels. These address bits, along with two new address bits reflecting the P, Q, R, or S input, must be pushed to the output along with the spike marking the time.

Figure 3-30 shows the first circuit for the address generation. The left half of the circuit includes a set of OR and NOR gates which act as a decoder. When there is a spike on one of the inputs, the addX nodes will change to the corresponding two bit code where addX<1> is the MSB and

addX<0> is the LSB. While the address is decoded every time there is a pulse only want the output address to change when the circuit is actually sending an output. The two D flip-flops constrain the address change at the quadtree output to occur only when outClk (from figure 3-29) is active. When outClk is active the current address at the input to the D flip-flop is loaded onto the address output buffers which hold the current address until the next output pulse.

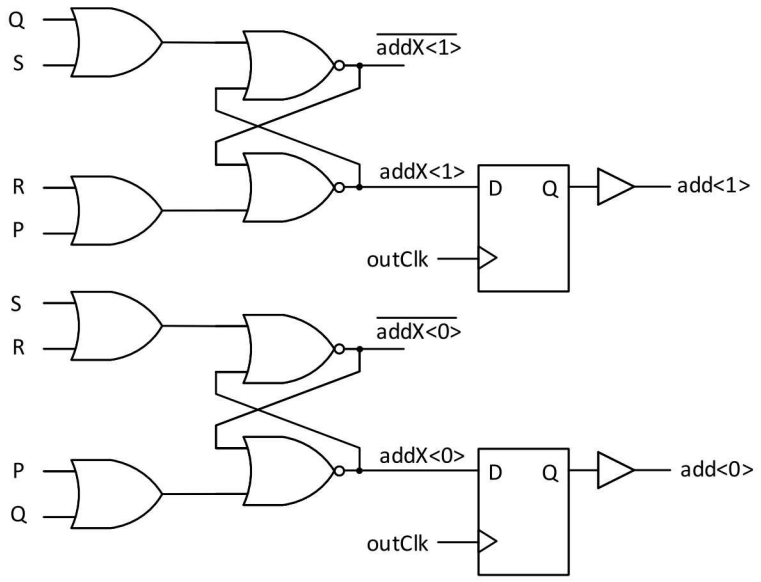

Figure 3-30. Quadtree output address generation circuit.

As previously mentioned, the quadtree element must also drive the address bits from the lower levels in the hierarchy. The circuit of figure 3-31 receives the address bits from the lower levels in the hierarchy and applies them to the set of tri-state buffers. The tri-state buffers corresponding to the most recent input spike are enabled through the decoder on the left. The tri-state buffer outputs change with every spike on the input; to control the output change we load them onto the D flip-flop in the same manner as in the local-stage address generation circuit of figure 3-30. The presented approach has twelve tri-state buffers in each set allowing up to six levels of address bits. In addition to the two bits generated at the current level that allows a total of 2^{14} or 16384 Haar coefficients. If there are fewer than twelve bits at the input, the unused higher MSBs can be tied to a logic low and the corresponding outputs ignored. This allows a single quadtree circuit design to be used at different levels in the hierarchy, trading implementation simplicity for redundancy.

3.7. QUADTREE PERFORMANCE

Quadtree performance is governed by the same issues that govern Haar node performance. Specifically, the quadtree element must decimate and calculate or forward address bits correctly, it must behave tolerably in the presence of overlapping input spikes, and we must estimate its energy use per input spike.

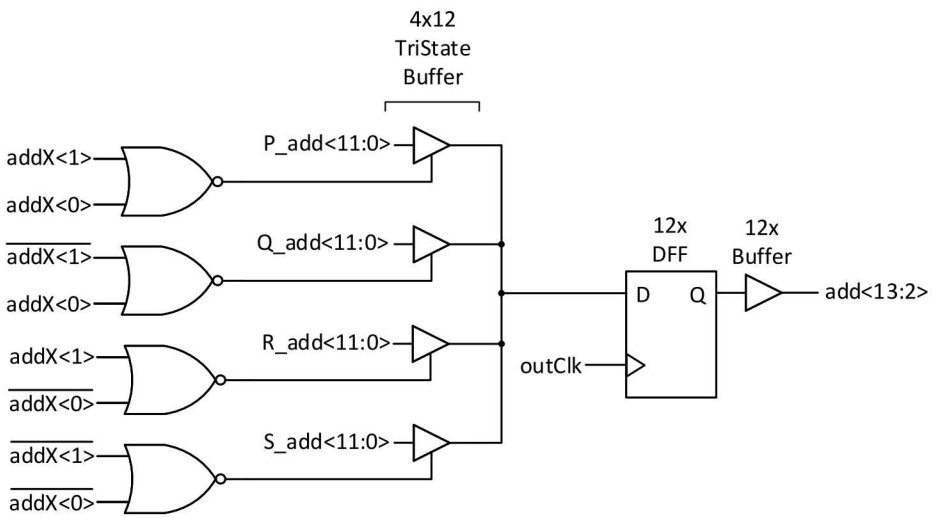

Figure 3-31. Quadtree output address driver circuit.

3.7.1. Quadtree Simulations in Normal Operation

The quadtree circuitry operates similarly to the approximation coefficient circuitry. Assuming the spikes have enough separation, the quadtree will output a spike every N input spikes, where the decimation factor N can be set to 1, 2, or 4 in the presented design. In the following simulations, $N = 2$, so the quadtree should output one spike for every other input spike. Unlike the c_A element, the quadtree must also output an address bit based on which input presented the spike.

Figure 3-32 shows the output of the quadtree node in the lowest level in the hierarchy – it passes no additional address bits to the output. There is 1 ns between spikes to allow the circuitry to operate without collisions or spike-saturation behavior. The output consists of a spike for every other input spike – a decimation rate of two, as expected. In addition to the output spike, there are two address bits which change coincidently with the output spikes. The address bits refer to which input the spike belongs to as mentioned in the previous section: P-00, Q-10, R-01, and S-11. For higher levels in the hierarchy, the quadtree will output two additional bits per level up. In figure 3-32 the initial address state is 11; however, without a spike present on QT output it has no meaning. The first two spikes arrive on input P, after the second one the address drops to 00 around 12 ns into the simulation, and concurrent with that drop QT output spikes.

3.7.2. Quadtree Simulation in Saturation

Referring back to figure 3-28, the spikes coming in to the quadtree are being added through an OR gate. If the spike rate at the input to the OR gate is too high, the OR gate output cannot respond quickly enough to go low before the next spike arrives at its input. If the output can not go low, then there is no rising edge on the clock input of the D flip-flops after the first pulse's rising edge.

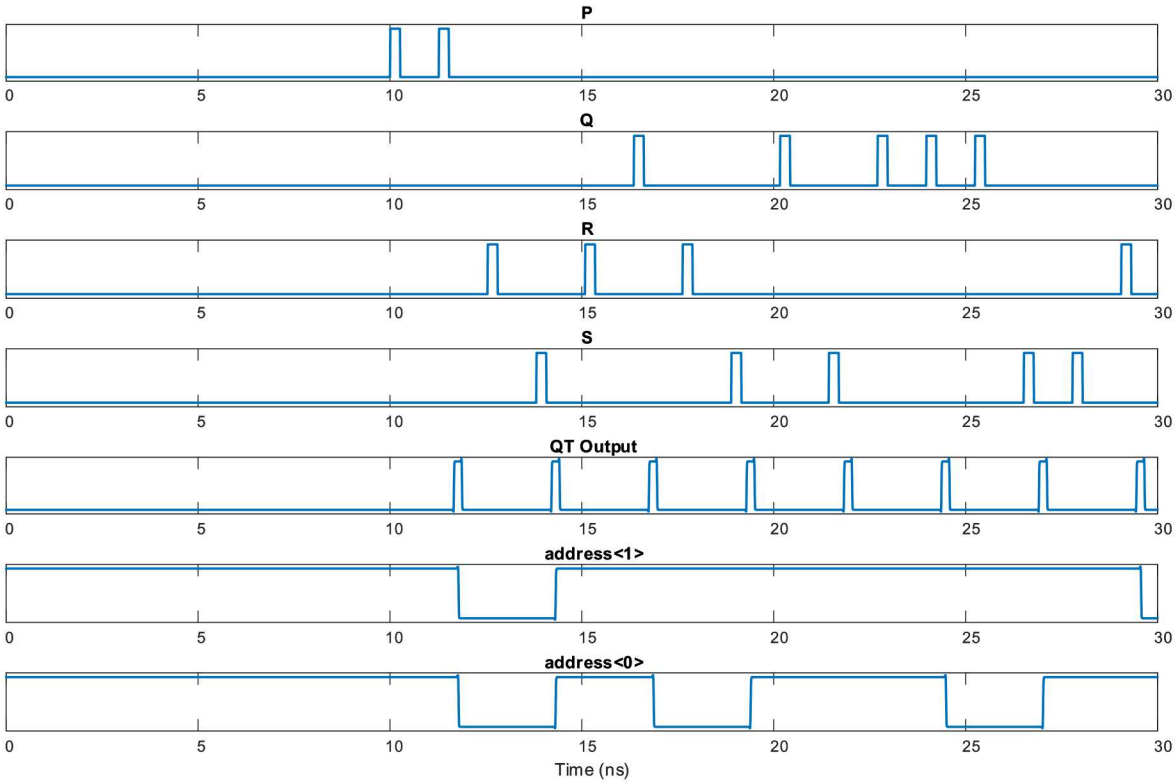

Figure 3-32. Quadtree simulated results for the lowest hierarchical level. The element decimates 2:1, so there is an output for every other input spike. The address reflects the forward input spike according to P-00, Q-10, R-01, and S-11.

In this case, only the first spike is counted, so if there is decimation, the quadtree will not have any output. Figure 3-33 shows that, when exposed to dense spiking, the quadtree element's output never spikes. There needs to be at least 200 ps between the falling edge of one spike and the rising edge of the next spike for the OR gate to operate correctly. When the OR gate's correct operation allows the rest of the circuitry to operate as desired.

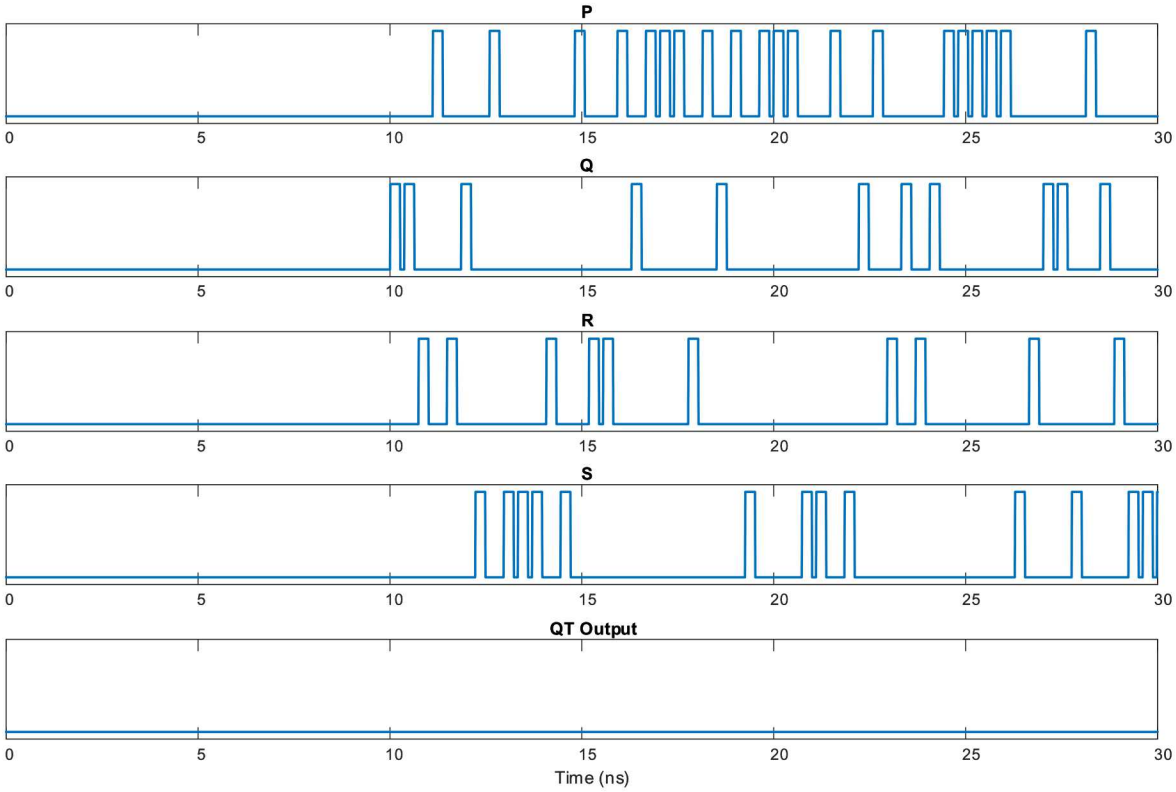

Figure 3-33. Quadtree simulated results with saturated input.

3.7.3. Quadtree Energy Consumption

The energy consumption analysis that was presented in section 3.4.3 for the Haar node was repeated for the quadtree. Figure 3-34 shows the integral of the quadtree dissipated power. The steady state, after spiking has ceased, results in a leakage power is 95.2 nW. The energy per spike (not including leakage current) is 197 fJ per input spike. These results are summarized in table 3-6.

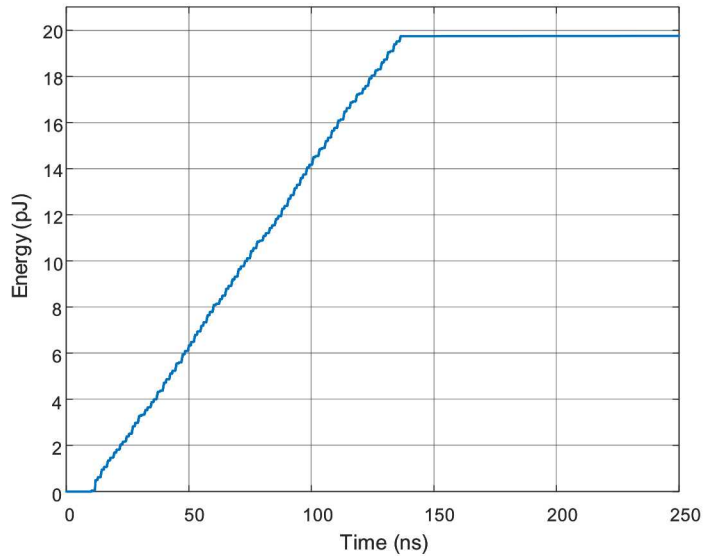

Figure 3-34. Integral of quadtree power consumption.

Table 3-6. Quadtree power consumption.

Element	Energy Per Spike (fJ)	Leakage Power (nW)
Quadtree	197	95.2

3.8. FULL SYSTEM LAYOUT AND SIMULATIONS

Layout translates the schematic circuit descriptions into the masks, metal, and exposed silicon to realize the design in CMOS. The realized circuit's behavior will differ from the conceptual behavior due, mainly, to conductor capacitance and resistance–the so-called *parasitic* properties. Complete layout provides accurate assessment of parasitic effects on circuits, accurate power estimates, and accurate space estimates. This section provides a complete layout in a single process node. Section 3.9 uses these results to explain power use at different nodes, and chapter 4 extends low-level circuit power estimates and node scaling properties to full system predictions.

3.8.1. Haar Node Layout

The Haar node layout was completed with the assumption of a 35 µm × 35 µm pixel area. Figure 3-35 shows the layout of the Haar node. The width of the layout is exactly 35 µm tall by 21 µm tall which takes up slightly more than half of one pixel's area.

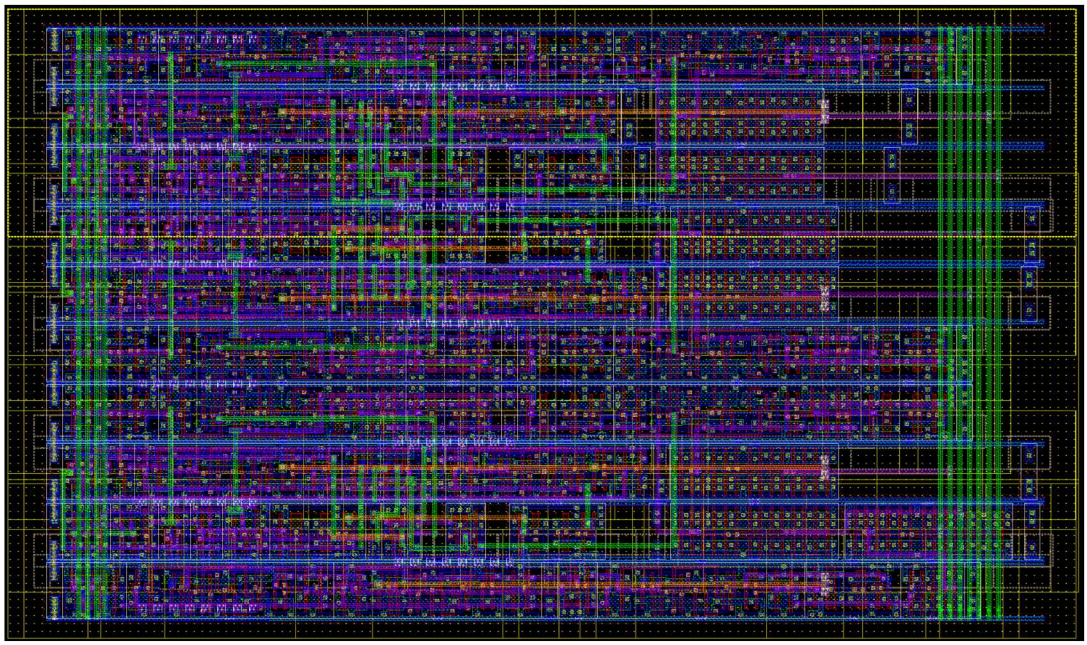

Figure 3-35. Haar node layout.

Figure 3-36 illustrates how the Haar node is positioned in a 2×2 pixel layout. At the lowest level in the hierarchy, one Haar node processes spikes coming from four pixels; this is why three of the pixels are empty. In a complete design, the upper half of the empty pixels holds the quadtree circuitry and Haar nodes at higher levels in the hierarchy.

The red square in each pixel (figure 3-36) is reserved for a photodiode. The vertical purple lines provide power, ground, and control signals to the FPA. The upper left pixel–depicted containing a Haar node–retains approximately one-quarter of its area for the ASP. After completion of the full

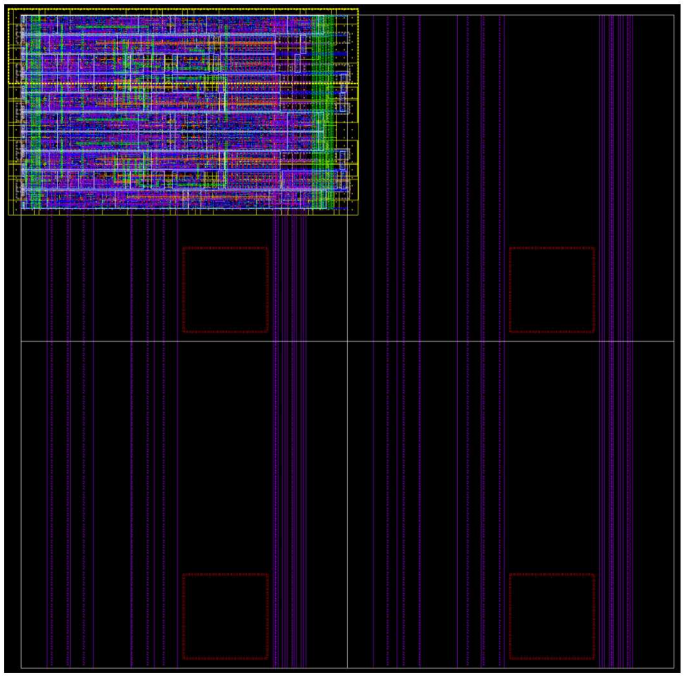

Figure 3-36. 2×2 **layout.**

FPA layout, each pixel includes one photodiode, one ASP circuit, and either a Haar node or quadtree element.

3.8.2. Quadtree Layout

Figure 3-37 shows the layout of the quadtree. It is the same size as the Haar node layout, 35 µm wide and 21 µm tall, so it fits in slightly more than half of one pixel's area.

3.8.3. Haar Node and Quadtree PEX Simulations

The layouts of the Haar node and quadtree were processed with a parasitic extraction (PEX) tool. This tool calculates the parasitic capacitance and resistance of all the metal interconnects. The PEX circuits can then be simulated to see how these additional parasitics affect the performance of each block.

All of the circuits in the Haar node and quadtree are digital standard cells with robust designs. Consequently, the primary difference between the PEX simulation and the ideal schematics is the power performance. Both of these blocks were simulated with parasitics included and the analysis from section 3.4.3 repeated. Table 3-7 summarizes the results. All of the additional parasitics cause the energy per spike to increase by approximately 75%, but the leakage power reduces when including parasitics.

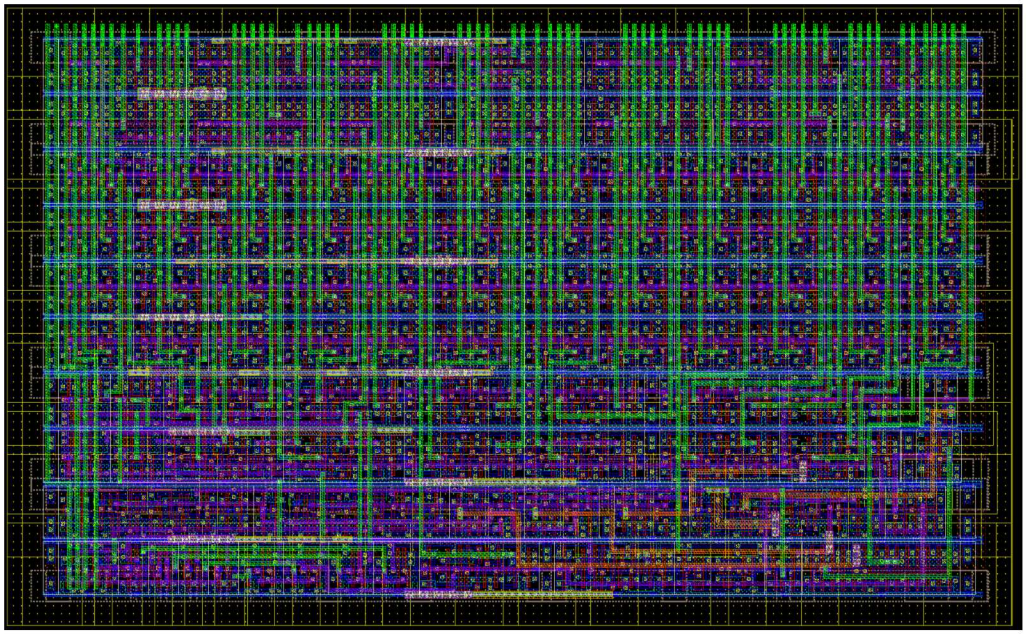

Figure 3-37. Quadtree layout.

Table 3-7. Haar node and quadtree power comparison. The PEX power numbers are consistent with the numbers in table 4-2 on page 100, though summarized in fewer numbers here.

Element	Energy Per Spike (fJ)	Leakage Power (nW)
Haar node (schematic)	362	120
Haar node (PEX)	617	50
Quadtree (schematic)	197	95
Quadtree (PEX)	349	67

3.8.4. 4×4 Pixel Array Layout

Figure 3-38 shows the layout of a 4×4 pixel array. This array has four Haar nodes at the pixel level in the hierarchy. The c_A outputs of each of these Haar nodes feed the inputs to a fifth Haar node. The c_H, c_V, and c_D outputs from the pixel level Haar nodes feed the quadtree. There are six c_H, c_V, and c_D outputs coming from each Haar node, or 24 total outputs which go to the quadtree. Each quadtree has four inputs, so there are six quadtree unit cells to process the Haar node outputs. The Haar node outputs go to the nearest quadtree input. There are five empty pixels that do not contain a Haar node or quadtree cell. These empty pixels are filled, at higher levels in the hierarchy, with more Haar nodes and quadtree cells.

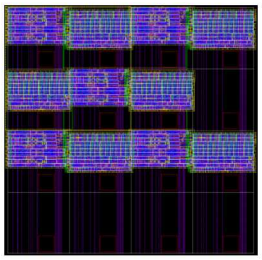

Figure 3-38. 4×4 pixel layout.

3.8.5. 8×8 Pixel Array Layout

Expanding on the 4×4 layout, to get the 8×8 layout of figure 3-39. This layout tiles four of the 4×4 layouts of figure 3-38 and adds one Haar node and six quadtree cells. The second level Haar nodes that take the c_A outputs from the lowest level Haar nodes are feeding the new Haar node–the level three Haar node. The c_H, c_V, and c_D outputs from the level two Haar node feed six new quadtree cells, added in the unoccupied pixel areas. The 24 quadtree cells from the 4×4 tiles (four tiles with six quadtrees each) have their output pulses and address bits passed to an additional six quadtree cells. These quadtree cells output four address bits.

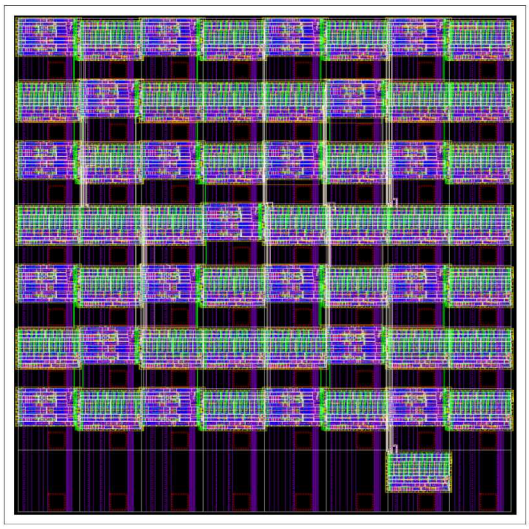

Figure 3-39. 8×8 **layout.**

3.8.6. 16×16 Pixel Array Layout

The 16×16 pixel array layout in figure 3-40 consists of four 8×8 pixel array cells tiled into a 16×16 array. One additional Haar node processes the outputs of the third level Haar nodes. There are eighteen additional quadtree cells added. Six of the quadtree cells receive input from the c_H, c_V, and c_D outputs from the third level Haar nodes. The other twelve quadtree cells are

aggregating the lower level quadtrees. Six of these quadtrees output one spike and four address bits while the other six output one spike and six address bits.

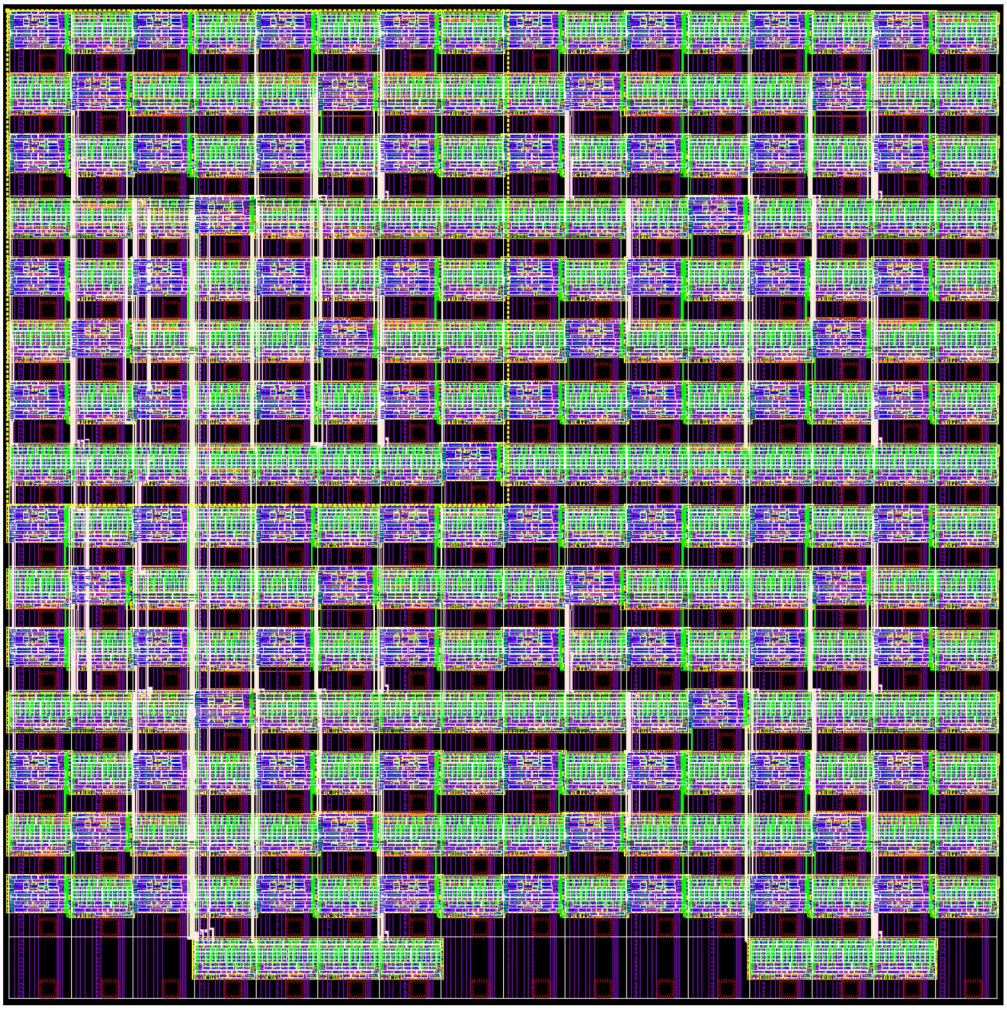

Figure 3-40. 16×16 layout.

3.8.7. Concerns With Layout Feasibility

The layout of the 16×16 pixel array was not completed. Approximately half of the interconnects were added. As the array size grows larger, the number of interconnections increases. At each level up in the hierarchy, the number of quadtree cells increases. As the number of quadtree cells increases, the number of additional address bits to route increases. It is simply not feasible to manually lay out larger arrays. For future work in developing a SPARR FPA, it is recommended that a digital IC designer convert the design to their design flow so that the layout can be produced through automatic place-and-route (APR).

Using an APR approach is likely the only way to complete the layout in a reasonable amount of time for larger, practical array sizes. Using a digital design approach will also be beneficial in improving signal timing which can be a concern for the larger array sizes since the Haar node and quadtree outputs must be routed long distances. Additional buffering can be added automatically where needed.

Another issue that may arise with this design is known as the antenna effect. In fabrication, an electric charge can build up on the metal lines that are being used to route the signals. As the line length increases, the charge increases. This accumulated charge can discharge into transistor gates causing transistor failure reducing yield during manufacturing. To fix antenna issues, diodes are placed in the layout to absorb the unwanted charge and prevent damage to each transistor's gate oxide. Since the layout of SPARR will consist of many long signal wires, care will be needed to eliminate antenna effects. Digital design approaches, like this in APR, can add antenna effect prevention diodes automatically.

3.9. POWER SCALING

The circuits presented in this document were designed for the Global Foundries 90 nm CMOS. The baseline design implemented in the 90 nm technology node requires a 35 µm × 35 µm pixel to hold the digital circuitry for the Haar transform and quadtree while maintaining some space for the ASP and photodiode in a monolithic implementation.

SPARR circuitry can be implemented in smaller area and lower power as the CMOS technology node advances. In general, each technology node decreases minimum feature size by a factor of about 0.7, that is, each smaller node's feature size about 70% of the larger. The area required by the circuit decreases by about 50% for each technology node.

Under the assumption that total FPA area will scale with technology node as above, and that frequency of operation will remain constant (i.e. similar pixel transfer functions), dynamic power ideally reduces by a factor of two for each technology node. In practice, one cannot realize this much improvement, primarily because the power supply voltages do not scale linearly with process nodes. Dynamic power will likely scale by a factor of about 1.5-1.8 reduction per technology node for the same circuit, assuming the frequency of operation stays the same, while occupying one half the area of the previous technology generation.

Another factor that must be considered is the increase in static power due to both transistor off-state leakage current and gate tunneling current at smaller technology nodes. There is no direct rule of thumb for the increase, but circuits that operate at relatively low frequency (in the SPARR architecture, those circuits that are functionally further from the pixel after decimation) may actually increase in power consumption due to an increase in static power dissipation.

Figure 3-41 illustrates how the power and pixel pitch may change with process node, assuming a 1.8 reduction factor in power per process node and a 70% pixel pitch reduction. In this figure, the power consumption for the 90 nm is our baseline and normalized to 1. To maximize the power reduction, the pixel pitch must be reduced at the 70% rate. If the pixel pitch is not reduced, the

logic circuitry that implements the Haar node and quadtree will still use less power, but the buffers that drive the signals from the output of these circuits to those at the next level in the hierarchy will not reduce in power. If the node size is decreased enough, the buffers become the dominant power consumer, so there is no further advantage in choosing a smaller (and more expensive) node size. Again referring to figure 3-41, if, for example, we plan to not implement a pixel with less than a 20 µm pitch, 45 nm would be the optimum technology when considering power consumption.

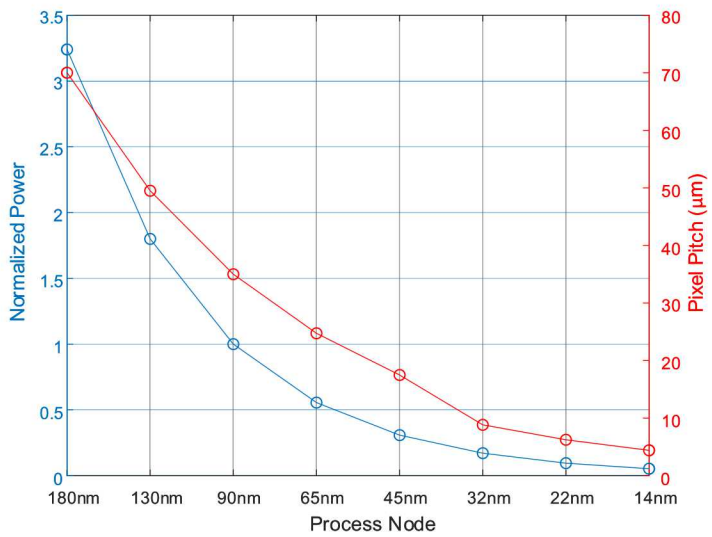

Figure 3-41. Power and pixel pitch scaling.

4. POWER ESTIMATES

4.1. ANALYSIS

Estimated power consumption for the SPARR sensor derives from quiescent draw (leakage current), photocurrent, and the spike-driven power consumption. Quiescent draw or leakage current is the power expended by the integrated circuit when it is doing nothing. Both photocurrent and spike processing are *dynamic*; the amount of power consumed in photocurrent depends on the light incident on the array while the amount of power consumed by spike processing depends on the frequency and pattern of spikes. The key items are the power consumed by the photodiodes P_{PD}, the power consumed in the spike processing in the Haar layer P_H, and the power consumed in the quadtree readout P_{QT}. Assembled, the total system power is

$$\begin{aligned} P &= P_{\text{PD}} + P_H + P_{QT} \\ &= P_{\text{PD-spike}} + P_{\text{PD-leak}} + P_{\text{PD-photo}} \\ &\quad + P_{\text{H-spike}} + P_{\text{H-leak}} \\ &\quad + P_{\text{QT-spike}} + P_{\text{QT-leak}}. \end{aligned} \tag{4.1}$$

The architecture of the Haar and quadtree design involves the ASPS feeding spikes into the Haar layer, and then the Haar-layer outputs feeding spikes into the quadtrees, where the data is read out. Figure 4-1 shows how these interconnect logically, pixels feeding the Haar transform layers, which in turn feed the quadtree read-out.

4.1.1. Leakage Power

With the preceding definitions, we can immediately define the power consumed by leakage, which depends principally on the number of pixels and the leakage per pixel. If the leakage power per pixel is $P_{\text{px-leak}}$, then the entire photodiode array's leakage power is

$$P_{\text{PD-leak}} = n_{\text{px}} P_{\text{px-leak}} \tag{4.2}$$

where n_{px} is the number of pixels in the array.

For the Haar network, the four different element types have a different power consumption, so we must count the number of times we compute each type of coefficient. We call this the number of nodes n_{node}, which is the same for all four Haar element types. Each node in the Haar network's input (level 1) receives spikes from four pixels. Therefore, the first level has $n_{\text{px}}/4$ nodes for each

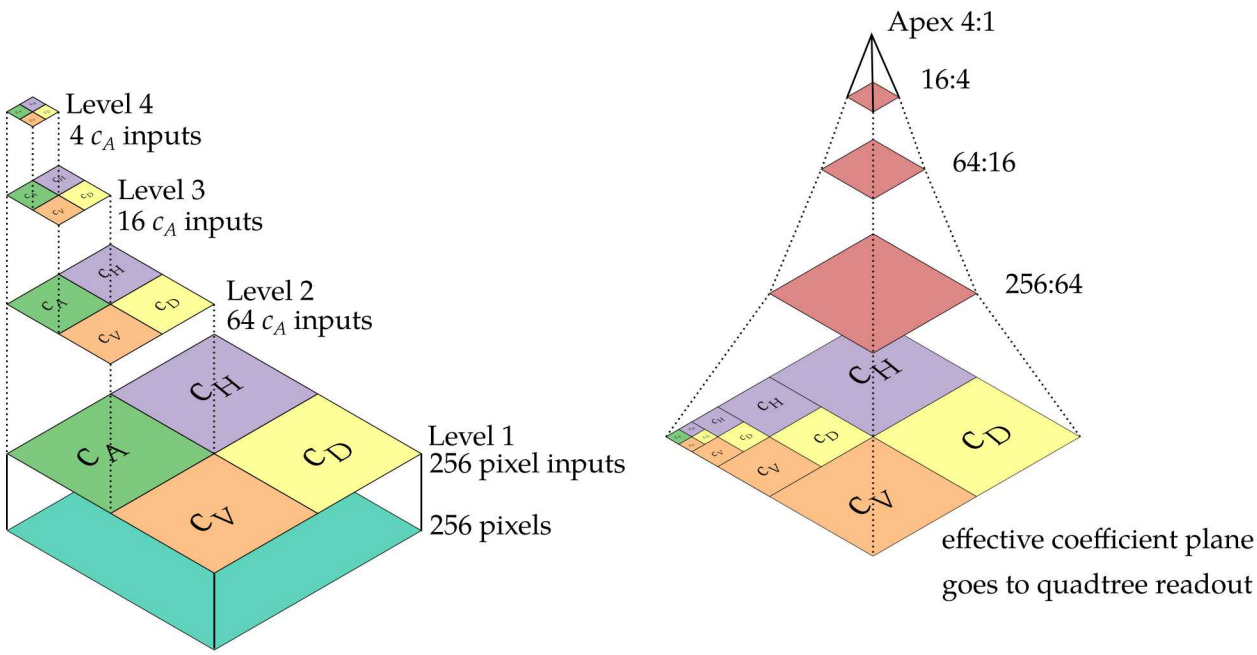

Figure 4-1. Depiction of pixels, Haar, and quadtree levels and structure. The diagram shows the logical layout; the physical layout of the circuits differ. The quadtree readout would typically connect spatially nearby coefficients, rather than the logically adjacent Haar coefficients.

Haar element, the second level has $n_{\text{px}}/16$ nodes, and so on. If $n_{\text{px}} = 2^L \times 2^L$, then there are L levels possible for both the Haar network and the quadtree. Therefore,

$$
\begin{aligned}
n_{\text{node}} &= \frac{n_{\text{px}}}{4} + \frac{n_{\text{px}}}{4^2} + \frac{n_{\text{px}}}{4^3} + \cdots + \frac{n_{\text{px}}}{4^L} \\
&= n_{\text{px}} \sum_{i=1}^{L} \left(\frac{1}{4}\right)^k \\
&= n_{\text{px}} \left(\frac{1/4 - (1/4)^{L+1}}{1 - 1/4}\right) \\
&= \frac{n_{\text{px}}}{3}\left(1 - \frac{1}{4^L}\right) \\
&= \frac{n_{\text{px}}}{3}\left(1 - \frac{1}{n_{\text{px}}}\right) \\
&= \frac{n_{\text{px}} - 1}{3} \\
&\approx \frac{n_{\text{px}}}{3} \quad \text{for } n_{\text{px}} \gg 1.
\end{aligned}
\tag{4.3}
$$

Since the horizontal and vertical Haar elements are identical, they consume the same amount of power per node. If the leakage power per Haar horizontal/vertical node is $P_{\text{H-leak-HV}}$, the leakage power per diagonal node is $P_{\text{H-leak-D}}$, and the leakage power per approximation node is $P_{\text{H-leak-A}}$,

then the entire array's Haar leakage power is

$$P_{\text{H-leak}} = n_{\text{node}} \left(2P_{\text{H-leak-HV}} + P_{\text{H-leak-D}} + P_{\text{H-leak-A}}\right). \tag{4.4}$$

Similar to the Haar network, each node in the quadtree input receives spikes from four inputs. The inputs are the Haar plane for the first quadtree level and previous quadtree levels for each subsequent quadtree level. Since the Haar network can produce positive and negative outputs, there is a positive quadtree and a negative quadtree. It can be shown that the number of nodes for the quadtree is the same as the number of nodes for each Haar element type n_{node}. Hence, if the leakage power per quadtree node is $P_{\text{QT-node-leak}}$, then the total quadtree leakage power is

$$P_{\text{QT-leak}} = 2n_{\text{node}} P_{\text{QT-node-leak}}. \tag{4.5}$$

4.1.2. Scene-Induced or Spike Power

To define the active spike-caused power consumption, we first need to express the rate of spikes coming from the pixels. We define the array-wide pixel spike rate as r_s, which we estimate with

$$r_s = \left(E_e A_{\text{ap}} T \frac{\lambda}{hc}\right)\left(\frac{QE(\lambda)}{N}\right) \tag{4.6}$$

where E_e is the incident irradiance at the aperature, A_{ap} is the aperture area, T is the optics system transmission factor, λ is the photon wavelength, h is Planck's constant, c is the speed of light, $QE(\lambda)$ is the quantum efficiency as a function of wavelength, and N is the number of electrons each pixel capacitor can hold. The first factor on the right-hand-side of equation (4.6) is the rate that photons reach the photodiode layer. The second term is the conversion factor from photons to spikes.

As mentioned above, each pixel has an electron accumulation well or capacitor. Upon filling its well, the pixel will spike and reset. The pixel accumulates charge as photoelectrons flow across the photodiode junction, which is typically reverse-biased, and the energy expended by the flow of photoelectrons is just voltage times current. Reverse biasing a diode places a voltage on the diode opposite its notional direction of conduction, thus causing conduction to occur only when photons liberate electrons. Since each pixel can hold N electrons before spiking and the diode junction is biased to V_{PD} volts, the power expended in photocurrent is

$$P_{\text{PD-photo}} = N r_s q V_{\text{PD}} \tag{4.7}$$

where q is the elementary charge $1.602\,176\,620\,8 \times 10^{-19}$ C. Typically, V_{PD} is of order 1 or 10^1, N is of order 10^3 to 10^6, and r_s is of order 10^{12}, so the total photocurrent power may be about 1 W. For many systems or scenes it will often be much less than 1 W.

The pixel electronics that emit the spike once the well is filled expend energy equal to $W_{\text{px-spike}}$ each time the pixel spikes. These terms combine to form the spike-induced or scene-induced photodetector layer power consumption

$$P_{\text{PD-spike}} = r_s W_{\text{px-spike}}. \tag{4.8}$$

The processing of the pixel spikes occurs in the Haar transform layer. Each pixel spike feeds one Haar processing node, and each node calculates four coefficients, diagonal, horizontal, vertical, and approximation or c_D, c_H, c_V, c_A respectively. The horizontal and vertical nodes are identical and each consume an average energy of W_{HV} per input spike. The diagonal node consumes W_D energy per input spike, and the approximation node uses W_A energy per input spike. The total power consumed in level 1 of the Haar transform is then

$$P_{\text{H-spike},1} = r_s(2W_{HV} + W_D + W_A). \tag{4.9}$$

The inter-stage spike decimation rate is configured so that the per-element spike rate out of the first stage is the same as the per-pixel spike rate into the first layer. The second layer, however, has only one fourth the elements of the first layer, so the power it consumes is only one fourth the first layer's. The third layer again consumes one fourth the power of the second layer and so on. This can be written

$$\begin{aligned}
P_{\text{H-spike}} &= P_{\text{H-spike},1} + P_{\text{H-spike},2} + \cdots + P_{\text{H-spike},L} \\
&= P_{\text{H-spike},1}\left(1 + \frac{1}{4} + \cdots + \frac{1}{4^{L-1}}\right) \\
&= P_{\text{H-spike},1}\sum_{k=0}^{L-1}\frac{1}{4^k} \\
&= P_{\text{H-spike},1}\left(\frac{1-(1/4)^L}{1-(1/4)}\right) \\
&= P_{\text{H-spike},1}\frac{4}{3}\frac{4^L-1}{4^L} \\
&= P_{\text{H-spike},1}\frac{4}{n_{\text{px}}}\frac{n_{\text{px}}-1}{3} \\
&= P_{\text{H-spike},1}\frac{4n_{\text{node}}}{n_{\text{px}}} && (4.10) \\
&\approx \frac{4}{3}P_{\text{H-spike},1} \quad \text{for } n_{\text{px}} \gg 1. && (4.11)
\end{aligned}$$

It is somewhat more complex to estimate the power consumed by the quadtree. The first step is to estimate the spike rate coming into the quadtree from the Haar transform. Because the Haar transform performs spatial whitening, the spike rate into the quadtree will be essentially uniform across the entire quadtree input. The rate spikes enter the quadtree is the pixel spike rate scaled by the whitening factor f_w. The whitening factor depends on the viewed scene, but assuming the scene has the same spatial spectrum that the whitener was designed for, the average spike rate per coefficient will be the same across the entire wavelet transform tree. Therefore we can find the reduction factor by finding the ratio of the average per-coefficient spike rate at level one to the input average spike rate per pixel. The spike rate is proportional to the optical power, so the brightness of the image is proportional to the spike rate. Furthermore, because a spike on a positive line uses as much power as a spike on a negative line, the magnitude sum is used. The first level of the Haar transform represents three quarters of all the coefficients. The assumption

that the whitening and the spatial spectrum match means that all coefficients' spike rates can be accounted for by multiplying the total spike rate across the three quadrants provided by the first level of the Haar transform by 4/3. Putting this together, we have

$$f_w = \frac{4}{3}\left(\frac{\sum|c_{H,1}| + \sum|c_{V,1}| + \sum|c_{D,1}|}{r_s}\right), \tag{4.12}$$

where $c_{H,1}$, $c_{V,1}$, and $c_{D,1}$ are the (signed) spike rates on the horizontal, vertical, and detail coefficient lines from level 1 of the Haar transform and the summation notation means to add the spike rate over all the coefficients in the H, V or C sets. The fraction f_w tells us what portion of the whitened scene comes out of the Haar transform, noting that the c_A portion goes into the next level of the Haar transform and therefore does not enter the quadtree at all.

The quadtree can be thought of as layers, with each layer having one fourth as many nodes and each layer adding two address bits to the layer below. The energy consumed per spike within a quadtree layer is constant. However, the added address bits for nodes closer to the read-out apex means those nodes use more energy per spike than nodes in the layer below. We assume the energy consumption in layer k is f_q times the energy consumption in layer $k-1$.

At the same time the power per spike is increasing, the number of spikes may be decreasing because of decimation within the quadtree node. Nodes can be configured to decimate 1:1 (no decimation), 2:1, 4:1, or more as designed. The decimation is described as a factor, f_d, which can be controlled by quadtree level. In a typical design, the quadtree reduction factor or spike rate throttling is not uniform from level to level up the tree. Assuming a quadtree element supports 1:1, 2:1, or 4:1 decimation, then the system would be configured to maximize decimation at the lowest level. For example, it would be preferable to set level 1 to 4:1 and level 2 to 1:1, rather than setting level 1 and level 2 to 2:1, though both configurations would produce the same final readout, at least if we ignore collisions. Using 4:1 in the first level reduces the chance of collisions at the second level, and may result in more accurate overall measurements coming from the array. Because quadtree elements nearer the apex use more power per spike (they drive more address lines) power is minimized by decimating at the lowest possible level.

The lowest level of the quadtree receives input from the Haar coefficient outputs. There are n_{px} Haar output elements. The spike rate into level 1 of the quadtree is $r_s f_w$. The output rate from that level (and input rate to the next) depends only on the decimation factor in level 1. Therefore, the input rate to level 2 is $r_s\, f_w\, f_{d,1}$. Similarly, the input rate to the third layer is $r_s\, f_w\, f_{d,1}\, f_{d,2}$, and so on. The spiking power consumed in the quadtrees is

$$\begin{aligned} P_{\text{QT-spike}} &= W_{\text{QT}}\, r_s\, f_w \left(1 + f_{d,1} f_q + f_{d,1}\, f_{d,2} f_q^2 + \cdots + f_{d,1}\, f_{d,2} \cdots f_{d,L-1} f_q^{L-1}\right) \\ &= W_{\text{QT}}\, r_s\, f_w \left(\sum_{i=0}^{L-1}\left(f_q^i \prod_{j=1}^{i} f_{d,j}\right)\right), \end{aligned} \tag{4.13}$$

where W_{QT} is the base-level quadtree element's energy per input spike.

When all the quadtree levels are configured to be non-decimating (1:1) or maximum decimating (4:1), then the levels all have the same decimation factor. When all the levels have the same

decimation factor, that is, $f_{d,1} = f_{d,2} = \cdots = f_d$ the power equation simplifies to

$$\begin{aligned} P_{\text{QT-spike}} &= W_{\text{QT}}\, r_s\, f_w \left(1 + f_d f_q + f_d^2 f_q^2 + \ldots\right) \\ &= W_{\text{QT}}\, r_s\, f_w \sum_{i=0}^{L-1} \left(f_d f_q\right)^i \\ &= W_{\text{QT}}\, r_s\, f_w \left(\frac{1 - \left(f_d f_q\right)^L}{1 - f_d f_q}\right) \end{aligned} \tag{4.14}$$

where $f_d f_q \neq 1$.

This spiking treatment does not separate the positive and negative spike lines. Empirically the agreement between the model and a simulation is quite good (see section 4.1.3). Although not proven, we argue that if two lines being differenced by a Haar node are nearly balanced then its output may include both positive and negative spikes but would only produce spikes at a low rate. Therefore, assuming spikes are strictly on either a positive or negative line causes only a small error in the total power model estimate.

4.1.3. Estimating the Haar Spike Rate Reduction f_w

The reduction in spike rate is defined in equation (4.12). This factor can be found by assuming an individual pixel's spike rate is proportional to the intensity on it, and that the spike rate out of the Haar transform is proportional to the magnitude of the value that would come from performing a static Haar transform on an image as opposed to a spiking process. We are inferring the spiking system's behavior by performing static image analysis using a two dimensional Haar image transform.

Figure 4-2 shows the input scene in reverse grayscale–black corresponds to faster spiking than white. The right side of the figure shows the magnitude $|c_H|$, $|c_V|$, and $|c_D|$ coefficients. The sum of the transformed magnitude coefficients divided by the sum of the input scene (all non-zero) produces the estimated fraction of spikes f_w.

The calculation of f_w across the entire array, from `sparrow`, shows $f_w = 0.113$ using the same input scene, accounting for spikes on both the positive and negative Haar outputs. In other words, the whitening factor estimated from a flat image (0.159) is very close to the whitening factor calculated by counting spikes with `sparrow` (0.113). The power model uses only the first level of transform; the image processing version (0.119) is close to the spike count estimate form `sparrow` (0.081) as well. The Haar image transform approach produces a slightly higher estimate for this scene. Analysis of several fully-illuminated earth scenes shows modest variation in the calculated f_w term, but all are approximately equal. Although other non-earth scenes will vary, provided the spatial power spectrum is similar the array will exhibit similar whitening factors.

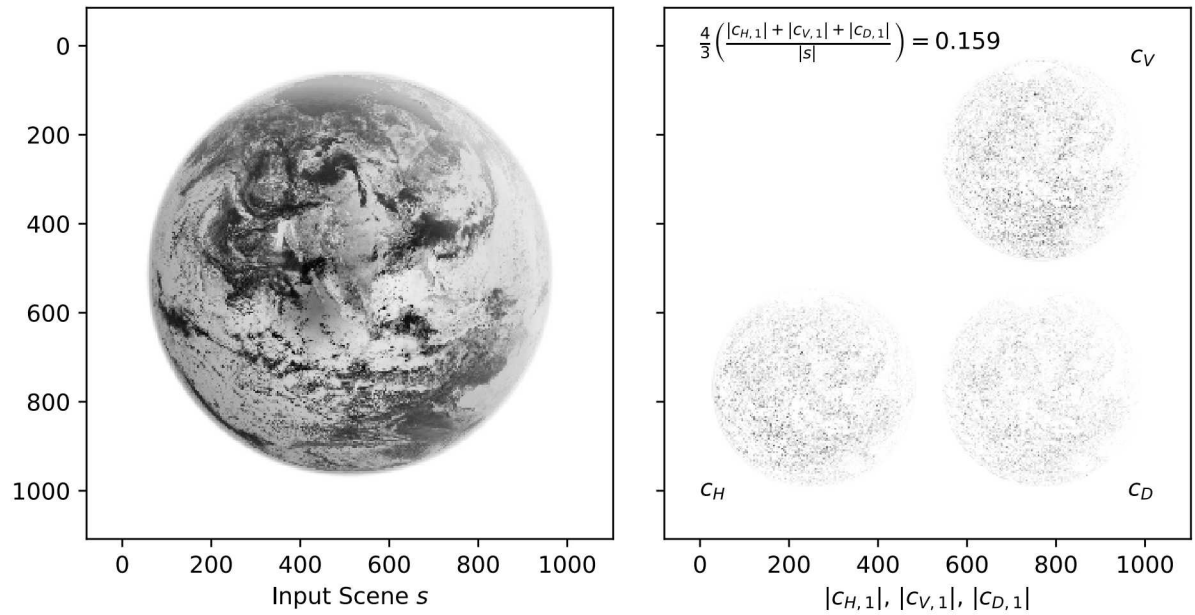

Figure 4-2. Estimation of f_w from a scene.

4.1.4. Scaling Power by Process Node

Integrated circuit production process nodes are usually identified by their smallest feature size in nanometers. For example, a 90 nm process node will produce circuits with physical features as small as 90 nm. Process nodes are usually available at steps of factors of $\sqrt{2}$, see table 4-1 [35]. For example, the next smaller process node from 90 nm would be about $90/\sqrt{2} \approx 65$ nm. According to Dennard scaling [9], each step in process node size would typically reduce power consumption by a factor of two, assuming operating frequencies remain the same and assuming the total circuit size contracts proportional to the process node scale reduction. In SPARR, the physical dimensions may not contract due to the pixel pitch remaining constant. When the circuit does not contract according to the process node there will be additional power lost in transmission at the relatively longer run lengths. In this case, we assume the power is also reduced by $\sqrt{2}$ for each reduction in process node size.

Using the process node power scaling coefficient s, we can rewrite equation (4.1) dependent on this scaling as,

$$\begin{aligned} P &= P_{\text{PD}} + P_H + P_{QT} \\ &= s\left(P_{\text{PD-spike}} + P_{\text{H-spike}} + P_{\text{QT-spike}}\right) \\ &\quad + P_{\text{PD-leak}} + P_{\text{PD-photo}} \\ &\quad + P_{\text{H-leak}} \\ &\quad + P_{\text{QT-leak}}. \end{aligned} \tag{4.15}$$

Table 4-1. CMOS process nodes between 5 and 150 nm.

Node Size nm	Reduction Factor	Size nm	Reduction Factor
150	0.83	32	0.8
130	0.87	28	0.88
110	0.85	22	0.79
90	0.82	20	0.91
80	0.89	16	0.8
65	0.81	14	0.88
55	0.85	10	0.71
45	0.82	7	0.7
40	0.89	5	0.71

In other words, the power scales by

$$s \in \left[2^{\text{steps}},\ 2^{\text{steps}/2}\right], \tag{4.16}$$

where the number of steps going from node size a to size b is

$$\text{steps} = \frac{\log b - \log a}{\log \sqrt{2}}. \tag{4.17}$$

Steps are really representing the number of factors of $\sqrt{2}$ there are between two process sizes, whether increasing or decreasing the node size. Increasing node sizes ($b > a$) results in positive steps while decreasing nodes ($a > b$) results in negative steps. For example, if, as in the example in section 4.1.5, we use a reference power at 90 nm and then ask about the power performance at a notional 32 nm process size, we find steps= -2.98 and therefore that the spiking power scales like

$$s \in [0.13, 0.36]. \tag{4.18}$$

4.1.5. Power Analysis of an Example

This section defines values for all the key terms in the equations in the power model, and then calculates the expected power. In addition, the detailed `sparrow` simulation was configured to process the same inputs, using the same system configuration, and that provided an estimate of the spike counts in a duration.

It may seem that the pixel spike rate r_s can be arbitarily large, but spikes arriving at a DAPE nearly simultaneously will collide, often resulting in no state change in the DAPE. Because the quadtree aggregates spikes as the spikes approach the apex, collision tends to be worse nearer the apex. Decimation, or throttling, in the quadtree nodes mitigates collision, and for the following calculation and simulation the throttling was experimentally determined to be 4:1, 4:1, then 2:1 in

Table 4-2. Notional parameters for analysis, from 90 nm process node. The Haar and quadtree power values are consistent with the PEX power numbers from table 3-7 on page 87.

Variable	Value		Note
n_{px}	1.05×10^{6}		number of pixels (1024^2)
process node	90	nm	process node feature size for estimates of W
r_s	5.47×10^{12}	/s	pixel spike rate (whole array)
N	800	e	number of electrons per spike
V_{PD}	3	V	Photodiode bias voltage
$P_{\text{px-leak}}$	5×10^{-7}	W	pixel leakage power
$P_{\text{H-leak-HV}}$	1.35×10^{-8}	W	Haar H and V element leakage power
$P_{\text{H-leak-D}}$	3.10×10^{-8}	W	Haar D element leakage power
$P_{\text{H-leak-A}}$	3.59×10^{-9}	W	Haar A element leakage power
$P_{\text{QT-node-leak}}$	4.87×10^{-8}	W	quadtree node leakage power
$W_{\text{PD-spike}}$	1×10^{-13}	J	energy per spike expended by pixel
W_{HV}	1.68×10^{-13}	J	Haar H or V node energy per input spike
W_D	1.65×10^{-13}	J	Haar D node energy per input spike
W_A	8.89×10^{-14}	J	Haar approximation node energy per input spike
W_{QT}	3.66×10^{-13}	J	Lowest-level quadtree energy per input spike
f_w	0.165		whitening Haar spike-rate reduction factor
f_d	4:1, 4:1, 4:1, 2:1, 1:1, …		quadtree decimation factors
f_q	1.16		quadtree energy per spike growth factor per layer

Table 4-3. Calculated results using table 4-2. The leakage results for the `sparrow` case are assumed to be identical, because `sparrow` does not calculate leakage.

Variable	Value			Note
	Model	`sparrow`		
L	10	10		number of quadtree or Haar levels
		Photodetector		
P_{PD}	1.07	1.07	W	Total photodetector power
$P_{\text{PD-leak}}$	0.524	0.524	W	Photodetector leakage power
$P_{\text{PD-spike}}$	0.547	0.547	W	Photodetector spike power
$P_{\text{PD-photo}}$	2.10×10^{-3}	2.10×10^{-3}	W	Photocurrent power
		Haar Transform		
P_{H}	4.32	4.31	W	Total Haar transform power
$P_{\text{H-leak}}$	0.0215	0.0215	W	Haar leakage power
$P_{\text{H-spike}}$	4.30	4.29	W	Haar spike power
		Quadtree		
P_{QT}	0.522	0.374	W	Total Quadtree power
$P_{\text{QT-leak}}$	0.0340	0.0340	W	Quadtree leakage power
$P_{\text{QT-spike}}$	0.489	0.340	W	Quadtree spike power
Total Power	5.92	5.76	W	
Spike-Only	5.34	5.18	W	Total power without leakage
Leakage-Only	0.580	0.580	W	Total power without spikes
Total Power (32 nm low)	0.13	0.13	W	Process node scaled low estimate
Total Power (32 nm high)	1.26	1.24	W	Process node scaled high estimate

the first three levels, and 1:1 in all subsequent levels. Both `sparrow` and the analytical power model used the same configuration.

Table 4-3 shows the results from the included power model, along with calculations based on the spike counts from `sparrow`.

This section's calculations show that, except for leakage, all power use is dynamic. Consequently the question "how much power does SPARR use?" has an irritating answer: "it depends on what SPARR is seeing!" To try to find a useful answer, assume that SPARR is viewing an earth scene so that the Haar spike rate reduction statistics apply (is in figure 4-2). With that assumption the entire power consumption can be summarized as the static (leakage) power plus a dynamic power consumption which depends only on the *average pixel spike rate*. Figure 4-3 shows how power use varies with that average spike rate, from less than half a Watt with few spikes to 10 W near peak supportable spike rates. Where a SPARR implementation fits into a total optical system depends on the ASP capacitance, the telescope, and of course the scene it is viewing. The SPARR

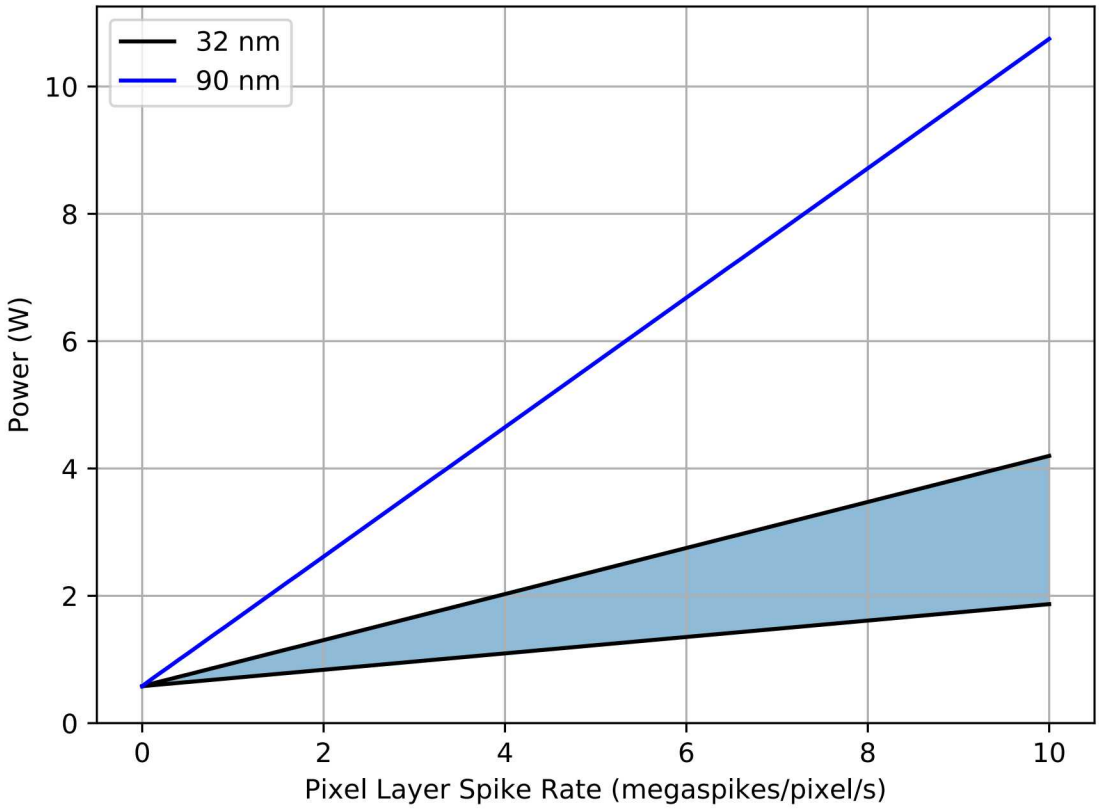

Figure 4-3. Power use as a function of the average pixel spike rate, using the 90 nm process node. Throttling increases as needed from no quadtree throttling at the left, to all layers using 4:1 decimation at the right. A 1024×1024 array is assumed, with earth-scene Haar properties. Scaling to a 32 nm node results in a range of power use shown by the shaded blue region; variation is due to unspecified run lengths (pixel pitch).

focal plane design must fit within its target system.

A new FPA development effort would probably not use the Global Foundries 90 nm process node, and instead use something newer–reaping power savings in the process. Assuming the same vigorous average pixel spike rate and fully-lit earth scene used for table 4-3, figure 4-4 shows how the predicted power scales for different design points. The blue region on the left reflects the range of power consumption due to buffer drivers and longer run lengths.

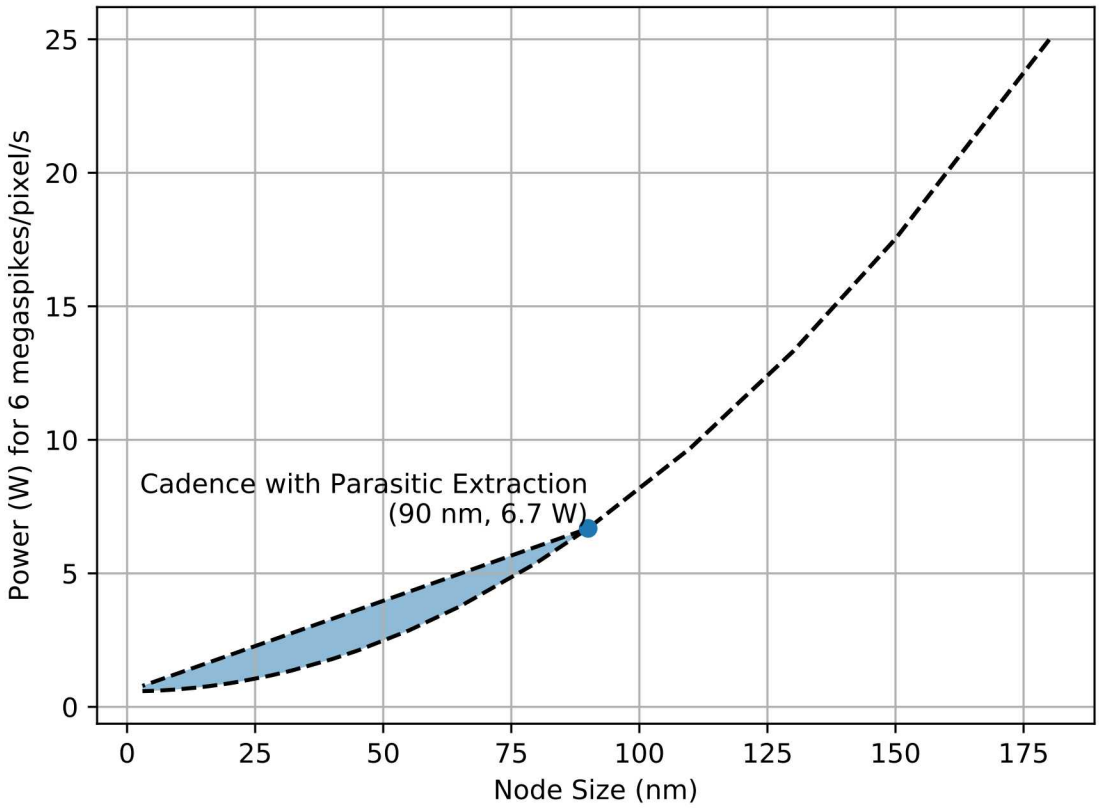

Figure 4-4. Power use as a function of the CMOS process node. Calculation is using a 1024×1024 pixel full earth scene with throttling 4:1, 4:1, 2:1, and then 1:1 for remaining quadtree levels.

5. USES FOR SPARR

This chapter discusses several uses of the SPARR sensor which cover only some of SPARR's possible uses. Nevertheless, the topics show that conventional functions like imaging and change detection work with a spiking output. Section 5.1 discusses change detection and time-of-onset estimation for transient optical signals, section 5.2 discusses tracking objects as they move across the focal plane. Finally, section 5.3 discusses using SPARR as an imager–analogous to a conventional camera. Together, these uses show major modes of operation all cooperating with the same spike detection and spike processing sensor.

5.1. SIGNAL DETECTION AND RECONSTRUCTION

5.1.1. Introduction

The goal of detection for the SPARR sensor is to be able to differentiate a small, fast-rising transient signal from the background, spatially locate it on the array, and temporally identify the onset and duration of the event. We define "small" as having an intensity significantly smaller than the average intensity of the background. For example, an event that has a peak intensity of 1% of the background would be considered small. Brighter events could also be considered small, but we will make 1% our goal. Very small signals such as these have a very low signal-to-noise ratio (SNR). Since SNR is defined as the intensity function of the signal divided by the intensity function of the noise, an event with maximum intensity of 1% of the background corresponds to −20 dB of SNR.

5.1.2. Data

Any event detection comes from the output of the quadtrees. Each quadtree output gives $\{\mathbf{S_0},\mathbf{S_1},\cdots\}$, where $\mathbf{S}_n$ refers to spike n and is defined by its output location, time-tag, and magnitude. We say output location here, rather than pixel location, because the quadtrees' output is from the Haar coefficients rather than directly from the pixels. The output location actually refers to a Haar coefficient location, as opposed to an input pixel location.

5.1.2.1. Preprocessing

We preprocess the spike stream into a time series for processing further in the pipeline. Given an integration frequency f, we define non-overlapping time windows of length $T = 1/f$. For each time window, we sum the total number of spikes in each output location that occurs during that time window, including the signs and magnitudes of each spike. This results in the time series x_{ij}^n, which is the number of summed spikes for output Haar location (i, j) in time bin n.

An example of this preprocessing for one output coefficient can be seen in figure 5-1. The plot on the left shows spikes with different magnitudes occurring at different times during the data stream on both the positive and negative spike lines for a given Haar coefficient. We create a time series from this by summing over all spikes during a given time interval. Here, we bin in intervals of 0.5 s (2 Hz integration frequency). The dotted lines on the left plot indicate the time windows. Binning in this way results in the time series to the right.

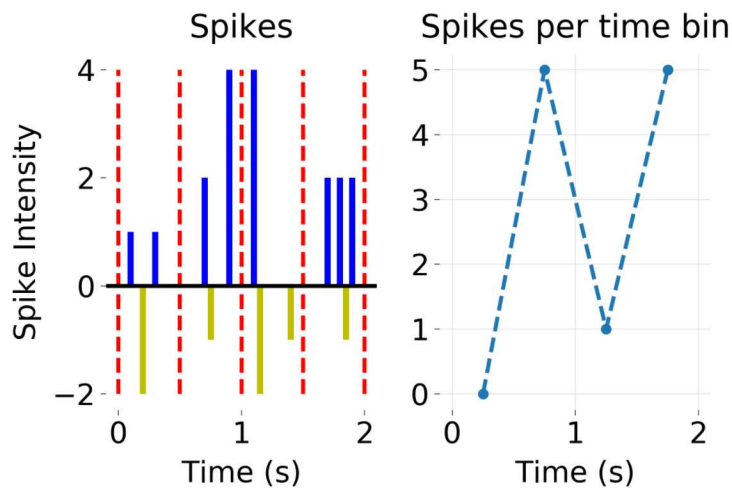

Figure 5-1. Preprocessing spikes to a time series. We sum the total number of spikes (on both the positive and negative spike lines) seen on the left over a given time interval (0.5 s) to get the resulting time series seen on the right.

We use this binned time series for detection for the remainder of this chapter. It is interesting to note that this binning process can also act as a rough signal reconstruction mechanism. Under the assumption of a piecewise linear signal, we can divide the time series by the window length to obtain a mean estimate of the rate of the signal at that time bin. If we reconstruct in this manner and then take the inverse Haar transform per time sample, we would get a whitened version of the input image.

5.1.2.2. Rationale

Due to the spatial nature of the Haar processing on-array, the output of the quadtrees represents an intertwining of spike streams from multiple pixels. Furthermore, the Haar transform also introduces a sign term, as there can be both positive and negative spikes as a result of the transform.

While the inter-arrival times between spikes at the pixel level are well-understood and follow a gamma distribution, the distribution of inter-arrival times between spikes out of the Haar transform and the quadtrees are much more difficult to characterize. This is due to the intertwining of neighboring pixels as a result of the Haar transform. We estimated the distribution of inter-arrival times after the Haar transform with simulation, but the distributions are complicated and involve a mixture of a uniform distribution with multiple approximately Gaussian distributions. We found no symbolic representation that could be used for rigorous statistical analysis.

In contrast, by employing the central limit theorem, we conclude that the we can approximate (with small error) the summation of spikes in a given time interval (including signs and magnitudes) as a Gaussian distribution with unknown mean and variance. Figure 5-2 shows a histogram of the number of spikes per time interval from a `sparrow` simulation, as well as the Gaussian fit to the data. As can be seen, the Gaussian distribution describes the random process reasonably well, with the note that the preprocessed data is discrete in nature (integer spike counts). The Gaussian fit does not enforce this, although we could enforce this by integration of the continuous distribution in the future.

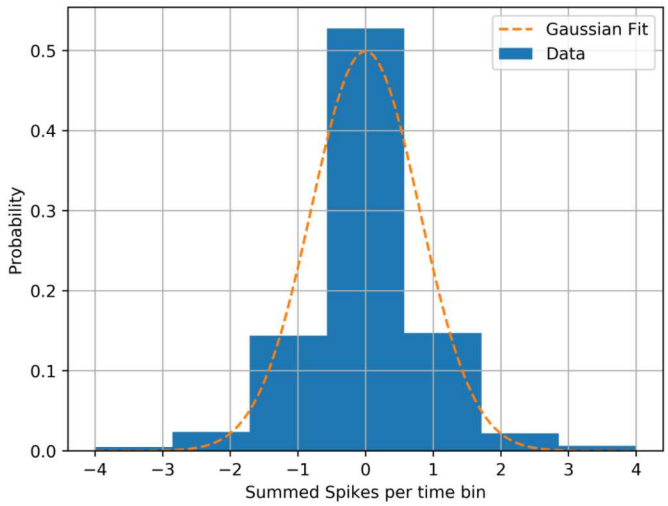

Figure 5-2. Distribution of spikes per 25 kHz from one Haar coefficient output. As can be seen, the distribution is accurately modeled by a normal distribution.

5.1.3. Algorithm Overview

We formulate the problem of detecting a fast-rising transient event as a change detection problem. We create a time-varying estimate of the statistical distribution of the background per Haar coefficient. We then compare all new data coming in to this background distribution, and if the new data is sufficiently different such that it is most likely not generated from the background distribution, we declare the presence of an event. This indicates that the distribution of the data has changed.

As described above, for each Haar coefficient we can model the distribution of spikes per time bin out of the quadtrees as a Gaussian random variable. We can create an estimate of the mean and variance describing the background, or noise, that our sensor observes. We model events in an additive fashion (i.e. $x_{ij}^n = s_{ij}^n + w_{ij}^n$, where s_{ij}^n is the portion of the received data due to the signal of interest at Haar location (i, j) and time bin n, and w_{ij}^n is white noise at the same location and time bin. An event results in a data stream described by a Gaussian random variable with a higher mean, and potentially a different variance as well, although empirical evidence indicates a fairly constant variance, at least for small signals. We analyze the data stream to search for a change in the underlying distribution.

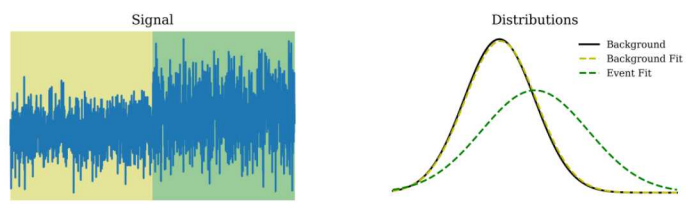

Figure 5-3. Change detection logic. We compare distributions between the background and new data coming in. If they come from different distributions, as is displayed with the green distribution, then a detection is declared. If they are very similar, as with the yellow distribution, no detection is declared.

Figure 5-3 outlines the change detection logic at a high level. The plot on the left shows the preprocessed quadtree outputs as a function of time. The yellow portion signifies the absence of a signal while the green portion signifies the presence of a signal. The abrupt shift in the preprocessed data also shows this change. The plot on the right shows three different distributions. The black distribution describes the running estimate of the background. This is the distribution against which we compare new data. The yellow distribution describes the data prior to the onset of the event (corresponding to the yellow highlight on the left plot). The green distribution describes the data after the onset of the event (corresponding to the green highlight on the left plot). Notice how similar the yellow distribution is to the background distribution. This is because the incoming data described by the yellow distribution is indeed from the background. No event is declared when analyzing the yellow data. In contrast, notice the difference in distributions between the green distribution and that of the background. This difference is indicative of the onset of an event.

5.1.4. Hypotheses

We define some notation here. As mentioned above, we maintain a running estimate of the background distribution. The notation $x \sim \backslash \left(\mu, \sigma^2\right)$ means that variable x is distributed according to the normal distribution with mean μ and variance σ^2. The background for a particular Haar coefficient out of the quadtrees, w_{ij}^n is modeled as a normal distribution according to

$$w_{ij}^n \sim \mathcal{N}\left(\mu_{\text{bg,ij}}, \sigma_{\text{bg,ij}}^2\right). \tag{5.1}$$

Likewise, we have a buffer of incoming data. This buffer, x_{ij}^n, is described by

$$x_{ij}^n \sim \mathcal{N}\left(\mu_{\text{buff,ij}}, \sigma_{\text{buff,ij}}^2\right). \tag{5.2}$$

Note that according to empirical evidence through numerous experiments, we make the assumption that $\sigma_{\text{bg,ij}}^2 = \sigma_{\text{buff,ij}}^2 = \sigma_{ij}^2$. For the purpose of readability, we will drop the spacial subscripts ij and move the temporal superscript n to a subscript where necessary.

We formulate the distribution comparison in the form of a hypothesis test. More formally, we have two hypotheses, $\mathcal{H}_0$ and $\mathcal{H}_1$. The null hypothesis, $\mathcal{H}_0$, is that no event occurs. This hypothesis states that the distribution describing the buffer is the same as the distribution describing the background, which means that $\mu_{\text{bg}} = \mu_{\text{buff}}$. The alternate hypothesis, $\mathcal{H}_1$, is that an event did occur, and $\mu_{\text{bg}} < \mu_{\text{buff}}$. More formally,

$$\mathcal{H}_0 : x_n \sim \mathcal{N}\left(\mu_{\text{bg}}, \sigma^2\right) \qquad \forall n \in \text{buffer}, \tag{5.3}$$

$$\mathcal{H}_1 : x_n \sim \mathcal{N}\left(\mu_{\text{buff}}, \sigma^2\right) \qquad \forall n \in \text{buffer}. \tag{5.4}$$

We reject the null hypothesis if

$$\frac{\mathcal{L}(\mathcal{H}_1)}{\mathcal{L}(\mathcal{H}_0)} > \tau, \tag{5.5}$$

where the left-hand-side of the equation is the likelihood ratio and the right-hand-side is a threshold. If the null hypothesis is rejected, a detection is declared.

5.1.5. Parameter Estimation

The parameters μ_{bg}, μ_{buff}, and σ^2 are not known and must be estimated. As such, we denote the parameters in equations (5.3) and (5.4) as $\hat{\mu}_{\text{bg}}$, $\hat{\mu}_{\text{buff}}$, and $\hat{\sigma}^2$, respectively. This adds uncertainty to the hypothesis test. In this section, we discuss how these parameters are estimated, as well as the uncertainty in the estimators.

5.1.5.1. Buffer Parameter Estimation

The only data we can observe is the buffer, and all parameter estimations come from observing it. The buffer has length N and is updated each time step. The new data is shifted into the buffer, and the oldest data (located at the $N-1$ memory address) is shifted out. We estimate the mean of the buffer with

$$\hat{\mu}_{\text{buff}} = \frac{1}{N}\sum_{n=0}^{N-1} x_n. \tag{5.6}$$

Since $x_n \sim \mathcal{N}\left(\mu_{\text{buff}}, \sigma^2\right)$,

$$\hat{\mu}_{\text{buff}} \sim \mathcal{N}\left(\mu_{\text{buff}}, \frac{\sigma^2}{N}\right). \tag{5.7}$$

This means that the parameter $\hat{\mu}_{\text{buff}}$ is, itself, a Gaussian random variable. It is unbiased, with a variance that decreases with N. There is, therefore, a trade-off for the size of the buffer. A larger buffer leads to a smaller-variance estimate of the mean of the buffer. However, it also requires that a change in distribution be present for a longer duration for the mean to change significantly. Hence, a larger N means it takes more time to determine the presence of an event, which may be problematic for time-sensitive missions.

To compute the variance of the buffer, we use

$$\hat{\sigma}^2 = \frac{1}{N-1}\sum_{n=0}^{N-1} (x_n - \hat{\mu}_{\text{buff}})^2. \tag{5.8}$$

Using equation (5.7) and $x_n \sim \mathcal{N}\left(\mu_{\text{buff}}, \sigma^2\right)$, we can compute that

$$\hat{\sigma}^2 \sim \Gamma\left(k = \frac{N}{2}, \theta = \frac{2\sigma^2}{N}\right) \tag{5.9}$$

where k is the shape parameter of the gamma distribution and θ is the scale parameter. The estimator for $\hat{\sigma}^2$ described in (5.8) is a modified version of the maximum likelihood estimate. The estimator results in an unbiased estimate described by a gamma random variable with variance $k\theta^2 = 2(\sigma^2)^2/N$. This estimator, like $\hat{\mu}_{\text{buff}}$, reduces variance with an increase in buffer size.

5.1.5.2. Background Parameter Estimation

All background parameter estimation stems directly from buffer estimations. In the absence of an event, the buffer is the background. However, in order to have a lower-variance estimate of the background, as well as to keep small events from bleeding into the background until detected, we maintain background estimates as running averages of the buffer estimates for each parameter, updated at a reduced rate. We have two additional buffers of length M, one containing the computed buffer means (referred to as m-buff) and one containing the computed buffer variances (referred to as v-buff). These are updated every K time steps. This slower update rate is to reduce computational complexity of the system while allowing a slowly varying background.

The parameter $\hat{\mu}_{\text{bg}}$ is simply the mean of the mean buffer. As such,

$$\hat{\mu}_{\text{bg}} = \frac{1}{M}\sum_{m=0}^{M-1}\hat{\mu}_{\text{m-buff},m} \tag{5.10}$$

$$\sim \mathcal{N}\left(\mu_{\text{bg}}, \frac{\sigma^2}{MN}\right), \tag{5.11}$$

since the sum of random Gaussian variables results in another random Gaussian variable.

It is important to note that there is some settling time required, where μ_{buff} must be equal to μ_{bg}. More specifically, when we start the system, we must wait N time samples before the data buffer is filled and outputs a valid mean estimate, and another KM time samples before the mean buffer is filled with valid data and thus produces a valid estimate of the mean of the background. During this time, there must only be background present in the received data, so that $\mu_{\text{buff}} = \mu_{\text{bg}}$. Only after this settling time has occurred can we begin to look for possible detections.

The parameter $\hat{\sigma}^2$ is simply the mean of the variance buffer. As such,

$$\hat{\sigma}^2 = \frac{1}{M}\sum_{m=0}^{M-1}\hat{\sigma}^2_{\text{v-buff,m}} \tag{5.12}$$

$$\sim \Gamma\left(k = \frac{NM}{2}, \theta = \frac{2\sigma^2}{MN}\right). \tag{5.13}$$

The same settling restrictions as for $\hat{\mu}_{\text{bg}}$ are present. The sum of random gamma variables results in another random gamma variable only if the added variables are independent. As such, this is only the true distribution if the $\hat{\sigma}_{\text{v-buff,m}}$ estimates are independent (assuming we update $\hat{\sigma}^2$ only when the buffer contains all new data from the last update). Therefore, K must be larger than N. Also note that the variance of both $\hat{\mu}_{\text{bg}}$ and $\hat{\sigma}^2$ reduce by MN. As such, we can increase M independently from N to reduce variance in our background estimations. However, the larger the M, the slower the background can vary before our background estimation falls behind.

5.1.6. Detection Computation

5.1.6.1. Log-Likelihood Ratio

The likelihood function of a Gaussian random variable described by μ and σ^2 for a data vector $\mathbf{x}$ of length N is

$$\mathcal{L}\left(\mathbf{x}; \mu, \sigma^2\right) = \left(\frac{1}{2\pi\sigma^2}\right)^{N/2} \exp\left(\frac{-1}{2\sigma^2}\sum_{n=0}^{N-1}(x_n - \mu)^2\right). \tag{5.14}$$

Equation (5.5), after substituting in the estimated parameters, becomes

$$\frac{\mathcal{L}(\mathcal{H}_1)}{\mathcal{L}(\mathcal{H}_0)} \approx \frac{\mathcal{L}\left(\mathbf{x}; \hat{\mu}_{\text{buff}}, \hat{\sigma}^2\right)}{\mathcal{L}\left(\mathbf{x}; \hat{\mu}_{\text{bg}}, \hat{\sigma}^2\right)} = \frac{\left(2\pi\hat{\sigma}^2\right)^{-N/2}\exp\left(\frac{-1}{2\hat{\sigma}^2}\sum_{n=0}^{N-1}\left(x_n - \hat{\mu}_{\text{buff}}\right)^2\right)}{\left(2\pi\hat{\sigma}^2\right)^{-N/2}\exp\left(\frac{-1}{2\hat{\sigma}^2}\sum_{n=0}^{N-1}\left(x_n - \hat{\mu}_{\text{bg}}\right)^2\right)} > \tau \tag{5.15}$$

We now take the logarithm of both sides and simply to get

$$\frac{N}{\hat{\sigma}^2}(\hat{\mu}_{\text{buff}} - \hat{\mu}_{\text{bg}})^2 > 2\log(\tau). \tag{5.16}$$

5.1.6.2. Threshold

We can choose $\log(\tau)$ in multiple ways, and we present three of them below.

Neyman-Pearson
A traditional way to compute the threshold is using the Neyman-Pearson method. This method sets a threshold for a constant false-alarm rate by solving

$$P_{\text{FA}} = \Pr(\mathcal{H}_1|\mathcal{H}_0) \tag{5.17}$$

$$= \Pr\left(\frac{N}{\hat{\sigma}^2}(\hat{\mu}_{\text{buff}} - \hat{\mu}_{\text{bg}})^2 > 2\log(\tau) \,\middle|\, \hat{\mu}_{\text{buff}} = \hat{\mu}_{\text{bg}}\right) \tag{5.18}$$

for $\log(\tau)$. However, this requires being able to accurately characterize the distribution of the left-hand-side of Equation (5.16). In this case, it is difficult to characterize that distribution. There is no simple closed-form solution, so we will not consider it further.

Bayes Risk
A different manner to compute the threshold is using a method called Bayes Risk. With this method, we assign a cost to each type of error (either a false positive or a false negative). We define a cost matrix $\mathbf{C}$, where C_{ij} is the cost if our algorithm selects $\mathcal{H}_i$ when $\mathcal{H}_j$ is true–in other words the cost of a mistake. Typically, the diagonal of $\mathbf{C}$ is 0, and we can define the other costs however we would like, either equivalently (such that it is equally bad to have a false positive as a false negative) or weighted to penalize one type of error more heavily.

Bayes Risk is the expected cost of the binary decision:

$$\mathcal{R} = \mathbb{E}(\mathbf{C}) = C_{01}\Pr(\mathcal{H}_0|\mathcal{H}_1)\Pr(\mathcal{H}_1) + C_{10}\Pr(\mathcal{H}_1|\mathcal{H}_0)\Pr(\mathcal{H}_0). \tag{5.19}$$

We seek to select a threshold for detection that minimizes $\mathcal{R}$. In order to do that, we set

$$\frac{\mathcal{L}(\mathcal{H}_1)}{\mathcal{L}(\mathcal{H}_0)} > \frac{(C_{10} - C_{00})\Pr(\mathcal{H}_0)}{(C_{01} - C_{11})\Pr(\mathcal{H}_1)} = \log(\tau), \tag{5.20}$$

where τ can be inserted into (5.16) [14].

$\Pr(\mathcal{H}_i)$ is called a prior. A prior outlines ones belief in a hypothesis *before* any data is received. This method assumes an accurate prior for both hypotheses. Since this is typically not feasible for detection applications, particularly anomaly detection, we do not implement this method in the system.

Heuristics
The final method for threshold computation is via heuristics. As mentioned in the previous paragraphs, there are more ideal ways to compute the desired threshold, but they tend to be

unsolvable either because the distribution is not closed-form or because we do not have good priors for the different hypotheses. As such, we have selected thresholds based off heuristics found by numerous simulations across a variety of scenarios.

5.1.7. Time-of-Arrival Estimation

As will be shown in section 5.1.9, the detection algorithm does well at *detecting* events. However, it does a poor job of estimating the onset of certain small events, at least as-is. This is because we use a buffer for estimation – while the use of the buffer facilitates detection of very small events, it also often requires multiple time steps of the event to be registered in the buffer before the mean estimation of the buffer changes enough to cause an event detection. Along the same lines, we also require multiple threshold crossings in a row before we declare an event was detected. As such, when the algorithm declares an event, we are often multiple time steps after the event occurred. Figure 5-4 displays the buffer of a pixel as an event is declared by our detection algorithm. As can be seen, the event starts around time sample 70, 30 time samples before the event is declared by our algorithm (which occurs at time step 100 in the figure, at the end of the buffer).

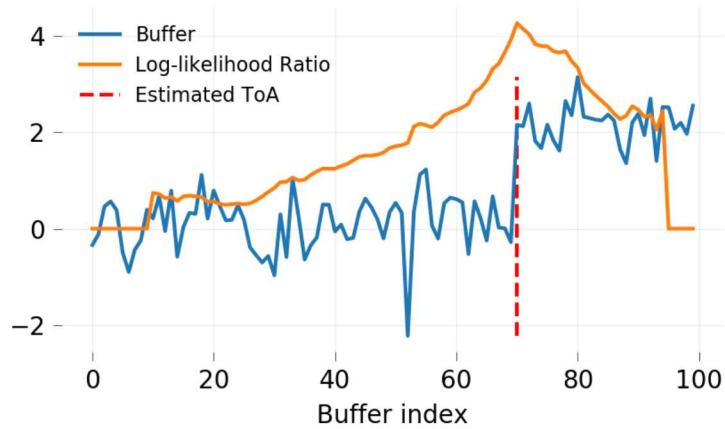

Figure 5-4. Buffer at time of event detection. As can be seen, the buffer has several time samples from the event before the event is declared. As such, if we were to time-tag the onset of an event at the time sample the event is detected, it would be late. Therefore, we use the log-likelihood ratio to better-inform our time-of-arrival estimate.

Since we have the buffer, we can do much better on our TOA estimation with a Bayesian update step. This step is another change detection algorithm, almost identical to the first, for only the values in the buffer. We now know that an event occurred. As such, we would like to isolate exactly where in the buffer the event occurred, or which time step the distribution changes. Therefore, we would like to find the time sample n_0 where

$$\mathcal{L}\left(\mathbf{x}; \mu_{\text{bg}}, \mu_{\text{evt}}, \sigma^2\right) \sim \begin{cases} \mathcal{N}\left(\mu_{\text{bg}}, \sigma^2\right), \forall n < n_0 \\ \mathcal{N}\left(\mu_{\text{evt}}, \sigma^2\right), \forall n \geq n_0 \end{cases}. \tag{5.21}$$

The time-of-arrival is thus $n_0 = \arg\max_n \mathcal{L}(\mathbf{x}; \hat{\mu}_{\text{before}}, \hat{\mu}_{\text{after}})$, where $\hat{\mu}_{\text{before}}$ is the mean of all time samples before n_0 in the buffer and $\hat{\mu}_{\text{after}}$ is the mean of all time samples including and after n_0 in the buffer. A scaled log-likelihood computation is shown in figure 5-4. As can be seen, we select the time index with the peak log-likelihood as the TOA, which greatly reduces the TOA error.

5.1.8. Implementation Details

5.1.8.1. Mathematical Updates

We need to update the estimates for the buffer parameters (mean and variance) at every time sample. We also would like the algorithm to be able to handle real-time streaming data. As such, we need these updates to be fast. In this subsection, we address the methods we have employed to reduce computational complexity and increase speed in these estimates.

Given an estimate for $\hat{\mu}_{\text{buff}}$ from the last time sample ($\hat{\mu}_{\text{buff},1}$), it is simple to update the estimate for this time sample ($\hat{\mu}_{\text{buff},0}$) given new data. In the simplest case, we would like $\hat{\mu}_{\text{buff},0} = \hat{\mu}_{\text{buff},1} + \hat{\mu}_{\text{update}}$. The mean estimate for the incoming data in the buffer is updated by

$$\begin{aligned}\hat{\mu}_{\text{update}} &= \hat{\mu}_{\text{buff},0} - \hat{\mu}_{\text{buff},1} \\ &= \frac{1}{N}(x_{-1} + x_{N-1}),\end{aligned} \tag{5.22}$$

where x_{-1} indicates the incoming time sample (not stored in the buffer yet for the mean estimate), and x_{N-1} is the value in the last slot of the buffer (the oldest value in the buffer, which will be rotated out when the most recent data is stored into the buffer). It is important to note that we compute the mean estimate *before* adding the newest data to the buffer, as this is the only way we have the oldest data still stored in memory. The estimate $\hat{\mu}_{\text{bg}}$ is similarly updated, but instead of using data from the data buffer, we use the newest estimate and the oldest stored values from the $\hat{\mu}_{\text{buff}}$ buffer. These, however, are only updated every K time steps, as already discussed.

The estimate $\hat{\sigma}^2$ is only slightly more complicated. We use

$$\hat{\sigma}^2_{\text{update}} = \hat{\sigma}^2_{\text{buff},0} - \hat{\sigma}^2_{\text{buff},1} \tag{5.23}$$

$$\begin{aligned}&= \frac{1}{N-1}\Bigg(x_{-1}^2 - x_{N-1}^2 - 2\hat{\mu}_{\text{buff},0}x_{-1} + 2\hat{\mu}_{\text{buff},1}x_{N-1} \\ &\qquad - 2(\hat{\mu}_{\text{buff},0} - \hat{\mu}_{\text{buff},1})\sum_{n=0}^{N-2} x_n + N\hat{\mu}^2_{\text{buff},0} - N\hat{\mu}^2_{\text{buff},1}\Bigg)\end{aligned} \tag{5.24}$$

This is the only parameter we update in this manner, as the rest are averages (either of the data or the different buffer parameters).

5.1.8.2. Detection Logic

Previously, we have stated that a "detection" occurs when the right-hand-side of equation (5.16) is greater than the left-hand-side. From a system perspective, this is not quite true. To reduce false alarms, we require that an event be present for multiple time samples in a row before an official detection is declared. A "detection" occurs when the right-hand-side of Equation (5.16) is greater than the left-hand-side for L time samples in a row for a given Haar coefficient, where L is typically somewhere between 2 and 5. It is important to note, though, that the time of the very first threshold crossing is marked as the time of arrival, as opposed to the Lth threshold crossing, as this provides an estimate of time-of-crossing with relatively little impact from the delays in the change detector.

5.1.8.3. Update Decision Logic

There is a nontrivial decision process as to when to update the various parameter estimations in the system. We get new incoming data on each time step. This data, representing the total number of spikes over the time interval, is used to update $\hat{\mu}_{\text{buff}}$ and $\hat{\sigma}^2$ each time step (see equations (5.22) and (5.23)). Likewise, the internal data buffer is updated with the new data, shifting out the oldest data at each time step.

We do not update the estimates for the background parameters ($\hat{\mu}_{\text{bg}}$ and $\hat{\sigma}^2$) at each time step. We update them *at most* every K time steps, which is a user-specified parameter. We do not update every time sample since we are assuming the background is slowly varying and thus does not require an update every time sample. However, there are times when we do not update the background, even at the Kth time sample. As soon as any event is detected, we freeze the background until the event is released. This means that, if there is an event being tracked anywhere on the array, we do not update the background anywhere on the array. We allow the background to begin updating as soon as all events are removed.

After the buffer parameters are estimated as described above, we compute equation (5.16), comparing the left-hand-side to the right-hand-side. If there is no detection, and we are not tracking an event that previously occurred on the array, then we do not freeze the array and it is free to update the background parameters every K time samples. However, if the right-hand-side is greater than the threshold, meaning a detection, *or* an event has previously occurred somewhere on the array and is currently being tracked by the system, then the array is frozen, and even if it is the Kth time sample, the background parameters and buffers are not updated.

5.1.8.4. Event Removal

Once a Haar coefficient achieves the L threshold crossings in a row to constitute an official detection, an event is created. This event is tracked by the system going forward. The event is only removed once the Haar coefficient no longer achieves threshold crossings for P time samples in a row, typically 10. Once the Haar coefficient has become sufficiently close to the background that

it is no longer distinguishable for P time samples in a row, the event is released. If no events are present, the background is able to start updating again.

5.1.8.5. Array Detection

To save computation, we do not run the algorithm over all of the Haar coefficients. Instead, we focus on just the coefficients most sensitive to change – the c_D coefficients in the lowest layer. While we acknowledge that the lowest c_H and c_V coefficients are also sensitive to change, we refrain from using these quadrants for detection at this time so that we do not need to mitigate the fact that we could have multiple detections from the same event (at least one in each quadrant).

While our algorithm works well (see section 5.1.9), we throw away a lot of usable data by only using the lowest c_D coefficients, as well as degrade our centroid-estimation ability. It is also possible take the inverse Haar wavelet transform and reconstruct the preprocessed time-series, and then run the detection algorithm on this new time series. We have found that if we reconstruct the signal with only the lowest level of the Haar coefficients (c_H, c_V, and c_D) and zero out all other coefficients, we get better results. While we have not done a deep-dive into why this is the case, we hypothesize that, by only reconstructing using the lowest level of the Haar coefficients, we are essentially removing the background noise induced at the higher levels of the transform. In this way, we can focus on only the high-frequency variations of the signal, which is what we really care about for our change detection. In this case, we run our detection algorithm over the entire array instead of just a quarter of the array.

5.1.9. Results

We are interested in analyzing the performance of our detection algorithm against very low-intensity events. As such, in this section we discuss an experiment we designed, executed, and analyzed to determine our probability of detection and false alarm against very small signals.

5.1.9.1. Experiment Overview

We are interested in characterizing the performance of our system in terms of probability of detection and false alarm versus event SNR. Do determine this, we define a constant background on a 16×16 SPARR array. Then, we subdivide between 1% and 6% of this background in 100 linear steps. For each step, we randomly generate 20 runs each with one event of that amplitude, located in the center of the array. Because some of our algorithm parameters exceeded 1% of background, we added additional runs below the 1% mark to ensure we captured the algorithm down to the failing point.

Every simulation was 0.6 seconds with the event starting at 0.3 seconds and ending at 0.5 seconds. The start and end of the event were set to be step functions. For all simulations, a pixel spike occurs after 7500 photon arrivals for each pixel. The remainder of the algorithm parameters can be found in tables 5-1 and 5-2.

Table 5-1. Thresholds used for each algorithm.

Algorithm	Threshold ($2\log(\tau)$)
Inverse Haar 25 kHz	73
Inverse Haar 50 kHz	105
c_D 25 kHz	50
c_D 50 kHz	50

Table 5-2. Algorithm parameters.

Frequency (kHz)	Number of detections in a row	Buffer size	Mean buffer size	Minimum settling time (scans)	Minimum skips before new event (scans)
25	5	100	40	2000	2000
50	5	100	40	4000	4000

5.1.9.2. Experimental Results

Since it is possible for the algorithm to declare a detection on the wrong pixel or during the wrong times, we define a true detection as one which occurred on the correct pixel between 0.3 and 0.5 seconds. We define a false detection as one which has a detection on any pixel outside the specified time window of 0.3 to 0.5 seconds or on any incorrect pixel during that time window. However, due to the point spread function (PSF) of the optics, it is possible for energy to smear onto neighboring pixels. As such, we do not penalize for detections on the pixels immediately surrounding the pixel on which the event is centered. Those detections are treated neither as false alarms nor true detections in our computation.

In this experiment, we explored detecting at 25 kHz and 50 kHz in the Haar domain (lowest c_D), as well as 25 kHz and 50 kHz in the reconstructed domain. The results are shown in figure 5-5. As a general trend, performance is better in the reconstructed domain, as well as when we use a lower frequency. We benefit if we take the computational time to invert the Haar transform and operate over the entire array. It is important to note that the thresholds used to generate this figure were heuristically chosen such that there were zero false alarms. As such, we could select different thresholds to improve detection performance at the expense of more false alarms.

Finally, we analyze our time-of-arrival accuracy, or our ability to time-tag the start of an event. Figure 5-6 shows the error (in μsec) of the TOA estimation compared to the true start time of the event for detection in the inverse Haar plane at 25 kHz. As can be seen, for events 2% of background and larger, we have almost no error in estimating the time-of-arrival of an event (this is limited to the 25 kHz resolution). For targets between 1.5% and 2% of background, we appear to have some spread in the time-of-arrival error, and this spread tends to be negative, meaning that we estimate that the event starts *before* it actually does.

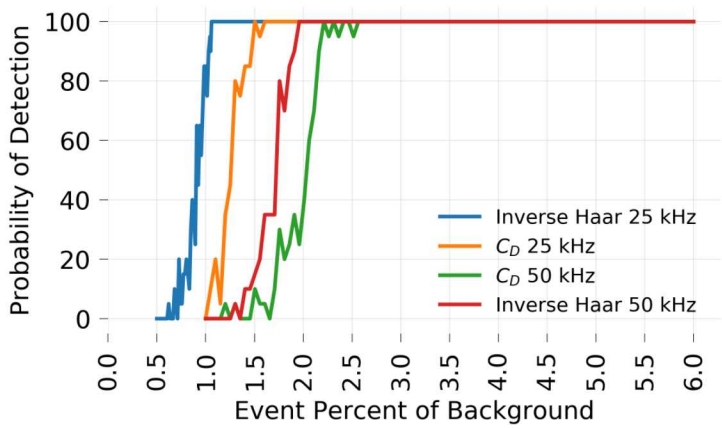

Figure 5-5. Probability of detection given different parameters for the detection algorithm. As can be seen, we perform best when we take the time to invert the Haar transform. Generally, detection performance decreases as the time bin frequency increases.

This is due to the Bayesian time estimation step. The overall change detection algorithm observes distributions that are fit from large amounts of data (the parameters are estimated using MN samples for the background, and N samples for the buffer), which results in low-variance estimates. As such, the detection algorithm can tell the distribution has changed for small changes. However, the Bayesian time estimation step only sees the current buffer. Furthermore, it must fit two distributions from the buffer, meaning that it is using (potentially significantly) less data points than N to estimate these parameters. As such, it struggles to identify the change point in the buffer. For this reason, for small signatures, where the distribution on one side of the event to the other is not significant, our time estimation is roughly a uniform random variable across the buffer length. Since the start of the event is towards the tail end of the buffer (we do not need to populate the entire buffer with time samples including the event to change the distribution significantly enough for the change detection algorithm to detect it), this biases the time estimation to estimate a TOA before the actual event occurs.

An example of this can be seen in figure 5-7. This figure shows the buffer for a particular pixel at the time sample when an event at that pixel occurs, the computed log-likelihood ratio across the buffer, the estimated start time of the event, and the true start time of the event. The start time of the event is not immediately obvious. Furthermore, we zero out the log-likelihood ratio computation at the beginning and the end of the buffer due to too few time samples to accurately estimate the distributions. As can be seen, there does seem to be a peak in the log-likelihood ratio at the time of the true TOA. However, this peak is below 0, so it is not registered as the maximum in the system. We could solve this problem by reducing the value of the "invalid" time samples. However, this default value is essentially a threshold for how confident we must be in order to declare a TOA. Another solution would be to default report the time of the end of the buffer, rather than the beginning of the buffer. This would reduce the number of signals that are time-tagged before the event, changing them to be time-tagged after the event. However, we felt that it is best to estimate early, rather than late.

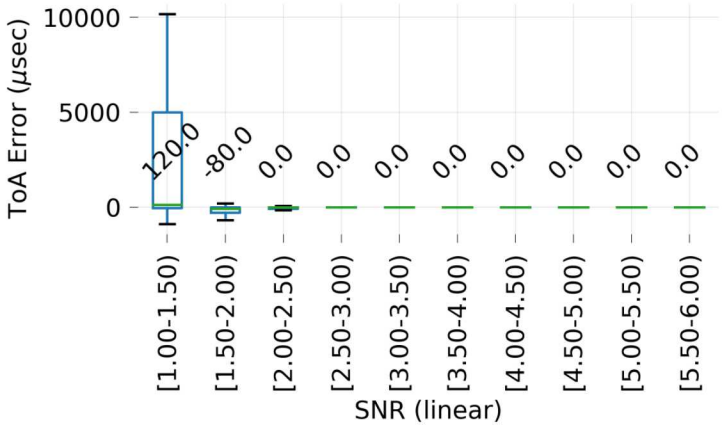

Figure 5-6. TOA estimation error (in µsec). As can be seen, for targets above about 2% of background, we have almost no error (due to the TOA Bayesian update step). However, for targets in the 1.5%-2% of background, the algorithm tends to time-tag the arrival early. Finally, for signatures smaller than 1.5% of background, we have large errors in the time-tag. This primarily occurs because, for these small signatures, the algorithm actually detects the *end* of the event (the falling edge) as opposed to the start of the event.

For signatures below about 1.5% of background, we tend to have much larger TOA errors. The reason for this is simple: we are triggering on the *removal* of the event, rather than the start. As can be seen in figure 5-8, the log-likelihood ratio of the rising edge is not enough such that a "detection" occurs at the onset of the event. The event then bleeds into the background throughout the event's life. Finally, when the event is removed, we detect the offset of the event. Therefore, we detect the event, but we time-tag the event as occurring at the end of the event, rather than the beginning.

The top pane of figure 5-8 shows the background and buffer mean estimates across time. The middle pane shows the log-likelihood ratio across time, as well as when a detection occurred. The bottom pane shows the processed signal across time.

5.1.10. Future Detection and TOA Algorithm Work

The algorithms presented here perform well; however computing detection on a per-spike basis, rather than binning the data across time and then computing detections may offer better performance where the Gaussian approximation is not good. That would include very low spike rates or other in-array signal processing. Per-spike calculations are difficult to implement because of the non-trivial statistics of the inter-arrival times of spikes after the Haar transform and the quadtree readout. A mixture model and its associated computational framework would address the non-Gaussian character of the spike intervals.

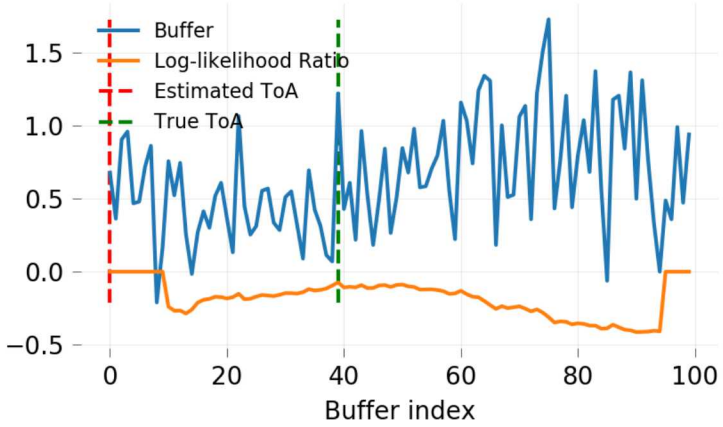

Figure 5-7. ToA estimation for a signal that is 1.5% of background. Here we see the data in the buffer when the change detection occurs, as well as the log-likelihood ratio, the estimated time-of-arrival, and the true time-of-arrival. As can be seen, the log-likelihood ratio never surpasses 0, so we do not accurately detect the start of the event.

Another improvement would modify the Bayesian time estimation step to operate on a per-spike basis, rather than a per time sample basis. At the cost of greater computational resources, the TOA error could be reduced.

Currently, the algorithm uses only the Haar coefficients from the first level of the transform to reconstruct the signal prior to detection. Events occupying larger areas–perhaps due to optical ghosts or scatter–may be better detected using more, or different, levels of the transform in reconstruction.

The algorithms used here do not account overtly for background changes due to scene motion. Incorporating space-time change, as opposed to just per-pixel time, would benefit practical applications of the sensor. Space-time change detection, and background subtraction, would benefit sub-pixel centroid estimation, useful for improved angles of incidence estimation.

5.2. OBJECT TRACKING

This chapter explores the application of a multi-target tracking algorithm to SPARR's output.

The best detection and tracking would follow a "track-before-detect" approach – treat every spike from the array as a "detection" and allow a tracking algorithm to determine where the relevant tracks are. However, spike speeds in the MHz to GHz range per pixel, along with potentially millions of pixels, would overtax current implementations of modern algorithms, such as multiple hypothesis tracker (MHT). The MHT algorithm, although essentially off-the-shelf, is a state-of-the art tracker and is too computationally expensive to operate on each spike.

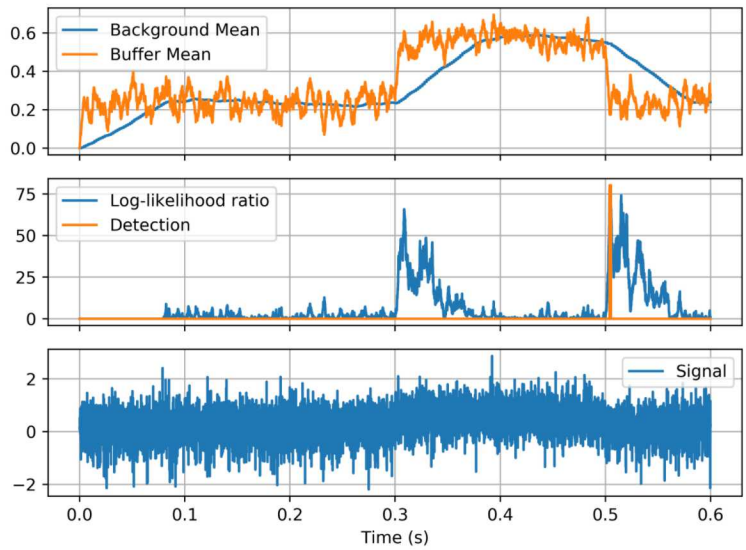

Figure 5-8. An example of tail trigger. The algorithm missed the onset of the event, instead triggering on the end of the event.

To ameliorate the computational burden, first run the detection algorithm described in Section 5.1 and feed those detections to the tracker. The detection algorithm is quite flexible; it can be configured to generate detections frequently with low thresholds which permits highly granular tracking.

The reader should consult the extensive literature on MHT for a deeper understanding. MHT is well documented in, for example Blackman[3]. Our tracking work models a sky-viewing telescope moving along a known path centered on a satellite, causing the background stars to pass linearly across the field of view. To apply the MHT algorithm, section 5.2.1, describes the experimental simulation environment, then sections 5.2.2 and 5.2.3 show the results of applying detection and tracking algorithms on the resulting SPARR data. Finally, section 5.2.4 outlines suggested future work for object tracking with SPARR.

5.2.1. Experiment

In the simulation, SPARR acts as a ground-based sensor following a satellite in the sky. Throughout the simulation, the followed satellite remains fixed near the center of the FPA. Stars streak through the sensor's field of view during the simulation.

Figure 5-9 shows the paths of different objects over time. The blue square represents SPARR. The sensor is simulated as a 256×256 pixel array. Each red circle indicates where the star is located at the beginning of the simulation; all stars begin on the left edge of the field of view. The red streaks show the stars' simulated trajectories, while the size of the red circle corresponds to the optical intensity (apparent magnitude) of each star. Each star's motion is linear throughout the five second simulation.

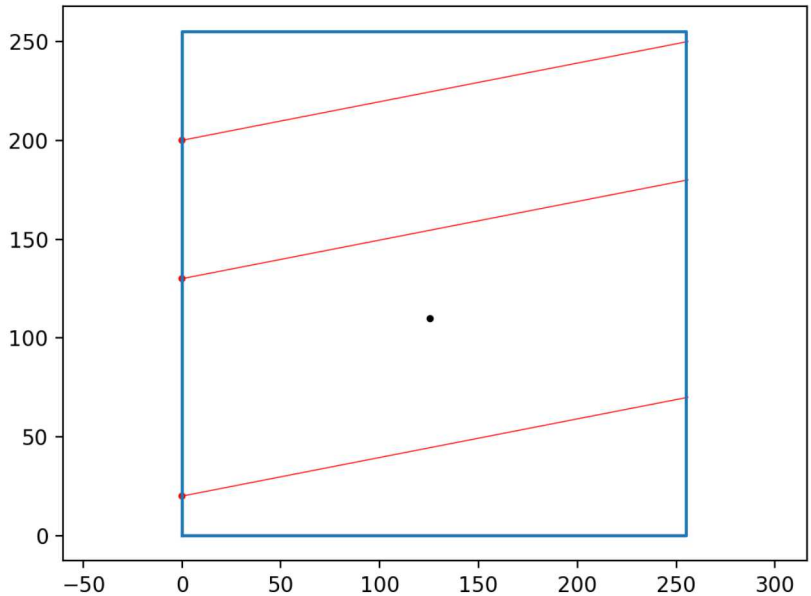

Figure 5-9. Star paths cross the sensor. The blue square is the sensor, the black circle near the middle of the square is the satellite being followed, and the red circles/streaks are stars that travel in (and out) of the field of view of SPARR.

5.2.2. Detection

The change detection algorithm described in section 5.1 is configured to detect, per time-step, if something of interest is located in any particular pixel rather than to detect and estimate TOA for fast-rising transients. The first modification declares a detection whenever a threshold crossing occurs rather than requiring multiple consecutive detections. Second, we reduced the size of the buffer to five samples, and the averaging buffer to only three samples because staring into the night sky generates uniformly dark background, at least as far as this simulation is concerned. With a dark background, almost any spikes are important and the background is almost uniformly unimportant, so we can reduce the complexity of background estimation. Furthermore, the tracking algorithms act to reduce the number of false detections that become objects of interest. Associating detections together with other detections or with existing tracks while ignoring false alarms is the tracking algorithm's work, not the detection algorithm's. Think of the detection algorithm as reducing the amount of data for the tracker – rather than tracking on every spike.

Figure 5-10 shows a three-dimensional representation of the detection algorithm's output. The x and y axes refer to the location of the detections on the SPARR array, while the vertical (z) axis refers to the time index of the detection. The detection algorithm's relaxed configuration frequently generates exceedances, as desired. The tracked satellite in the center of the focal plane is obvious as a dark cylinder down the middle of the image. The stars streaking through the FPA

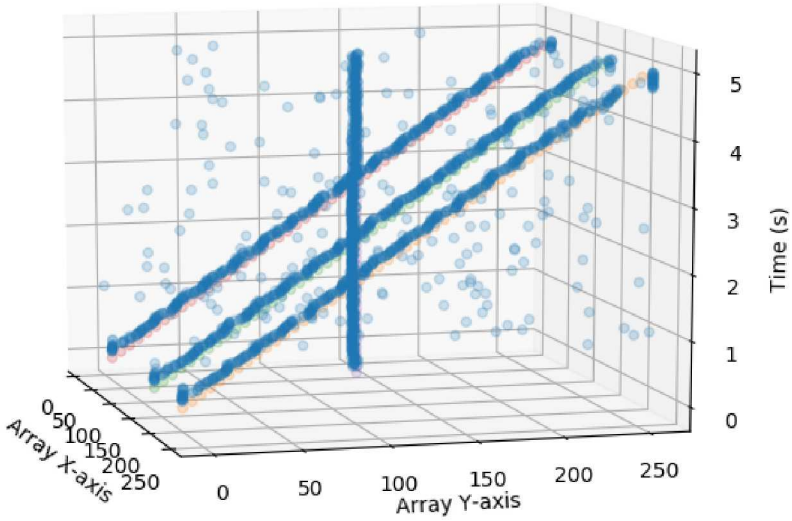

Figure 5-10. Output from the detection processing that feeds the tracking algorithm.

are also visible as dark cylinders. We employ MHT to convert these messy detections into exploitable tracks, as well as to reduce the background noise.

5.2.3. Multiple Hypothesis Tracker

The detection data visualized in figure 5-10 passes through an MHT algorithm to produce the results in figure 5-11. The figure displays the SPARR array, with a black circle on the plot marking each time step.

The satellite is obvious as a black mass of detected points near the array's center. Since the satellite is not moving relative to the array during the simulation, the mass of detections is expected. The circumference of the cloud is indicative of measurement error combined with error due to Kalman filter propagation when no measurement was available.

The stars streaking through the simulation appear as black streaks across the figure. The MHT algorithm connects events together to extract the path of each star and almost completely eliminates false detections–there are no false tracks output from the algorithm. Furthermore, the tracker reproduces the trajectory of each star, each track follows a linear path across the sensor in the simulation, matching the stars. In a few cases, the star locations at a given time sample proposed by the MHT appear as deviations from the linear path. These deviations are due to Kalman filter motion propagation errors; however, they are quickly corrected.

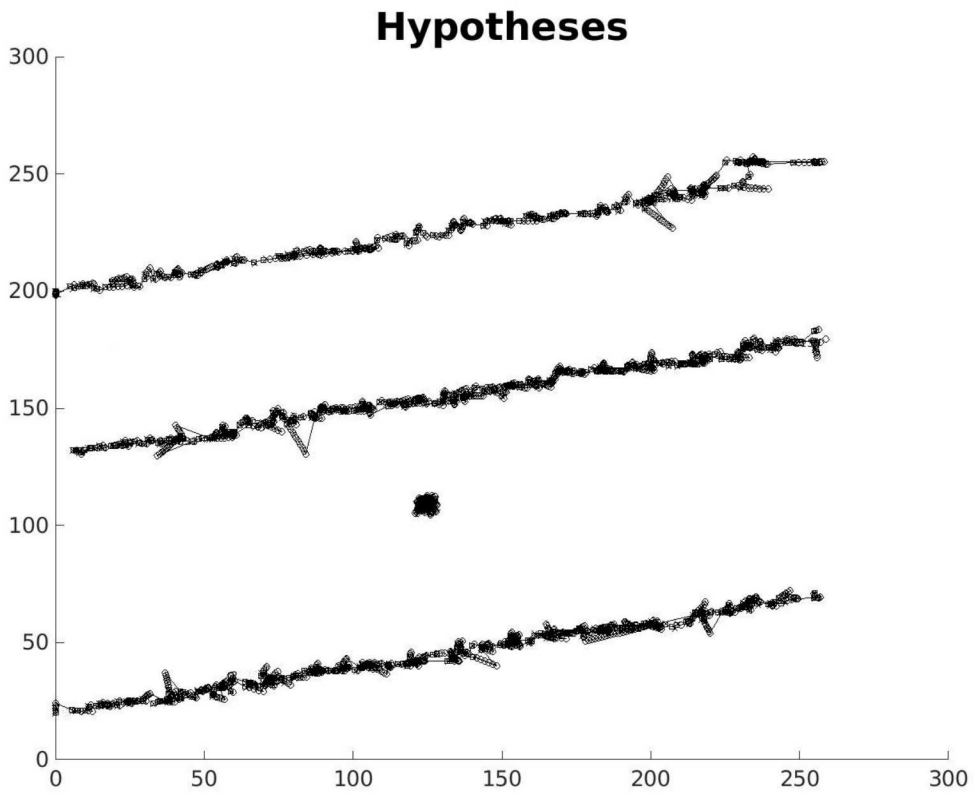

Figure 5-11. Output from MHT. The frequent detections associate with the stars and the followed object in the center. Additional noisy detections appear as random markers throughout the volume.

5.2.4. Future Tracking Work

While applying an off-the-shelf tracking algorithm to SPARR output certainly has its appeal, projecting SPARR into the world of "detect-then-track" and framing the data does not take advantage of the spiking output from SPARR. Future work should revolve around developing tracking algorithms that use the native asynchronous SPARR output, pre-detection.

Finite-Set Statistics-based trackers are one promising technology. These trackers, such as Bernoulli filters[29], explicitly model the appearance and disappearance of targets. We believe leveraging this capability could help track on a per-spike basis.

Spiking artificial neural networks offer another promising technology. There are limited mature methods for training spiking neural networks, and methods will be needed to learn and model complex, high-speed dynamic systems like those required to track on a per-spike basis from SPARR. However, this technology should allow processing spikes directly, without framing in time.

5.3. IMAGING WITH SPARR

Like many sensors with an FPA, images can be created using the SPARR outputs. That is, the spikes representing the translation, throttled Haar transform, and quadtree readout of an image can be transformed back into an image. Of course, the spikes that were lost in the interactions in the Haar state machines and the decimated spikes in the quadtrees cannot be exactly recovered, but the inversion is still possible in a statistical sense. That is, the Haar throttling and quadtree decimation are known, so they can be accounted for in the inversion process by appropriately weighting individual spikes. There are two caveats. The first is that enough spikes must be used for the resulting image to accurately reflect the optical image. As more spikes arrive at the SPARR readout, the reconstructed image quality will improve. The second caveat is that spike collisions lose information. Spike collisions are something like the *dead time* in energetic particle counters. It may be possible to perform a statistical correction for that dead time, though particle arrivals are Poisson whereas spikes from SPARR follow more complicated distributions. We must rely on the pixel capacitor well sizes, the quadtree decimations, and an intelligent hardware layout to minimize spike collisions.[1]

It would be possible to reverse the Haar transform with spike-based processing, either inside SPARR (before the quadtree) or in spike-based off-array hardware. This approach would add significant complexity and we did not consider it in any detail. Instead, we assume the quadtree readout spikes are transmitted to an off-array system that collects spikes for some amount of time and uses that collection of spikes to create an image. We will show with an example how the error for the image reconstruction improves as more spikes are collected from the readout, and we will show the effect of decimation on the reconstruction.

5.3.1. Imaging Example

To demonstrate SPARR's ability to create an image, we will use the image of the earth shown in figure 5-12. This image is 512×512 and is normalized so that the brightest pixel has a value of unity. In this way, each image pixel represents the likelihood a photo-electron will arrive in a specific interval in an ASP inside `sparrow`'s model. We set the base rate of photon arrivals to 10^9 photons per second and the pixel electron well size to 5000 electrons, then ran the simulation for 0.05 seconds to generate the final result. This simulation was repeated eight times where each simulation used a constant, but different, amount of decimation in the quadtrees. We used decimations of 1:1 (i.e., no decimation), 2:1, 4:1, and so on until 128:1. The random number seed was constant across all simulations so that the same pixel spike times were used in every simulation.

[1] The hardware layout is important because each pixel spike and approximation Haar spike (except for the apex Haar node) is fed into four different state machines. It is possible for two or more of these state machines to output a Haar spike at nearly the same time, which are then input to a quadtree node. If the spikes at nearly identical times enter the same quadtree node, there will be a collision leading to at least one lost spike no matter how well the remainder of the system is designed.

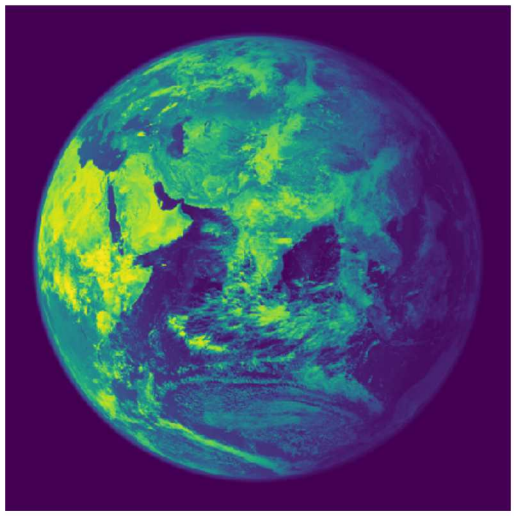

Figure 5-12. Image of the earth.

Figure 5-13 shows the percent error of the various reconstructions as a function of the number of spikes used in the reconstructions. Here, we have defined percent error as

$$\text{percent error} = \frac{\left\|X - \hat{X}\right\|_F}{\|X\|_F} \times 100 \tag{5.25}$$

where $\|\cdot\|_F$ denotes the Frobenius norm, which is defined by

$$\|X\|_F = \sqrt{\sum_i \sum_j X_{ij}^2}. \tag{5.26}$$

This is sometimes called the percent root-mean-square difference (PRD) and can be considered an extension of the usual Euclidean norm for vectors. In figure 5-13, we see that the error generally decreases as the number of spikes increases. The trend is not monotonic because the distribution of spikes at the readout is only approximately the true distribution. That is, spikes that make up a dim pixel could arrive before spikes that make up a bright pixel, which would momentarily decrease the contrast between those two pixels in the reconstruction. If the scene did not change, however, the distribution of spikes would approach the true distribution as time approached infinity.

Figure 5-13 also shows that number of spikes is more important than the decimation level for determining the percent error. The exception to this is the error for the simulation with no decimation decreases faster than the other simulations as the number of spikes increases. This may also be true for higher decimations, but the figure is inconclusive on this (viz., the 2:1 decimation line may be starting to fall off faster than the higher decimations, but not enough spikes were generated to know for sure).

This figure shows that the percent error can be made small as long as enough readout spikes are given, which is exactly what we would expect. Although time is not shown in figure 5-13, all eight

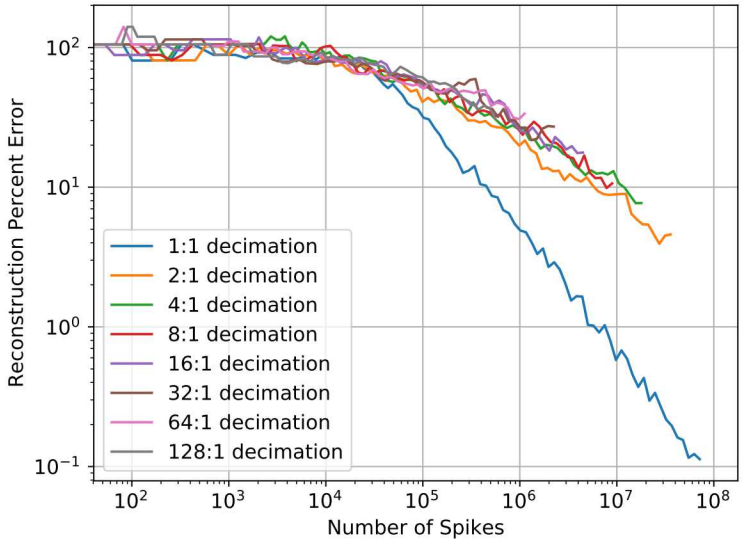

Figure 5-13. Percent error for each decimation as a function of spike count.

simulations were for 0.05 seconds, and we see that the lines for higher decimations end sooner than lines for lower decimations. This means that if the quadtrees are using large decimations, more time is required to obtain an image of some desired quality than if less decimation was used. If the image is static or close to static for the time it takes to collect the spikes, then more time is not an issue. This is akin to shutter speed for an ordinary camera where a higher decimation corresponds to a slower shutter speed. Too much movement in the time this process takes will result in a blurry image.

To see the quality of the reconstructed images, we show the resulting image from the end of each simulation (i.e., 0.05 seconds). The reconstructions and the computed percent errors for the 1:1, 2:1, 4:1, and 8:1 decimations are shown in figure 5-14, and the reconstructions for the 16:1, 32:1, 64:1, and 128:1 decimations are shown in figure 5-15. The number of quadtree spikes used in these reconstructions is shown in table 5-3 along with the number of spikes per pixel. Qualitatively, there is no meaningful difference between the 1:1 decimation reconstruction and figure 5-12, and there is very little difference with the 2:1 decimation reconstruction. A few small errors become visible for the 4:1 decimation reconstruction, but more serious errors appear in the 8:1 and 16:1 decimation reconstructions. However, the visual artifacts are still minimal and likely would not affect any application of the image. Table 5-3 shows that this requires about 17 spikes per pixel, which is much better than the 273 spikes per pixel required for the 0.1% error reconstruction. The reconstructions for the 32:1 and higher decimations show much more serious artifacts, but even these may be tolerable for some applications. If 30% error is tolerable, then table 5-3 shows that only two spikes per pixel are necessary for a reconstruction.

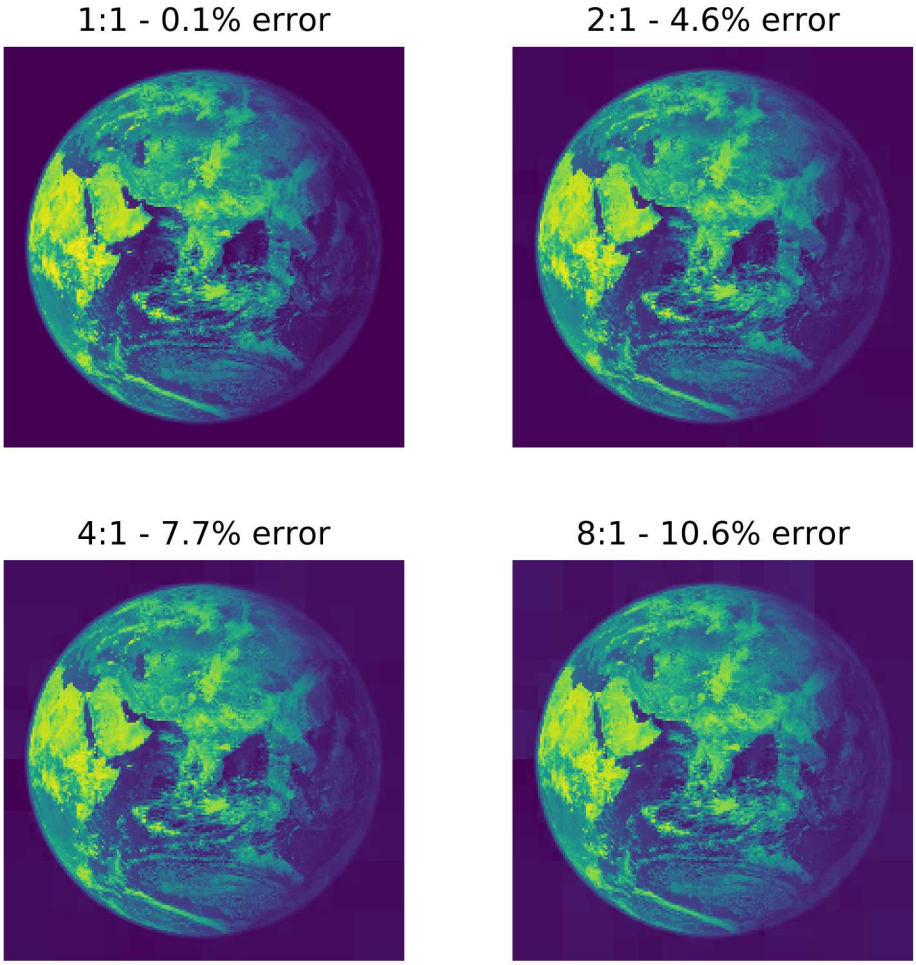

Figure 5-14. Image reconstructions for the 1:1, 2:1, 4:1, and 8:1 decimations at 0.05 seconds.

5.3.2. Conclusion

We have demonstrated that SPARR can be used to generate usable images in spite of the signal processing present in the throttled Haar transform and the decimation present in the quadtrees. As long as spike collisions are minimized, usable images are available for as low as 2 spikes per pixel. For our example, this resulted in about 30% error, but if this is intolerable, more spikes can be collected at the quadtree readout until the desired error is achieved. The only additional requirement is that the time period is not so long that the background appreciably changes, which would cause blur. The future work (section 6) discusses performing image reconstruction differently at different coefficient levels; high levels of the Haar transform represent large areas, which typically will not move appreciably in a short time. Improved reconstruction or estimation–rather than just weighted spike counting–would certainly improve the image reconstruction. However, this is true of any imaging camera, so it is not an unusual requirement. It

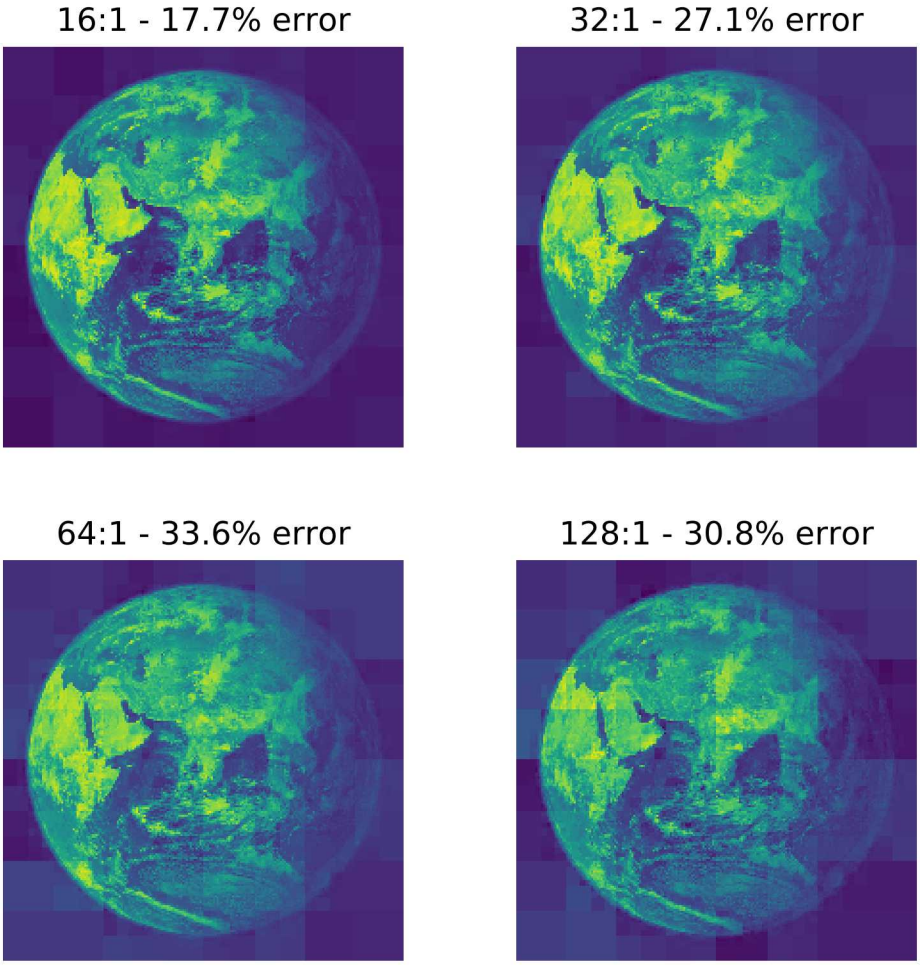

Figure 5-15. Image reconstructions for the 16:1, 32:1, 64:1, and 128:1 decimations at 0.05 seconds.

is similar to shutter speed requirements on a camera or clocking requirements on a camcorder.

Table 5-3. Number of spikes used for the image reconstructions.

Decimation	Number of Spikes	Spikes per Pixel
1:1	71 571 428	273.0
2:1	35 785 736	136.5
4:1	17 892 893	68.3
8:1	8 946 471	34.1
16:1	4 473 264	17.1
32:1	2 236 661	8.5
64:1	1 118 355	4.3
128:1	559 203	2.1

6. FUTURE WORK

SPARR is an unusual sensor; not directly producing framed or row-by-row data. It offers time resolution as fine as the switching time of the circuits, typically around one nanosecond. However, data is not available from every pixel at that rate. SPARR has sparse but highly precise time resolution. We are aware of no other sensor that can stream data continuously with high spatial resolution, low latency, and precise time accuracy.

A consequence of its unusual place among sensors is a dearth of processing that takes advantage of its unusual characteristics. During this research we defined detection by change to the time-binned spikes coming from a pixel or from a coefficient (see section 5.1), but this ignored the fine time resolution. In essence, integrating spikes over time windows treats the sensor as a rate encoder. Reincorporating timing into the processing suggests two distinct signal processing paths: neuromorphic computing and subframe image reconstruction.

Neuromorphic, meaning *brain shaped*, is an elastic term of art. Used here, neuromorphic computing means the use of spikes to communicate between small dynamic processing elements (neurons). A spike represents a short pulse or single bit transmitted from one neuron to typically many other neurons. The relative timing can encode a great deal of information. For example, see [33, 12]. Conventional neural networks have been converted to spiking neural networks with tools like Whetstone [31], though currently neuromorphic process training is immature [2]. Nevertheless, we followed the ideas of VanRullen [33, 34] and implemented an image reconstruction based on a function of the time of the first spike per coefficient to come from readout. Each coefficient is associated with a positive line and a negative line. Let t_{ij} be the time of the first spike on either line for the coefficient at address (i, j), and let s_{ij} be the corresponding sign of the quadtree line (i.e., +1 for the positive quadtree and −1 for the negative quadtree). If coefficient (i, j) does not spike during some specified interval of time, encode its time as $t_{ij} = \infty$. Then, we can reconstruct the coefficients with

$$x_{ij} = \frac{s_{ij}}{t_{ij}}. \tag{6.1}$$

The two dimensional array $\mathbf{x}$ represents the coefficient plane at a snapshot some instance from a reference time. As time passes, $\mathbf{x}$ has fewer and fewer zero values, and consequently better information. Figure 6-1 shows a sequence of images reconstructed at fixed intervals after the focal plane pixel integration capacitors have been reset.

All of this is to show that *completely ignoring the average rate of coefficient spiking* nevertheless produces a usable image output. Furthermore the reconstruction available very shortly after reset is already revealing the actionable information from the focal plane. This suggests:

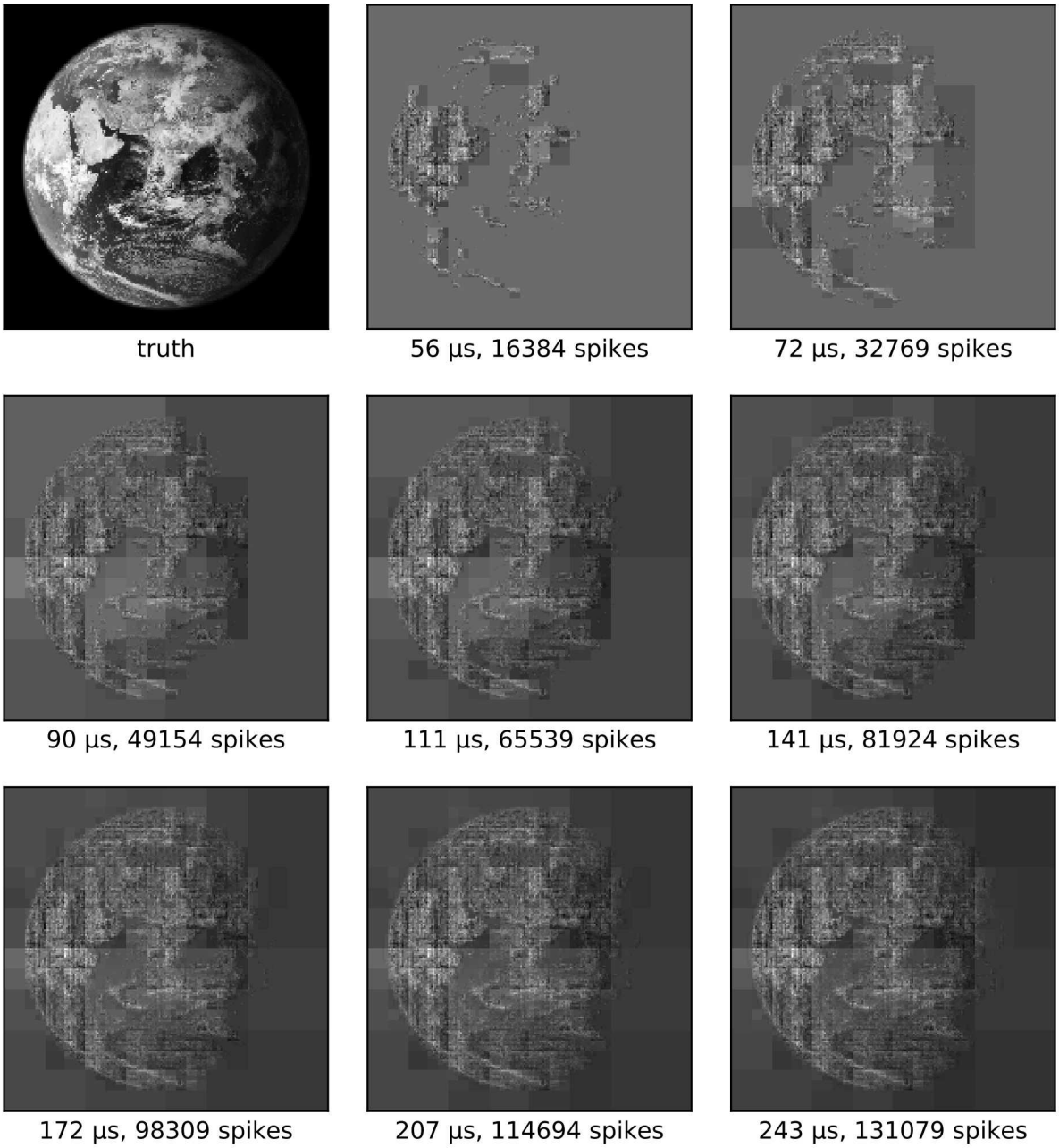

Figure 6-1. Evolution of reconstructed image as time passes and more coefficients acquire nonzero values.

- information from the focal plane is likely to be actionable with very low latency and correspondingly enables wide-bandwidth control loops,
- information encoded in spike times offers significant value and that while we have shown rate encoding is effective for transient detection the combination should be more powerful,
- a system capable of processing both time and number of spikes should be able to perform imaging functions and be reactive, in other words a neuromorphic processing system is a natural partner to SPARR.

SPARR presents a second opportunity to benefit from the continuous stream of data at fine time resolution. Specifically, compensation for motion experienced by the sensor is possible at the spike resolution. The exact time resolution of the sensor depends the brightness of features in the scene and on the size of the integration capacitor. In many scenarios motion compensation may be possible at microsecond resolution. Such fast compensation make it possible to build images without motion blur, even from unstable platforms and even without motion compensating optics.

The research opportunities and needs are summarized:

- Couple SPARR to neuromorphic computing systems to exploit the unique time information the sensor provides.
- Develop per-spike motion compensation, and possibly motion detection, capability to enable high-resolution operation on unstable platforms.
- Develop a rigorous interpretation of the spike timing statistics for multi-electron integration. Small systems of four pixels and processing elements, along with a modest photo-electron integrating capacity of a few hundred electrons have been analyzed as Markov processes and used to validate the `sparrow` simulator. The implemented Markov models have been built with sparse arrays to minimize memory use, but they can grow very little within the memory and processing capacities of all available computers.
- Treat the existing processing as spatial-temporal, rather than strictly spatial. The DAPES have memory, but that memory cannot be translated to a fixed amount of time since it depends on the exact sequence of spikes arriving. Nevertheless, time is integral to the behavior of the system.
- Add overt time processing to the array. Many systems, both constructed and biological, have a background suppression function and can only respond to change. SPARR mostly assumes a static behavior; the only exceptions are the small memory behavior of DAPES and the throttle control in the quad-tree readout.

REFERENCES

[1] M. J. Berry, D. K. Warland, and M. Meister. The structure and precision of retinal spike trains. *Proceedings of the National Academy of Sciences*, 94(10):5411–5416, May 1997. doi: 10.1073/pnas.94.10.5411. URL https://doi.org/10.1073/pnas.94.10.5411.

[2] Zhenshan Bing, Claus Meschede, Florian Röhrbein, Kai Huang, and Alois C. Knoll. A survey of robotics control based on learning-inspired spiking neural networks. *Frontiers in Neurorobotics*, 12:35, 2018. ISSN 1662-5218. doi: 10.3389/fnbot.2018.00035. URL https://www.frontiersin.org/article/10.3389/fnbot.2018.00035.

[3] S. S. Blackman. Multiple hypothesis tracking for multiple target tracking. *IEEE Aerospace and Electronic Systems Magazine*, 19(1):5–18, Jan 2004. ISSN 0885-8985. doi: 10.1109/MAES.2004.1263228.

[4] B. Buchanan, C. Camperi-Ginestet, T. Morris, M. Brooke, S. DeWeerth, N. Jokerst, and M. Allen. High density focal plane signal processing using 3-d vertical interconnects. In *Proceedings of 1994 37th Midwest Symposium on Circuits and Systems*. IEEE, 1994. doi: 10.1109/mwscas.1994.519220. URL https://doi.org/10.1109/mwscas.1994.519220.

[5] Erhan Çinlar. *Introduction to Stochastic Processes*. Dover Publications, Inc., Mineola, New York, 2013.

[6] G. Cohen, S. Afshar, A. van Schaik, A. Wabnitz, T. Bessell, M. Rutten, and B. Morreale. Event-based Sensing for Space Situational Awareness. In *Advanced Maui Optical and Space Surveillance (AMOS) Technologies Conference*, page 25, 2017.

[7] J.G. Daugman. Complete discrete 2-d gabor transforms by neural networks for image analysis and compression. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 36(7):1169–1179, July 1988. doi: 10.1109/29.1644. URL https://doi.org/10.1109/29.1644.

[8] M. Davies, N. Srinivasa, T. H. Lin, G. Chinya, Y. Cao, S. H. Choday, G. Dimou, P. Joshi, N. Imam, S. Jain, Y. Liao, C. K. Lin, A. Lines, R. Liu, D. Mathaikutty, S. McCoy, A. Paul, J. Tse, G. Venkataramanan, Y. H. Weng, A. Wild, Y. Yang, and H. Wang. Loihi: A neuromorphic manycore processor with on-chip learning. *IEEE Micro*, 38(1):82–99, January 2018. ISSN 0272-1732. doi: 10.1109/MM.2018.112130359.

[9] R.H. Dennard, F.H. Gaensslen, V.L. Rideout, E. Bassous, and A.R. LeBlanc. Design of ion-implanted MOSFET's with very small physical dimensions. *IEEE Journal of Solid-State Circuits*, 9(5):256–268, October 1974. doi: 10.1109/jssc.1974.1050511. URL https://doi.org/10.1109/jssc.1974.1050511.

[10] Richard M. Fujimoto. Parallel discrete event simulation. *Commun. ACM*, 33(10):30–53, October 1990. ISSN 0001-0782. doi: 10.1145/84537.84545. URL http://doi.acm.org/10.1145/84537.84545.

[11] Park Hays. Spike arrival time model. Technical report, Sandia National Laboratories, 2018.

[12] Arthur R. Houweling and Michael Brecht. Behavioural report of single neuron stimulation in somatosensory cortex. *Nature*, 451(7174):65–68, December 2007. doi: 10.1038/nature06447. URL https://doi.org/10.1038/nature06447.

[13] R. Rex Kay, David V. Campbell, Subhash L. Shinde, Alan Y. Hsu, Darwin K. Serkland, Jeffrey A. Mercier, James E. Levy, Michael L. Holmes, William S. Cavanaugh, Kenneth E. Wojciechowski, Robert O. Lovejoy, G. Ronald Anderson, Sita S. Mani, Joy M. Barker, Dahwey Chu, Todd M. Bauer, Randy J. Shul, Robert J. Fogler, Kurt A. Sandquist, Gordon A. Keeler, Thomas P. Swiler, and Bernhard Jokiel Jr. Highly pixelated, hyper-temporal sensors for global awareness. Technical report, Sandia National Laboratories, September 2009.

[14] S.M. Kay. *Fundamentals of Statistical Signal Processing: Detection theory*. Prentice Hall Signal Processing Series. Prentice-Hall PTR, 1998. ISBN 9780135041352.

[15] Jeffrey Kern. Parametric, frequency-domain approach for clutter analysis and rejection in remote sensing. Technical report, Sandia National Laboratory.

[16] Kristin Koch, Judith McLean, Ronen Segev, Michael A. Freed, Michael J. Berry, Vijay Balasubramanian, and Peter Sterling. How much the eye tells the brain. *Current Biology*, 16 (14):1428–1434, July 2006. doi: 10.1016/j.cub.2006.05.056. URL https://doi.org/10.1016/j.cub.2006.05.056.

[17] Helga Kolb. Facts and figures concerning the human retina. https://www.ncbi.nlm.nih.gov/books/NBK11556/, July 2007. In: Kolb H, Fernandez E, Nelson R, editors. Webvision: The Organization of the Retina and Visual System [Internet]. Salt Lake City (UT): University of Utah Health Sciences Center; 1995-.

[18] Louis Lapicque. Recherches quantitatives sur l'excitation électrique des nerfs traitée comme une polarisation. *J. Physiol. Pathol. Gen.*, 9:620–635, 1907.

[19] A. A. Lazar and L. T. Toth. Perfect recovery and sensitivity analysis of time encoded bandlimited signals. *IEEE Transactions on Circuits and Systems I: Regular Papers*, 51(10): 2060–2073, Oct 2004. ISSN 1549-8328. doi: 10.1109/TCSI.2004.835026.

[20] A. A. Lazar, E. K. Simonyi, and L. T. Tóth. A real-time algorithm for time decoding machines. In *2006 14th European Signal Processing Conference*, pages 1–5, Sept 2006.

[21] Aurel A. Lazar. Time encoding with an integrate-and-fire neuron with a refractory period. *Neurocomputing*, pages 65–66, 2004.

[22] Aurel A. Lazar, Ern K. Simonyi, and László T. Tóth. Time encoding of bandlimited signals, an overview, 2005.

[23] S. Marĉelja. Mathematical description of the responses of simple cortical cells. *J. Opt. Soc. Am.*, 70(11):1297–1300, Nov 1980. doi: 10.1364/JOSA.70.001297. URL http://www.osapublishing.org/abstract.cfm?URI=josa-70-11-1297.

[24] Carver Mead. *Analog VLSI and Neural Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989. ISBN 0-201-05992-4.

[25] P. A. Merolla, J. V. Arthur, R. Alvarez-Icaza, A. S. Cassidy, J. Sawada, F. Akopyan, B. L. Jackson, N. Imam, C. Guo, Y. Nakamura, B. Brezzo, I. Vo, S. K. Esser, R. Appuswamy, B. Taba, A. Amir, M. D. Flickner, W. P. Risk, R. Manohar, and D. S. Modha. A million spiking-neuron integrated circuit with a scalable communication network and interface. *Science*, 345(6197):668–673, aug 2014. doi: 10.1126/science.1254642. URL https://doi.org/10.1126/science.1254642.

[26] Jim Nutaro. A discrete event system simulator, 2018. URL https://web.ornl.gov/~nutarojj/adevs/.

[27] Raghu Pasupathy. *Generating Nonhomogeneous Poisson Processes*. 01 2011. ISBN 9780470400531. doi: 10.1002/9780470400531.eorms0355.

[28] Vision Research. Phantom v1840/v2640 data sheet. https://www.phantomhighspeed.com/products/cameras/ultrahighspeed/v2640, April 2019.

[29] B. Ristic, B. Vo, B. Vo, and A. Farina. A tutorial on bernoulli filters: Theory, implementation and applications. *IEEE Transactions on Signal Processing*, 61(13): 3406–3430, July 2013. doi: 10.1109/TSP.2013.2257765.

[30] Kurt A. Sandquist, Paul M. Furth, Alex H. Hsia, James E. Levy, and R. Rex Kay. Ultra-high dynamic range focal plane array circuit architecture. Technical Report SAND2009-0534, Sandia National Laboratories, Feb 2009.

[31] William M. Severa, Craig M. Vineyard, Ryan Dellana, and James B. Aimone. Whetstone: An accessible platform-independent method for training spiking deep neural networks for neuromorphic processors. http://prod.sandia.gov/sand_doc/2018/180580a.pdf, January 2018.

[32] Brian Tyrrell, Kirk Anderson, Justin Baker, Robert Berger, Matthew Brown, Curtis Colonero, Joseph Costa, Brian Holford, Michael Kelly, Eric Ringdahl, Kenneth Schultz, and James Wey. Time delay integration and in-pixel spatiotemporal filtering using a nanoscale digital CMOS focal plane readout. *IEEE Transactions on Electron Devices*, 56(11): 2516–2523, November 2009. doi: 10.1109/ted.2009.2030719. URL https://doi.org/10.1109/ted.2009.2030719.

[33] Rufin VanRullen and Simon J Thorpe. Surfing a spike wave down the ventral stream. *Vision Research*, 42(23):2593–2615, October 2002. doi: 10.1016/s0042-6989(02)00298-5. URL https://doi.org/10.1016/s0042-6989(02)00298-5.

[34] Rufin VanRullen, Rudy Guyonneau, and Simon J. Thorpe. Spike times make sense. *Trends in Neurosciences*, 28(1):1 – 4, 2005. ISSN 0166-2236. doi: https://doi.org/10.1016/j.tins.2004.10.010. URL http://www.sciencedirect.com/science/article/pii/S0166223604003546.

[35] WikiChip. Technology node. https://en.wikichip.org/w/index.php?title=technology_node&oldid=92578. URL https://en.wikichip.org/w/index.php?title=technology_node&oldid=92578. Accessed : 2019-08-21.

[36] Mark Woods. Probability of spikes with the Haar wavelet. Technical report, Sandia National Laboratories, 2018.

[37] Mark Woods. Generating random variables from a non-stationary poisson process. Technical report, Sandia National Laboratories, 2018.

[38] Mark Woods. A piecewise linear approximation for the likelihood function. Technical report, Sandia National Laboratories, 2018.

[39] Xilinx Corporation. Aurora 64B/66B v11.2 LogiCORE IP product guide. https://www.xilinx.com/support/documentation/ip_documentation/aurora_64b66b/v11_2/pg074-aurora-64b66b.pdf, April 2018.

APPENDIX A. ACRONYMS

APR automatic place-and-route

ASIC application-specific integrated circuit

ASP asynchronous self-resetting pixel

CDF cumulative density function

CMF cumulative mass function

CMOS complementary metal-oxide-semiconductor

CRLB Cramér-Rao lower bound

CTMC continuous time Markov chain

DAPE digital asynchronous processing element

DC direct current

DWT discrete wavelet transform

FOV field of view

FPA focal plane array

IAF integrate-and-fire

IC integrated circuit

IID independent and identically distributed

LSB least significant bit

MHT multiple hypothesis tracker

MLE maximum likelihood estimator

MSB most significant bit

PDF probability density function

PEX parasitic extraction

PMF probability mass function

PRD percent root-mean-square difference

PSF	point spread function
SNR	signal-to-noise ratio
SOI	silicon-on-insulator
SPAD	single-photon avalanche photodiode
SPARR	Spiking/Processing Array
TDM	time decoding machine
TEM	time encoding machine
TOA	time-of-arrival

APPENDIX B. SPARROW USER'S GUIDE

B.1. EXECUTIVE SUMMARY

The `sparrow` simulator is a discrete event simulator utilizing the ADEVS package [26] from Oak Ridge National Laboratory and is designed to model the signal processing present on the proposed SPARR FPA. It uses a YAML formatted input deck and outputs an HDF5 output file. This document assumes knowledge of the SPARR FPA and the signal processing it performs.

B.2. INPUT FILE

B.2.1. Input YAML File

Sparrow utilizes a YAML formatted input file.

- `output` – Contains keys related to the simulation output file.
  - `filename` – Name of the output file.
  - `interval` – Amount of time in seconds to save accumulated image outputs.
  - `variables` – List of the groups to save to the output file. There are four different kinds of outputs: spike list outputs, framed image outputs, spike count outputs, and throttling outputs. The data structures for all output types are fully described in section B.3.

    The spike list outputs give a list of output spike times in seconds and the corresponding locations, decimations, and signs. The spike list outputs are:

    * `pixelSpikes` – Saved every time each pixel generates a spike.
    * `outputHaar` – Saved every time each Haar element (positive or negative) produces a spike.
    * `output` – Saved every time the apex quadtree element (positive or negative) produces a spike.
    * `outputSpatialSummation` – If spatial summation and differencing is enabled, outputs spikes from the spatial summation layer.

* `outputSpatialContrast` – If spatial summation and differencing is enabled, outputs spikes from the spatial contrast layer.

The framed image outputs give accumulations of spikes into image frames according to the `interval` input. The framed image outputs are:

* `pixelSpikesImage` – Accumulated number of spikes produced by each pixel.
* `outputHaarImageP` – Accumulated number of positive spikes output by each element in the Haar decomposition.
* `outputHaarImageN` – Accumulated number of negative spikes output by each element in the Haar decomposition.
* `outputHaarImage` – Accumulated output by each element in the Haar decomposition. This image subtracts the negative spikes from the positive spikes.
* `outputImageP` – Accumulated number of positive spikes on a pixel basis output from the quadtree readout.
* `outputImageN` – Accumulated number of negative spikes on a pixel basis output from the quadtree readout.
* `outputImage` – Accumulated spikes from the quadtree readout where each spike is weighted according to its quadtree decimation. This image subtracts the negative spikes from the positive spikes.
* `outputSpatialSummationImage` – If spatial summation and differencing is enabled, this outputs an accumulated image output from the summation layer.
* `outputSpatialContrastImage` – If spatial summation and differencing is enabled, this outputs the spatial difference image output from the contrast layer.

The spike count outputs are simply the total number of processed spikes, and are always output. The spike count outputs are:

* `pixelSpikeCounts` – Number pixel spikes processed over the duration of the simulation.
* `haarCAPSpikeCounts` – Number of positive output spikes from the Haar c_A nodes over all levels of the transform.
* `haarCANSpikeCounts` – Number of negative output spikes from the Haar c_A nodes over all levels of the transform.
* `haarCHPSpikeCounts` – Number of positive output spikes from the Haar c_H nodes over all levels of the transform.
* `haarCHNSpikeCounts` – Number of negative output spikes from the Haar c_H nodes over all levels of the transform.

    * `haarCVPSpikeCounts` – Number of positive output spikes from the Haar c_V nodes over all levels of the transform.
    * `haarCVNSpikeCounts` – Number of negative output spikes from the Haar c_V nodes over all levels of the transform.
    * `haarCDPSpikeCounts` – Number of positive output spikes from the Haar c_D nodes over all levels of the transform.
    * `haarCDNSpikeCounts` – Number of negative output spikes from the Haar c_D nodes over all levels of the transform.
    * `quadTreePCounts` – Number of spikes generated over all nodes in the positive quadtree output.
    * `quadTreeNCounts` – Number of spikes generated over all nodes in the negative quadtree output.
    * `quadTreeLevelCounts` – Number of spikes input to each level of the quadtree. There is one output per level, with the first entry being the level and the second entry being the number of spikes that entered that level.
    * `quadTreeCounts` – If spatial summation and differencing is enabled, this outputs the number of spikes generated over all nodes in the quadtree.
    * `decimatorSpikeCounts` – If spatial summation and differencing is enabled, this outputs the number of spikes generated by the decimator levels used for spatial summation.
    * `spatialSummationCounts` – If spatial summation and differencing is enabled, this outputs the number of spikes generated by the spatial summation.
    * `spatialContrastCounts` – If spatial summation and differencing is enabled, this outputs the number of spikes generated by the spatial differencing.

  The throttling outputs give the current and new throttling levels. The throttling outputs are:

    * `throttling` – Output when using spatial sum/differencing with a single quadtree.
    * `throttlingPositive` – Output when using Haar decomposition. Throttling level of positive quadtree.
    * `throttlingNegative` – Output when using Haar decomposition. Throttling level of negative quadtree.

- `terminateTime` – Simulation time in seconds at which to end the simulation. The `startTime` under `array` specifies the starting time in seconds of the simulation.
- `input` – Contains keys related to the simulation input file.

- `filename` – HDF5 filename that contains a per pixel spike list, the background likelihood image, or a series of background likelihood images and times depending on the `arrayType` input.

- `array` – Contains keys related to the array and the event specification.
  - `seed` – Random number seed to use for generating pixel spikes.
  - `startTime` – Starting time for the simulation in seconds.
  - `arrayType` – Use 0 if each pixel is given a predetermined spike list in the input HDF5 file. Use 1 if each pixel determines its spike times based on the background likelihoods in the input HDF5 file and an event specification described later. The thinning algorithm is used to sample the event [27]. Use 2 if you sample the event's background likelihoods in the input HDF5 file and an event specification described later. The algorithm described in [5] is used to sample the event, which can be helpful for speeding up the sampling. Use 3 if the background likelihoods are changing accoding the input HDF5 file.
  - `pixelResetTime` – Time in seconds that it takes for a pixel to reset. This means that if another spike is generated during this reset time, that spike will be ignored and the reset time will be extended to the reset time after the second, dropped spike. The result is that a pixel in saturation will not spike.
  - `startPixelAtZero` – If set to 1, then it will cause all pixel electron wells to be initialized to the reset level, nominally zero. If set to 0 or not present, then pixels will be randomly initialized between the reset level and the full level.
  - `numberElectrons` – Mean number of electrons it takes a pixel to spike. The number of electrons to spike is recomputed for each spike based on a Gaussian distribution.
  - `numberElectronsStdDev` – Standard deviation of the number of electrons it takes a pixel to spike. The number of electrons to spike is recomputed for each spike based on a Gaussian distribution.
  - `resetLevel` – Mean number of electrons for a pixel electron well to reset to after a spike. The reset level is recomputed each spike based on a Gaussian distribution.
  - `resetLevelStdDev` – Standard deviation of the number of electrons for a pixel electron well to reset to after a spike. The reset level is recomputed each spike based on a Gaussian distribution.
  - `readoutMethod` – Use 0 for quadtree readout. Use 1 for array accumulator readout (not typically used).
  - `processingMethod` – Use 0 for spatial sum/difference (not typically used). Use 1 for the Haar transform.

- `spikesToPrecompute` – In order to facilitate generating spikes in parallel, pixels maintain a buffer of this many spikes up to the termination time of the simulation. When any pixel empties this buffer, all pixels are requested to fill their buffers to this level. The larger this value, the greater the amount of parallelism at the expense of up-front time computing the spikes.
- `pixelsPerSecond` – The number of times per second the array accumulator outputs a pixel. For array accumulator readout only.
- `throttleInterval` – How often in seconds the quadtree checks its throttling settings. For quadtree readout only.
- `increateRateThreshold` – If the quadtree receives more spikes per `throttleInterval` than this value, it increases the amount of throttling performed. For quadtree readout only.
- `decreateRateThreshold` – If the quadtree receives less spikes per `throttleInterval` than this value, it decreases the amount of throttling performed. For quadtree readout only.
- `initialThrottleLevel` – Initial throttling level to use. The quadtree decimation is 2 to this input's power. For quadtree readout only.
- `baseRate` – This is the base electron rate to use for generating pixel spikes. A likelihood of 1 means a pixel will accumulate electrons on average at this rate.
- `psfSigma` – Specifies the PSF of the optical system using a simple Gaussian model. Only used for events specified below.
- `sampling` – Contains keys related to sampling the likelihood function of each pixel. This sampling is necessary if using `arrayType` 2 or 3.
  - `type` – Sampling type can be either `linear` or `adaptive`.
  - `interval` – Time interval in seconds between samples. For linear sampling only.
  - `minInterval` – Minimum time interval between samples. For adaptive sampling only.
  - `maxInterval` – Maximum time interval between samples. For adaptive sampling only.
  - `tolerance` – Relative error tolerance for testing sample points. For adaptive sampling only.
  - `numTests` – Number of testing points per interval. For adaptive sampling only.
- `events` – List of events to use for the simulation. Each event must be modeled as a moving cosine or using polynomials to describe the motion and intensity. In both cases, the following inputs are used:
  - `startTime` – Starting time in seconds of the event. The event will be 0 intensity before this time.

* `stopTime` – Stopping time in seconds of the event. The event will be 0 intensity after this time.
* `psfType` – Either `gaussian` or `lorentzian`. If not specified, `gaussian` is used.
* `psfSigma` – For `gaussian psfType`, this input is the variance of the Gaussian. For `lorentzian psfType`, this input is the gamma parameter of the Lorentzian. In either case, this input is the scale parameter of the distribution.

For a cosine signal, the event likelihood is given by

$$L(t) = L_0 + A\cos(\omega t + \phi),$$

and the following inputs must be used:

* `xLoc` – Column location of the event in pixels at $t = 0$.
* `yLoc` – Row location of the event in pixels at $t = 0$.
* `deltaX` – Linear column velocity of the event in pixels per second.
* `deltaY` – Linear row velocity of the event in pixels per second.
* `amplitude` – Amplitude of the cosine curve. Given by A in the above equation.
* `offset` – Constant offset of the event. Given by L_0 in the above equation.
* `angleRate` – Angular frequency of the cosine curve. Given by ω in the above equation.
* `phase` – Phase of the cosine curve. Given by ϕ in the above equation.
* `constantIntensity` – Boolean stating if the event changes intensity or not. In this case, the event's likelihood is $L(t) = L_0 + A$. Although the same could be acheived using $\omega = 0$ and $\phi = 0$, this option allows for code optimizations.

For the polynomial type event, arbitrary order polynomials are used to describe the column motion, the row motion, and the event likelihood. In this case, the following inputs are used:

* `xPolynomial` – List of coefficients of the polynomial $x(t)$ describing the column motion where the lowest order coefficient is first. Time t is relative to the start of the event, not the simulation.
* `yPolynomial` – List of coefficients of the polynomial $y(t)$ describing the row motion where the lowest order coefficient is first. Time t is relative to the start of the event, not the simulation.
* `intensityPolynomial` – List of coefficients of the polynomial $L(t)$ describing the event likelihood where the lowest order coefficient is first. Time t is relative to the start of the event, not the simulation.

- `resetTimes` – Contains keys that specify the amount of time in seconds it takes for the given item to reset. If the item receives a spike during the reset time, the spike is ignored and the reset time is recomputed to the time of last input spike plus the reset time. This behavior means that if the input is saturated all input spikes are ignored until the element goes the full reset time without an input spike.
  - `decimator` – Decimator DAPE.
  - `aggregator` – Aggregator DAPE.
  - `differentiator` – Differentiator DAPE.
  - `quadTree` – Quadtree DAPE
  - `haarAA` – c_A Haar element DAPE.
  - `haarDA` – c_H Haar element DAPE.
  - `haarAD` – c_V Haar element DAPE.
  - `haarDD` – c_D Haar element DAPE.
- `outputDelays` – Contains keys that specify the time in seconds it takes from when an input spike arrives that triggers an output until the output spike is generated.
  - `decimator` – Decimator DAPE.
  - `aggregator` – Aggregator DAPE.
  - `differentiator` – Differentiator DAPE.
  - `quadTree` – Quadtree DAPE
  - `haarAA` – c_A Haar element DAPE.
  - `haarDA` – c_H Haar element DAPE.
  - `haarAD` – c_V Haar element DAPE.
  - `haarDD` – c_D Haar element DAPE.
- `haarNumStates` – Contains keys that control the number of states used in the Haar transform state machines.
  - `cATop` – Apex c_A node.
  - `cAOther` – All other c_A nodes.
  - `cH` – All c_H nodes.
  - `cV` – All c_V nodes.
  - `cD` – All c_D nodes.
- `haarResetDifferential` – Contains keys that control the decimation of the Haar transform state machines. For example, a value of 2 gives a 2:1 decimation level.

* `cATop` – Apex c_A node.
* `cAOther` – All other c_A nodes.
* `cH` – All c_H nodes.
* `cV` – All c_V nodes.
* `cD` – All c_D nodes.

B.2.2. Input HDF5 File

If the array is in the precomputed spikes input mode (`arrayType` is 0), the input HDF5 file must contain a `/data` group that contains a `[height, width, numSpikes]` array where each pixel has a list of spikes. If the number of spikes is different for each pixel, then `NaN` can be used to pad the array to a rectangular size.

For simulations where the spike times are computed by the simulation (`arrayType` is 1 or 2), the input file must contain a `/background_likelihood` group which has a `[height, width]` array that holds the likelihood that each pixel will spike between 0 and 1. A pixel with a likelihood of 1 will spike on average at the `baseRate` specified in the input YAML file.

For simulations where a series of background likelihoods are specified (`arrayType` is 3), the input file must contain a `/times` group which has a list of times for the background likelihoods and a `/data` group that contains a `[height, width, numTimes]` array that holds the background likelihoods for each pixel at all simulation times.

B.3. OUTPUT FILE

The output HDF5 file contains 3 main types of outputs, individual spike outputs, image outputs, and spike counts.

Spike outputs save 6 items for each spike: time, x location, y location, spike weight through the Haar transform, spike weight through quadtree readout, and spike sign. The spike weights are equivalent to the amount of decimation performed on each spike.

Image outputs save `[frame, height, width]` as a 3D dataset. The number of images saved is

$$\left\lceil \frac{\text{total simulation time}}{\text{output interval}} \right\rceil$$

where $\lceil \cdot \rceil$ denotes the ceiling function.

Spike counts are saved in an array of 6 elements where the first entry contains the spike count, and all other elements are 0. This may seem strange, but it allows us to reuse much of the HDF5 output code with only a slight increase in memory.

B.4. IMPLEMENTATION DETAILS

B.4.1. Performance Improvements to Likelihood Computation

A significant portion of the `sparrow` simulation time is spent computing when a pixel will spike. Due to the nature of photon arrival times, it was initially necessary to simulate the arrival of every photon in each pixel using the thinning algorithm [27].

The first performance improvement came when the photon arrival likelihood was normalized by the peak likelihood that a pixel could see over the duration of the simulation. See section B.4.2 for more details.

The second performance improvement only applied to pixels with a constant likelihood. This allowed replacement of the per-photon simulation with a single per-spike computation [11].

The third spike determination performance improvement performed a piecewise linear fit to the time-varying likelihood function which allowed a single function evaluation to determine the time of the next spike [37], [38].

B.4.2. Likelihood Normalization

In order to reduce computation time, the likelihood used on a per-pixel basis is normalized by the peak likelihood of each pixel over the duration of the simulation. This requires being able to compute the minimum distance between the pixel and the path of the event, and the maximum intensity of an event over the duration. If we let

$$u \sim \text{Uniform}(0,1),$$

then we can compute the next time with

$$t = \frac{t_{\text{last}} - \log(u)}{(\text{base rate})(\text{per pixel gain})L_{\text{peak}}}.$$

We then use the likelihood function to evaluate $L_{\text{next}}(t) = L(t_0 + t, x, y, L_{\text{bg}})$ where L_{bg} is the background likelihood and compute

$$L(t) = \frac{L_{\text{next}}(t)}{L_{\text{peak}}}.$$

If a uniformly distributed random number $v < L(t)$, then the value of u is accepted as a contributing electron. The per-pixel offset is added to L_{bg} prior to the computation above.

B.4.3. Parallelizing Sparrow

Due to the asynchronous nature of photon arrival times, a discrete event simulation framework is required. The basic nature of a discrete event simulation is that a given processing node can either generate discrete events, or spikes, based on an internal process or based on receiving an external spike. In cases where spikes are generated completely internally, the computations can happen in parallel; one output spike is not dependent on another output spike.

Once a node is introduced that accepts external spikes as input and produces output spikes based on that external stimulus, the resulting data dependency requires that the spikes be processed sequentially. Spikes processed out of order result in future events influencing the past, or acausal behavior [10].

Within the `sparrow` simulation, computation of pixel spike times is a wholly internal spiking process. As such, the spike times can be computed in parallel. `sparrow` makes use of this by maintaining a buffer of the next N spikes for each pixel, and when any single pixel exhausts its list of precomputed spikes, all pixels are refilled in parallel to N spikes. Subsequent to the pixel spikes, the SPARR architecture is tightly connected via the Haar wavelet transform and the quadtree readout, which makes parallelization of these layers very difficult. If a multi-die approach to SPARR were pursued, then simulation could be performed on each die independently.

The simulation framework currently being used does support multithreading; however, when that option was enabled for SPARR simulations, the performance decreased significantly over the single-threaded version. There are other performance improvements that could be investigated, but those would likely require understanding and rewriting significant portions of the discrete event simulation framework currently being used, which was outside the scope of this effort.

B.5. CODE STRUCTURE

B.5.1. ADEVS

The ADEVS discrete event simulation framework is based on the idea of an atomic model. This model can either be triggered internally (`delta_int()`), in which case, it must provide a `ta()` function that specifies the time until next trigger, or it can be triggered in response to an external stimulus (`delta_ext()`). In the latter case, the `ta()` function should return the maximum possible value until the model receives that external stimulus at which point it should return the time to next trigger.

B.5.2. Pixel Types

`Sparrow` supports three different types of pixels. The first pixel type has a precomputed list of spikes that is read from an input HDF5 file. This is contained in the `PixelAtTimes` class. Given

the simulations performed, these sequences of spikes quickly grow large enough that it's faster to generate them in memory than it is to read them off disk.

The second pixel type, `PixelPoisson`, computes spikes internally based on a likelihood function. The likelihood function itself is based on the PSF of the optics, and the change in position and intensity of the event over time. This class simulates each photon to compute when a pixel will spike. Pixels further than six times the PSF spread value will not include contributions from an event.

`PixelPoisson` maintains a list of the next n spikes, and when any pixel runs out of spikes, all pixels replenish their spike list to n spikes, or to the spike after the simulation termination time. The likelihood calculation used by `PixelPoisson` includes the option to add per-pixel bias (added to the background likelihood) and gain corruption (multiplicative with the base rate), as well as pixels not resetting to 0 electrons, and not triggering exactly at the requested trigger level.

The third pixel type, `PixelPoissonSampled`, is a subclass of `PixelPoisson` and generates a piecewise-linear representation of the likelihood function from `PixelPoisson` to improve performance.

The latter two pixel types are included in the `PixelPoissonArray` class, which handles notifying all the pixels to generate spikes, when one pixel runs out of spikes. This enables the parallel speedup discussed in section B.4.3.

In support of events changing intensity and location over time, there is an `EventStructure` class that stores the data and provides the methods necessary to model the intensity or location function and use it to compute the pixel trigger times. The standard `EventStructure` has a cosine curve for the intensity and a linear velocity vector. `EventStructurePolynomial` replaces this with an nth order polynomial fit for both the intensity and the position as a function of time.

B.5.3. Processing Layers

`Sparrow` contains two processing layers. The first is a spatial summation filter followed by a spatial difference filter. Notionally this sums the energy in the PSF, then it applies an edge detection filter to find point-like features. Since the filtering is being performed in the spike domain, there's no concept of a fractional spike. Instead, the spike rate is used to encode intensity, so spike decimation is used to multiply the rate by a number between zero and one.

`Sparrow` contains a `Decimator` class that is capable of performing this decimation. For ease of circuit implementation, the decimator always represents a factor of $1/n$ since this just means pass every n^{th} spike.

The spatial filtering is handled by the `WeightedCombiner` class. This class takes a 2D array of integer filter weights in a region around the target pixel. When a spike arrives from a pixel, the weight at that location is added to the current total weight, and when that reaches a threshold, a spike is output. Since the weights can be both positive and negative, this allows creation of both the spatial summation, and spatial difference filters. There is a simpler version of

`WeightedCombiner` called `Aggregator`, which simply outputs all the input spikes it receives. It's akin to an OR gate in the actual hardware.

The Haar wavelet decomposition is handled by the `DifferentialNode` class [36]. It uses differential signaling through a positive and negative spike line to represent the sign of the data. The state machine has a user-specifiable number of states and the decimation is controlled by the state the machine transitions upon output.

All of the classes above handle reset times and output delays. Reset times force the element to ignore spikes until the reset time has elapsed, with a spike arriving during the reset time causing the reset time to extend. Output delays cause appropriate latencies in spikes between when they should be output and when they actually are output. Both of these behaviors were intended to mimic the performance of actual hardware.

B.5.4. Readout Methods

`Sparrow` contains two readout methods. The first readout method, contained in `ArrayAccumulator`, attempts to readout the pixel with the most spikes since the last readout. It accomplishes this by keeping track of the number of times any pixel in each row spiked. Then during readout, it selects the row with the most spikes, finds the pixel in that row with the most spikes, and reads it off. Finally, it subtracts the number of spikes read out from the row sum. This method had issue with spike collisions since any spike in a pixel within the row tries to increment the same counter.

The second readout method, contained in `QuadTreeNode`, connects all the outputs in a 2D quadtree, with each node accepting inputs from a 2×2 grid of nodes below it. When a node receives a spike, that node passes along the spike as well as 2 address bits that specify which input caused the spike. The address bits are propagated up the tree, growing by 2 bits at every level, until the final node is reached.

This method is capable of outputting every spike along with its location as long as the spike rate is low. When the spike rate increases, it can quickly become saturated. Each layer is capable of independently providing a 2:1 or 4:1 decimation if commanded to do so by the `QuadTreeTrafficControl`. Under decimation, only every other or every fourth spike is passed at each level. This results in the same statistical distribution of spikes at the output as at the input, but with some quantization loss.

The `QuadTreeTrafficControl` monitors the number of output spikes and increases the throttling rate if it receives more than a set number of spikes in a given time interval, and decreases the throttling rate if it receives less than a set number of spikes. All of these parameters are configurable in the input file. When the throttling rate is changed, it is set for all nodes at a given level. A single level will never have a mix of throtting across its nodes.

B.5.5. Data Output

Data is output in HDF5 format files by the `HDF5Writer` class. There is a helper class, `ObserverHDF5` that can be connected to other ADEVS primitives and logs any spikes received to a specified HDF5 output variable. These observers can log both the individual spikes, or images containing the number of times a spike was received for each pixel. In addition, positive and negative spikes are logged individually as well as in summation.

APPENDIX C. ALTERNATIVE SIGNAL PROCESSING

C.1. ALTERNATIVE SIGNAL PROCESSING

The goal of signal processing within the SPARR FPA is to remove most of the unneeded information, enabling achievable data read-out rates. The throttled Haar transform as described in section 2.3 represents only one possibility for accomplishing this goal. Another idea is to try to maximize the SNR and subtract the background, which is the topic here. We perform both operations via filters; a matched filter maximizes signal, and a spatial differentiator subtracts background . Combined, these filters both maximize SNR and reduce the total data needed.

C.1.1. Matched Filter Spikes

Many designs are possible for a matched filter. For SPARR, we approximated an optical point spread function PSF that overfills a single pixel with a 3×3 pixel filter that decimates each pixel's spike train according to figure C-1. That is, for each pixel, the eight surrounding pixel spike outputs are decimated according to figure C-1 and then those spike trains are added together. Pixels on the edge of the FPA have fewer than eight neighboring pixels (either three or five), but we simply treat the nonexisting pixels as zeros. If we index the 3×3 array of pixels in row-major order (i.e., according to figure C-2), then we can let $g^{(m)}(K)$ for $m = 0,1,\ldots,8$ be the PMF for the number of pixel spikes for the pixel at index m, which comes directly from equation (2.8), and then apply the corresponding decimation $D_{\mathrm{mf}}^{(m)}$ to get

$$\Pr(K \text{ spikes after decimation}) = \frac{1}{D_{\mathrm{mf}}^{(m)}} \sum_{j=0}^{D_{\mathrm{mf}}^{(m)}-1} \sum_{i=KD_{\mathrm{mf}}^{(m)}-j}^{(K+1)D_{\mathrm{mf}}^{(m)}-j-1} g^{(m)}(i) \tag{C.1}$$

$$= \frac{1}{D_{\mathrm{mf}}^{(m)}} \sum_{j=0}^{D_{\mathrm{mf}}^{(m)}-1} \left[G^{(m)}\big((K+1)D_{\mathrm{mf}}^{(m)} - j - 1\big) - G^{(m)}\big(KD_{\mathrm{mf}}^{(m)} - j - 1\big) \right] \tag{C.2}$$

$$\equiv g_D^{(m)}(K). \tag{C.3}$$

Finally, the PMF for the number of matched filter spikes comes directly from equation (2.35). We find

$$\Pr(K \text{ matched filter spikes}) = \left(g_D^{(0)} * g_D^{(1)} * \cdots * g_D^{(8)}\right)(K) \tag{C.4}$$

$$\equiv h(K). \tag{C.5}$$

4:1	2:1	4:1
2:1	1:1	2:1
4:1	2:1	4:1

Figure C-1. A 3 × 3 matched filter decimation scheme.

0	1	2
3	4	5
6	7	8

Figure C-2. Row-major indexing scheme for the 3 × 3 matched filter.

C.1.2. Differentiator Spikes

The differentiating filter is only slightly more complicated. We again use a 3×3 filter and decimate according to figure C-3. We also use figure C-2 for the indexing so that the matched filter PMFs are $h^{(m)}(K)$ for $m = 0,1,\ldots,8$. Although most of the decimations are now negative, we still apply the decimation as if they are positive. That is we decimate according to

$$\Pr(K \text{ spikes after decimation}) = \frac{1}{|D_{\text{diff}}^{(m)}|} \sum_{j=0}^{|D_{\text{diff}}^{(m)}|-1} \sum_{i=K|D_{\text{diff}}^{(m)}|-j}^{(K+1)|D_{\text{diff}}^{(m)}|-j-1} h^{(m)}(i) \tag{C.6}$$

$$= \frac{1}{|D_{\text{diff}}^{(m)}|} \sum_{j=0}^{|D_{\text{diff}}^{(m)}|-1} \left[H^{(m)}\left((K+1)|D_{\text{diff}}^{(m)}|-j-1\right) - H^{(m)}\left(K|D_{\text{diff}}^{(m)}|-j-1\right)\right] \tag{C.7}$$

$$\equiv h_D^{(m)}(K). \tag{C.8}$$

−12:1	−6:1	−12:1
−6:1	1:1	−6:1
−12:1	−6:1	−12:1

Figure C-3. A 3×3 differentiator decimation scheme.

Clearly, there will be a positive and negative spike train for each pixel. Unlike with the Haar transform, it is fairly simple to obtain the PMFs for the number of positive and negative spikes from the differentiator. The positive output is simply

$$\Pr(K \text{ pos. differentiator spikes}) = h_D^{(4)}(K) \tag{C.9}$$
$$= h^{(4)}(K) \tag{C.10}$$
$$\equiv u^+(K). \tag{C.11}$$

The negative output uses equation (2.35) to obtain

$$\Pr(K \text{ neg. differentiator spikes}) = \left(h_D^{(0)} * \cdots * h_D^{(3)} * h_D^{(5)} * \cdots * h_D^{(8)}\right)(K) \tag{C.12}$$
$$\equiv u^-(K). \tag{C.13}$$

Equations (C.11) and (C.13) can be used directly for the quadtree PMF outputs given in equation (2.75). Finally, if the PMF for the number of net differentiator spikes is desired, we can use equation (2.37) to get

$$\Pr(K \text{ net differentiator spikes}) = \left(u^- \star u^+\right)(K) \tag{C.14}$$
$$\equiv u(K). \tag{C.15}$$

The spatial matched-filter then differentiator concept was evaluated with the `sparrow` simulator and through Cadence simulation. However, a weakness of the matched filter is that it increases the spike rate local to the pixel by the magnitude sum of the coefficients. For this analysis, the sum of matched filter coefficients is four, so the spike rate *out of the matched filter* would be four times the spike rate from a single pixel, with unfortunate effects on dynamic range. However, for some applications similar functional realizations could be realized without the spike rate amplification effect, e.g., placing the differentiator first and following it with a signed matched filter. Our main reason for pursuing the Haar transform approach is its applications to imaging and multiscale spatial processing–features unavailable in the local differentiator design.

APPENDIX D. SIGNAL-TO-NOISE RATIO

We model the arrival of photons on the sensor as realizations of a nonhomogeneous Poisson point process. $X(t)$ is the random signal signifying the number of photons the sensor receives at a given time. That is

$$X(t) \sim \text{Poiss}(\lambda(t)) \tag{D.1}$$

where $\lambda(t)$ is the time-dependent rate function, representing the average instantaneous number of photon arrivals per unit time.

We model $X(t)$ as a summation of two inherent signals, $S(t)$, representing the important signal component of the received data, and $N(t)$, representing noise and background. Both $S(t)$ and $N(t)$ are nonhomogeneous Poisson point processes as well, since the sum of two Poisson random variables results in a Poisson random variable. Hence,

$$X(t) = S(t) + N(t) \tag{D.2}$$

where

$$S(t) \sim \text{Poiss}(\lambda_S(t)), \tag{D.3a}$$

$$N(t) \sim \text{Poiss}(\lambda_N(t)), \tag{D.3b}$$

and

$$\lambda(t) = \lambda_S(t) + \lambda_N(t). \tag{D.4}$$

The instantaneous SNR is the ratio of the instantaneous power of the signal component compared with that of the noise component:

$$\text{SNR} = \frac{\text{instantaneous power of } S(t)}{\text{instantaneous power of } N(t)}. \tag{D.5}$$

The instantaneous energy of an optical signal such as $X(t)$ is $hf\Lambda(t_0,t)$ where h is the Planck constant, f is the photon frequency, and

$$\Lambda(t_0,t) = \int_{t_0}^{t} \lambda(\tau)\, d\tau \tag{D.6}$$

is the number of photons incident on the sensor between times t_0 and t. Since power is energy per time,

$$P(t) = \lim_{\Delta t \to 0} \frac{hf\Lambda(t_0,t)}{\Delta t} \tag{D.7}$$

where $P(t)$ is the instantaneous power of the signal and $\Delta t = t - t_0$.

With this, the instantaneous signal-to-noise ratio is

$$\text{SNR}(t) = \lim_{\Delta t \to 0} \left(\frac{hf\Lambda_S(t_0,t)}{\Delta t} \right) \Bigg/ \left(\frac{hf\Lambda_N(t_0,t)}{\Delta t} \right) \tag{D.8}$$

$$= \lim_{\Delta t \to 0} \frac{\Lambda_S(t_0,t)}{\Lambda_N(t_0,t)}, \tag{D.9}$$

where $\Lambda_S(t_0,t)$ is the number of photons incident on the sensor due to the signal component during the interval Δt, and $\Lambda_N(t_0,t)$ is the number of photons incident on the sensor due to the noise component during the interval Δt. Therefore,

$$\text{SNR}(t) = \lim_{t \to t_0} \frac{\int_{t_0}^{t} \lambda_S(\tau)\, d\tau}{\int_{t_0}^{t} \lambda_N(\tau)\, d\tau} \tag{D.10}$$

$$= \lim_{t \to t_0} \frac{\frac{d}{dt} \int_{t_0}^{t} \lambda_S(\tau)\, d\tau}{\frac{d}{dt} \int_{t_0}^{t} \lambda_N(\tau)\, d\tau} \tag{D.11}$$

$$= \frac{\lambda_S(t)}{\lambda_N(t)} \tag{D.12}$$

where we used L'Hospital's rule to evaluate the limit. We now have that the signal-to-noise ratio of the signal incident on the sensor is the ratio of the rate function of the signal component to that of the noise component.

APPENDIX E. SPIKE INTENSITY RECONSTRUCTION VIA THE GAMMA DISTRIBUTION

E.1. SIGNAL MODEL

The SPARR sensor has been designed so that the background appears spatially white for a nominal earth scene. When some event of interest occurs, the background will no longer appear to be spatially white in the neighborhood of the event. In this case, we would like to be able to take the output spikes from the positive and negative quadtrees and reconstruct the signal time series for identification purposes. All signals are transient by definition, so the arrival of photons from the event could be accurately modeled as a nonstationary Poisson process. This is simply a Poisson process whose rate is a function of time ($\lambda = \lambda(t)$). However, the analysis is greatly simplified by using a stationary Poisson process. Therefore, we will assume that given a sufficiently small time window, we can model the arrival of photons from the event of interest as a stationary Poisson process. This means that if a pixel spikes after receiving N photons, then the arrival times of pixel spikes follows a gamma distribution with shape parameter N and rate λ.

However, the pixel spikes are propagated through a series of state machines that approximate a Haar discrete wavelet transform and then through two quadtrees for a double spike-train readout. Two quadtrees are needed because the Haar transform produces positive and negative output spikes, and we keep each line separate (i.e., there is a quadtree for the positive spikes and one for the negative spikes). The presense of the Haar transform significantly complicates the statistics, so as a first try, we will assume it is not present. This means that the pixel spikes go immediately through a single quadtree for the readout.

The quadtree only slightly complicates the statistics. It performs decimation so that spike collisions are infrequent. The amount of decimation is known, so a quadtree spike has a factor D associated with it. Hence, for a given pixel address, each quadtree spike represents D pixel spikes, which in turn each represent N photon arrivals. This means each quadtree spike represents $M = DN$ photon arrivals. We should note that D is known for each spike, but is not constant. SPARR applies more decimation in the quadtree as the rate of input spikes increases, but reduces the decimation when the input spike rate decreases. The decimation can be viewed as a source of quantization error, so less decimation means more information is being preserved.

If we let Δt be the interval of time between two quadtree spikes, then this waiting time follows a gamma distribution with shape parameter M and rate λ. That is, $\Delta t \sim \Gamma(M,\lambda)$, which has a PDF

$$f(\Delta t; M, \lambda) = \frac{\lambda^M \Delta t^{M-1} e^{-\lambda \Delta t}}{\Gamma(M)} \tag{E.1}$$

where $\Gamma(\cdot)$ is the gamma function defined by

$$\Gamma(s) = \int_0^\infty t^{s-1} e^{-t}\, dt. \tag{E.2}$$

Let us assume in some interval of time we receive $K+1$ quadtree spikes for some pixel, and the spike times are given by t_i for $i = 0,1,\ldots,K$. Then, each $\Delta t_i = t_i - t_{i-1}$ is an independent and identically distributed (IID) sample from a gamma distribution with shape parameter M_i and rate λ for $i = 1,2,\ldots,K$. If we define $\mathbf{T} \equiv (\Delta t_1, \Delta t_2, \ldots, \Delta t_K)$ and $\mathbf{M} \equiv (M_1, M_2, \ldots, M_K)$, then we can define the likelihood of a rate λ given $\mathbf{T}$ and $\mathbf{M}$ to be

$$\mathcal{L}\big(\lambda | \mathbf{T}, \mathbf{M}\big) = \prod_{i=1}^{K-1} \frac{\lambda^{M_i} \Delta t_i^{M_i - 1} e^{-\lambda \Delta t_i}}{\Gamma(M_i)}. \tag{E.3}$$

E.2. MAXIMUM LIKELIHOOD ESTIMATOR

We would like to estimate the rate $\hat{\lambda}$ from an observed train of quadtree spikes. One way to do this is to maximize the likelihood function (E.3) with respect to λ. Due to the monotinicity of the logarithm function, we will actually maximize the log-likelihood, which is given by

$$\ell\big(\lambda | \mathbf{T}, \mathbf{M}\big) = \sum_{i=1}^{K} \Big[M_i \log(\lambda) + (M_i - 1)\log(\Delta t_i) - \lambda \Delta t_i - \log\big(\Gamma(M_i)\big) \Big]. \tag{E.4}$$

Taking the partial derivative of equation (E.4) with respect to λ, we find

$$\frac{\partial \ell\big(\lambda | \mathbf{T}, \mathbf{M}\big)}{\partial \lambda} = \sum_{i=1}^{K} \left(\frac{M_i}{\lambda} - \Delta t_i \right). \tag{E.5}$$

The MLE for the rate of photon arrivals is found by setting this expression equal to zero. We get

$$\hat{\lambda} = \frac{\sum_{i=1}^{K} M_i}{\sum_{i=1}^{K} \Delta t_i} \tag{E.6}$$

$$= \frac{N \sum_{i=1}^{K} D_i}{t_K - t_0}, \tag{E.7}$$

which means that the MLE for the rate of photon arrivals is the number of photon arrivals divided by the time interval.

Although the MLE is generally not unbiased or efficient (meaning that the variance is equal to the Cramér-Rao lower bound (CRLB)), it becomes unbiased and efficient as $K \to \infty$. This means

$$\hat{\lambda} \sim \mathcal{N}\left(\lambda, I^{-1}(\lambda)\right) \tag{E.8}$$

as $K \to \infty$ where $I(\lambda) = -\mathrm{E}(\partial^2 \ell / \partial \lambda^2)$ is the Fisher information and $\mathrm{E}(\cdot)$ denotes the expected value. To show this, let us first compute the expected value of $\hat{\lambda}$. This is given by

$$\mathrm{E}\left(\hat{\lambda}\right) = \mathrm{E}\left(\frac{\sum_{i=1}^{K} M_i}{\sum_{i=1}^{K} \Delta t_i}\right) \tag{E.9}$$

$$= \left(\sum_{i=1}^{K} M_i\right) \mathrm{E}\left(\frac{1}{\sum_{i=1}^{K} \Delta t_i}\right) \tag{E.10}$$

$$= \left(\sum_{i=1}^{K} M_i\right) \mathrm{E}\left(\frac{1}{Y}\right) \tag{E.11}$$

where $Y = \sum_{i=1}^{K} \Delta t_i \sim \Gamma(\sum_{i=1}^{K} M_i, \lambda)$, which follows from the fact that each $\Delta t_i \sim \Gamma(M_i, \lambda)$ and the sum of gamma distributed variables is also gamma distributed.

If we now let $Z = 1/Y \sim \mathrm{IG}(\sum_{i=1}^{K} M_i, \lambda)$ where $\mathrm{IG}(\cdot)$ is the inverse gamma distribution, then we have

$$\mathrm{E}\left(\hat{\lambda}\right) = \left(\sum_{i=1}^{K} M_i\right) \mathrm{E}(Z) \tag{E.12}$$

$$= \left(\sum_{i=1}^{K} M_i\right) \left(\frac{\lambda}{\sum_{i=1}^{K} M_i - 1}\right) \tag{E.13}$$

$$= \frac{\sum_{i=1}^{K} M_i}{\sum_{i=1}^{K} M_i - 1} \lambda \tag{E.14}$$

where we have used the fact that if $X \sim \mathrm{IG}(a, b)$, then $\mathrm{E}(X) = b/(a-1)$ so long as $a > 1$. For SPARR, this means $\sum_{i=1}^{K} M_i > 1$, which is almost always true. The only case where it is not true is if the capacitor well size was $N = 1$, there is no decimation in the quadtree (i.e., $D_i = 1$), and we are only considering one time interval (i.e., $K = 1$), which are not restrictive assumptions. Equation (E.14) shows that the MLE given by equation (E.7) is indeed biased. However, our claim that it becomes unbiased as $K \to \infty$ is also clearly seen to be true.

To test whether the MLE is efficient, we start by computing the variance of $\hat{\lambda}$. This is given by

$$\mathrm{Var}\left(\hat{\lambda}\right) = \mathrm{Var}\left(\frac{\sum_{i=1}^{K} M_i}{\sum_{i=1}^{K} \Delta t_i}\right) \tag{E.15}$$

$$= \left(\sum_{i=1}^{K} M_i\right)^2 \mathrm{Var}\left(\frac{1}{\sum_{i=1}^{K} \Delta t_i}\right) \tag{E.16}$$

$$= \left(\sum_{i=1}^{K} M_i\right)^2 \mathrm{Var}(Z) \tag{E.17}$$

$$= \left(\sum_{i=1}^{K} M_i\right)^2 \frac{\lambda^2}{\left(\sum_{i=1}^{K} M_i - 1\right)^2 \left(\sum_{i=1}^{K} M_i - 2\right)} \tag{E.18}$$

$$= \frac{\left(\sum_{i=1}^{K} M_i\right)^2}{\left(\sum_{i=1}^{K} M_i - 1\right)^2 \left(\sum_{i=1}^{K} M_i - 2\right)} \lambda^2 \tag{E.19}$$

where we have used the fact that if $X \sim \mathrm{IG}(a,b)$, then $\mathrm{Var}(X) = b^2/((a-1)^2(a-2))$ so long as $a > 2$. For SPARR, this is slightly more restrictive, but not at all prohibitive. Now, the CRLB is given by

$$\text{CRLB} = I^{-1}(\lambda) \tag{E.20}$$

$$= -\frac{1}{\mathrm{E}\left(\frac{\partial^2 \ell(\lambda|\mathbf{T},\mathbf{M})}{\partial \lambda^2}\right)} \tag{E.21}$$

$$= \frac{1}{\mathrm{E}\left(\sum_{i=1}^{K} \frac{M_i}{\lambda^2}\right)} \tag{E.22}$$

$$= \frac{1}{\sum_{i=1}^{K} M_i} \lambda^2. \tag{E.23}$$

The CRLB is the number such that CRLB $\leq \mathrm{Var}(\hat{\lambda})$. We now see that the MLE given by equation (E.7) is not efficient, but our earlier claim that it becomes efficient as $K \to \infty$ can now be seen to be true.

E.3. MODIFIED ESTIMATOR

We propose to modify the MLE to remove the bias shown in equation (E.7). We propose

$$\hat{\lambda}' = \frac{\sum_{i=1}^{K} M_i - 1}{\sum_{i=1}^{K} \Delta t_i} \tag{E.24}$$

$$= \frac{N \sum_{i=1}^{K} D_i - 1}{t_K - t_0}. \tag{E.25}$$

Now if we compute the expected value of $\hat{\lambda}'$, we find

$$\mathrm{E}\left(\hat{\lambda}'\right) = \mathrm{E}\left(\frac{\sum_{i=1}^{K} M_i - 1}{\sum_{i=1}^{K} \Delta t_i}\right) \tag{E.26}$$

$$= \left(\sum_{i=1}^{K} M_i - 1\right) \mathrm{E}\left(\frac{1}{\sum_{i=1}^{K} \Delta t_i}\right) \tag{E.27}$$

$$= \left(\sum_{i=1}^{K} M_i - 1\right) \left(\frac{\lambda}{\sum_{i=1}^{K} M_i - 1}\right) \tag{E.28}$$

$$= \lambda, \tag{E.29}$$

which shows that $\hat{\lambda}'$ is an unbiased estimator.

Although equation (E.25) is unbiased, it is still not efficient. Computing the variance of $\hat{\lambda}'$, we find

$$\mathrm{Var}\left(\hat{\lambda}'\right) = \mathrm{Var}\left(\frac{\sum_{i=1}^{K} M_i - 1}{\sum_{i=1}^{K} \Delta t_i}\right) \tag{E.30}$$

$$= \left(\sum_{i=1}^{K} M_i - 1\right)^2 \mathrm{Var}\left(\frac{1}{\sum_{i=1}^{K} \Delta t_i}\right) \tag{E.31}$$

$$= \left(\sum_{i=1}^{K} M_i - 1\right)^2 \frac{\lambda^2}{\left(\sum_{i=1}^{K} M_i - 1\right)^2 \left(\sum_{i=1}^{K} M_i - 2\right)} \tag{E.32}$$

$$= \frac{1}{\sum_{i=1}^{K} M_i - 2} \lambda^2. \tag{E.33}$$

Comparing equation (E.33) with (E.23), we clearly see our new estimator is not efficient, but we still have the property that it becomes efficient as $K \to \infty$. However, comparing equation (E.33) with (E.19), we see that $\mathrm{Var}(\hat{\lambda}') < \mathrm{Var}(\hat{\lambda})$.

We should note here that although we have an improvement in both the expected value and variance of our new MLE, the improvement is marginal. The reason is that each $M_i = ND_i$ is large due to the fact that N is large. Specifically, N is typically on the order of 10^3 or larger, so subtracting 1 or 2 has a negligible difference even for small K.

E.4. PERFORMANCE OF THE ESTIMATORS

In order to show that these formulas hold in practice, we ran a Monte Carlo simulation where we defined a true photon arrival rate of $\lambda = 1 \times 10^7$ and a capacitor well size of $N = 800$. For simplicity, we let $D_i = 1$ for all i. Using these parameters, we generated anywhere between 2 and 500 random draws from a gamma distribution, computed photon rate estimates using equations (E.7) and (E.25), and then computed the sample means and variances. These means and variance were compared to the theoretical expected values found in equations (E.14) and (E.29) and theoretical variances found in equations (E.19) and (E.33). The variances were also compared to the CRLB found in equation (E.23).

Figure E-1 shows the theoretical and Monte Carlo estimated means of the estimators. As can be seen, both the original and modified estimators are close to the true photon arrival rate. For small K, the modified estimate is closer, which is to be expected, but the difference is small.

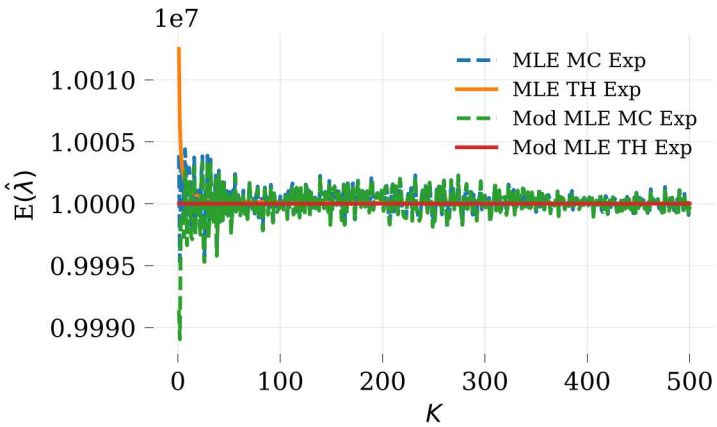

Figure E-1. Mean of the estimators. Here we include lines for the mean of the MLE estimator and the modified estimator. For comparison, we have also included lines for the theoretical expectation of these estimators. MC stands for Monte Carlo, and TH stands for theoretical. As can be seen, the MLE is biased at low N, while the modified estimator is not.

Figure E-2 displays the absolute percent error as K increases. The trend matches that displayed in figure E-1: the two estimators perform about the same, and their performance increases as K increases.

Figure E-3 plots the variance of the estimators as K increases. As expected, the performance increases as K increases, and the variances are indistinguishable from the CRLB.

E.5. DETECTION

The estimators found above allows us to estimate the rate of photon arrivals λ. This rate is specifically the rate of photon arrivals from the background (i.e., no signal of interest present). We will hereafter refer to this as the noise rate λ_N. If a signal of interest is present, we would like to determine a criterion for specifying a signal detection. We would also like to determine the probability of detection, but there is a tradeoff between signal detection and false alarms. That is, we have to set a threshold where if the threshold is set too low, we will likely capture all signals of interest, but also have an unacceptably high false alarm rate. Conversely, if the threshold is set too high, we will have very few false alarms, but we will also miss too many signals of interest depending on the brightness of the signal. Hence, the detection criterion, the probability of false alarm, and the probability of detection will depend on a chosen probability of false alarm for some specified signal brightness. That is, we choose an acceptable probability of false alarm for a signal that has a defined brightness in order to define a detection threshold. Then, for a signal of any

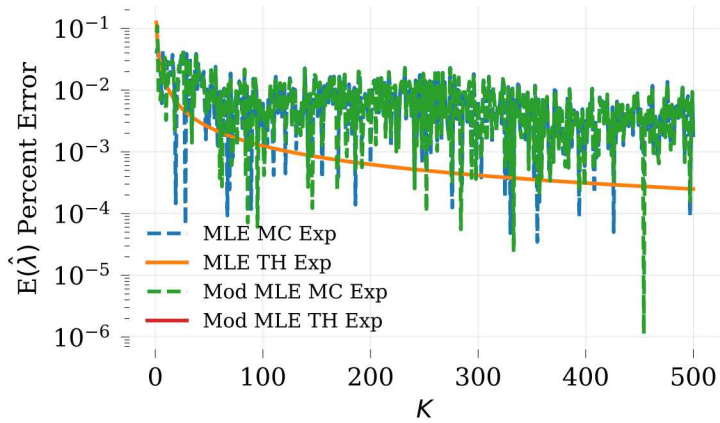

Figure E-2. Percent error of the estimators. The percent errors generally decrease as K increases.

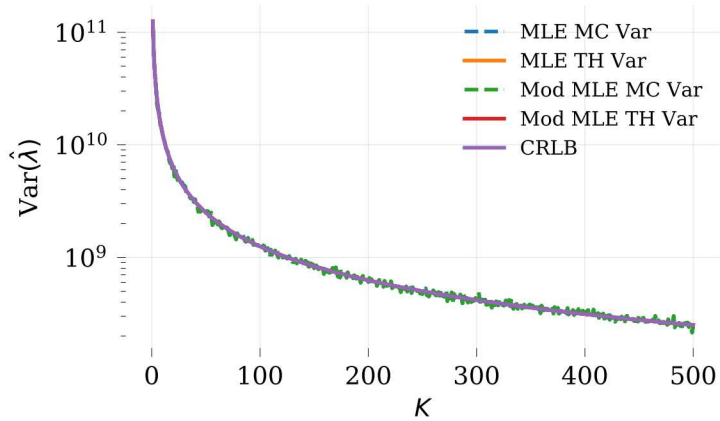

Figure E-3. Variances of the MLEs. All variances are very close to the CRLB.

brightness, we can compute the probability of false alarm and detection given that threshold. Further, we can determine in real time whether a given pixel is witnessing an event of interest using the detection criterion.

To show this, we define λ_S to be the rate of photon arrivals when a signal is present. Here, $\lambda_S > \lambda_N$ because a signal must make the background brighter. In order to define a signal detection criterion, we will use the likelihood ratio test. Specifically, the null hypothesis $\mathcal{H}_0$ is that no signal is present. The alternative hypothesis $\mathcal{H}_1$, by contrast, is that a signal is present. Both hypotheses have a likelihood associated with them. They are

$$\mathcal{H}_0 : \qquad \mathcal{L}(\lambda_N|\mathbf{T},\mathbf{M}) = \prod_{i=1}^{K-1} \frac{\lambda_N^{M_i} \Delta t_i^{M_i-1} e^{-\lambda_N \Delta t_i}}{\Gamma(M_i)}, \tag{E.34}$$

$$\mathcal{H}_1: \qquad \mathcal{L}(\lambda_S|\mathbf{T},\mathbf{M}) = \prod_{i=1}^{K-1} \frac{\lambda_S^{M_i} \Delta t_i^{M_i-1} e^{-\lambda_S \Delta t_i}}{\Gamma(M_i)}. \tag{E.35}$$

The likelihood ratio test states that in order to reject the null hypothesis (i.e., a signal is present), the ratio of the null likelihood to the alternative likelihood must be less than some threshold. That is,

$$LR = \frac{\mathcal{L}(\lambda_N|\mathbf{T},\mathbf{M})}{\mathcal{L}(\lambda_S|\mathbf{T},\mathbf{M})} < \tau \tag{E.36}$$

where LR is the likelihood ratio and τ is the threshold. If we substitute equations (E.34) and (E.35) into (E.36), simplify, and take a logarithm, we find

$$\sum_{i=1}^{K} \left[(\lambda_S - \lambda_N)\Delta t_i - \log\left(\frac{\lambda_S}{\lambda_N}\right) M_i \right] < \log(\tau), \tag{E.37}$$

which is our detection criterion in terms of a chosen threshold τ.

As stated above, we would like to define our detection criterion in terms of a chosen probability of false alarm and signal brightness. To do this, we start by rearranging equation (E.37). We find

$$\sum_{i=1}^{K} \Delta t_i < \frac{1}{\lambda_S - \lambda_N} \left[\log(\tau) + \log\left(\frac{\lambda_S}{\lambda_N}\right) \sum_{i=1}^{K} M_i \right]. \tag{E.38}$$

The probability of a false alarm is the probability of equation (E.38) being true while the null hypothesis is also true. Mathematically, we can write

$$P_{FA} = \Pr\left(\sum_{i=1}^{K} \Delta t_i < \frac{1}{\lambda_S - \lambda_N} \left[\log(\tau) + \log\left(\frac{\lambda_S}{\lambda_N}\right) \sum_{i=1}^{K} M_i \right] \bigcap \mathcal{H}_0 \right) \tag{E.39}$$

$$= \Pr\left(\sum_{i=1}^{K} \Delta t_i < \frac{1}{\lambda_S - \lambda_N} \left[\log(\tau) + \log\left(\frac{\lambda_S}{\lambda_N}\right) \sum_{i=1}^{K} M_i \right] \middle| \mathcal{H}_0 \right) \Pr(\mathcal{H}_0) \tag{E.40}$$

$$= \Pr\left(\sum_{i=1}^{K} \Delta t_i < \frac{1}{\lambda_S - \lambda_N} \left[\log(\tau) + \log\left(\frac{\lambda_S}{\lambda_N}\right) \sum_{i=1}^{K} M_i \right] \middle| \mathcal{H}_0 \right) \tag{E.41}$$

where P_{FA} is the probability of false alarm. The last equality above comes from the fact that if no signal is present, $\Pr(\mathcal{H}_0) = 1$. Since the sum of waiting times is gamma distributed, we can write $Y_N \equiv \sum_{i=1}^{K} \Delta t_i \sim \Gamma(\sum_{i=1}^{K} M_i, \lambda_N)$ where we use λ_N because no signal is present. With this, we have

$$P_{FA} = \Pr\left(Y_N < \frac{1}{\lambda_S - \lambda_N} \left[\log(\tau) + \log\left(\frac{\lambda_S}{\lambda_N}\right) \sum_{i=1}^{K} M_i \right] \right) \tag{E.42}$$

$$= P_\gamma\left(\sum_{i=1}^{K} M_i, \frac{\lambda_N}{\lambda_S - \lambda_N} \left[\log(\tau) + \log\left(\frac{\lambda_S}{\lambda_N}\right) \sum_{i=1}^{K} M_i \right] \right) \tag{E.43}$$

where $P_\gamma(\cdot,\cdot)$ is the regularized, lower, incomplete gamma function and is defined by

$$P_\gamma(s,x) \equiv \frac{1}{\Gamma(s)} \int_0^x t^{s-1} e^{-t}\, dt. \tag{E.44}$$

This comes from the fact that the cumulative density function (CDF) of $X \sim \Gamma(a,b)$ is $P_\gamma(a,bx)$.

If we now let $P_\gamma^{-1}(\cdot,\cdot)$ be the inverse in the second argument of the regularized, lower, incomplete gamma function such that $P_\gamma^{-1}(s,P_\gamma(s,x)) = x$ and $P_\gamma(s,P_\gamma^{-1}(s,x)) = x$, with some algebra we find that

$$\log(\tau) = \frac{\lambda_S - \lambda_N}{\lambda_N} P_\gamma^{-1}\left(\sum_{i=1}^{K} M_i, P_{FA}\right) - \log\left(\frac{\lambda_S}{\lambda_N}\right)\sum_{i=1}^{K} M_i. \tag{E.45}$$

As stated earlier, we need to choose an acceptable probability of false alarm for some signal brightness in order to fully define our threshold. That is, we let $P_{FA} = P_{FA}^*$ and $\lambda_S = \lambda_S^* = \lambda_N(1+\alpha^*)$. Here, $\alpha^* > 0$ is the fraction above the background brightness on which to define an acceptable probability of false alarm. With this, our threshold is fully defined with

$$\log(\tau) = \alpha^* P_\gamma^{-1}\left(\sum_{i=1}^{K} M_i, P_{FA}^*\right) - \log(1+\alpha^*)\sum_{i=1}^{K} M_i. \tag{E.46}$$

We are now able to define our detection criterion. If we set $\lambda_N = \hat{\lambda}_N$ and $\lambda_S = \hat{\lambda}$ in equation (E.38), we find

$$(\hat{\lambda} - \hat{\lambda}_N)\sum_{i=1}^{K} \Delta t_i < \log(\tau) + \log\left(\frac{\hat{\lambda}}{\hat{\lambda}_N}\right)\sum_{i=1}^{K} M_i. \tag{E.47}$$

The idea here is that we have an estimate for $\hat{\lambda}_N$ based off of previous spike rates (a running average of some sort). For every time bin, we compute a new photon rate estimate $\hat{\lambda}$, which is then used in equation (E.47) to determine if the pixel detected an event of interest. If the criterion is not met, we can simply use $\hat{\lambda}$ to update $\hat{\lambda}_N$.

We can now compute a probability of false alarm for any change in brightness. In equation (E.43), we can let $\lambda_S = \lambda_N(1+\alpha)$. We should note that for a false alarm there is no real signal. Hence, in this case, α represents an increase in the instantaneous photon arrival rate although the long term rate λ_N remains constant. With this substitution, we find

$$P_{FA} = P_\gamma\left(\sum_{i=1}^{K} M_i, \frac{1}{\alpha}\left[\log(\tau) + \log(1+\alpha)\sum_{i=1}^{K} M_i\right]\right). \tag{E.48}$$

Equation (E.48) is more useful from a theoretical point of view. That is, it tells you the probability of false alarm supposing that α is known. Since α here represents an instantaneous fractional increase in the photon arrival rate, it cannot be exactly known in practice. However, from a theoretical point of view, equation (E.48) can help us choose suitable values for P_{FA}^* and α^*. After all, we may find P_{FA} to be intolerable for certain values of α even if some P_{FA}^* and α^* by themselves appear tolerable.

Now concerning the probability of signal detection, we assume both the alternative hypothesis $\mathcal{H}_1$ and equation (E.38) are true. If we perform a similar analysis as was done to arrive at equations (E.43) and (E.48), we will find

$$P_D = P_\gamma\left(\sum_{i=1}^{K} M_i, \frac{1+\alpha}{\alpha}\left[\log(\tau) + \log(1+\alpha)\sum_{i=1}^{K} M_i\right]\right). \tag{E.49}$$

Again, this equation is useful from a theoretical perspective and can be used in tandem with equation (E.48) to adjust the parameters P^*_{FA} and α^*.

E.6. DETECTION EXAMPLES

To show how this works in practice, let us assume that there are $M_i = 800$ photon arrivals for every quadtree spike and we wait to receive 35 quadtree spikes ($K = 34$) for a given pixel. We will also specify a probability of false alarm of $P^*_{FA} = 1 \times 10^{-6}$ for a yet to be determined background level α^*. To determine this background level, we need to know the probability of detection if $\alpha = \alpha^*$. If the probability of detection is too low, we either need to choose a different P^*_{FA} or a different α^*. In figure E-4, we show P_D from equation (E.49) using $\alpha = \alpha^*$. We see that for the chosen P^*_{FA}, we have a high probability of detection (about 0.95) for $\alpha^* \geq 0.04$. Hence, in what follows, we will choose $\alpha^* = 0.04$, which means that we want our probability of false alarm to be 1×10^{-6} for a signal that is a factor 1.04 brighter than the nominal background.

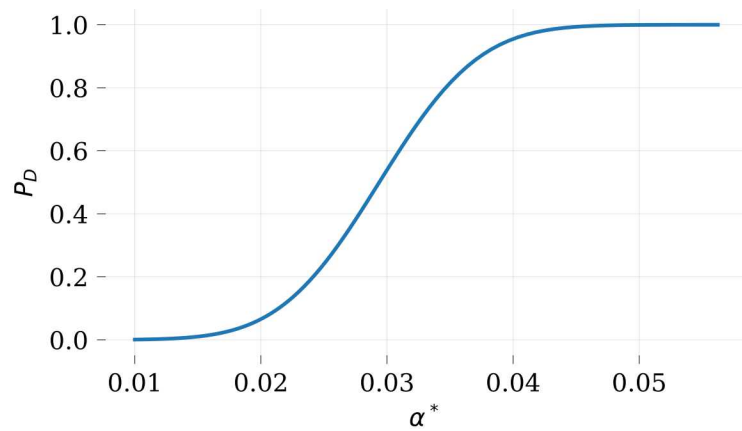

Figure E-4. Probability of detection as a function of signal level assuming $\alpha = \alpha^*$. This plot was made with the choice $P^*_{FA} = 1 \times 10^{-6}$ and $\sum_{i=1}^{K} M_i = 27200$. We can see that choosing $\alpha^* \geq 0.04$ will result in a high probability of detection.

We would now like to see how the probability of false alarm and detection perform with $\alpha^* = 0.04$ for other signal levels. As such, we plot equations (E.48) and (E.49) in figures E-5 and E-6, respectively. Figure E-5 is similar to figure E-4, and still has high probability of detection for $\alpha \geq 0.04$. Figure E-6 shows the probability of false alarm increases over 1×10^{-6} for $0.0189 \leq \alpha \leq 0.04$, which is not unexpected since we are only requiring that $P_{FA} = 1 \times 10^{-6}$ for $\alpha = 0.04$.

We would now like to know if we can reliably detect a signal that is 1% of the background (i.e. $\alpha = 0.01$). It should be clear from figure E-5 that we cannot unless if we wait for more photons. To determine the number of photons, we plotted equation (E.49) with $P^*_{FA} = 1 \times 10^{-6}$, $\alpha^* = 0.04$,

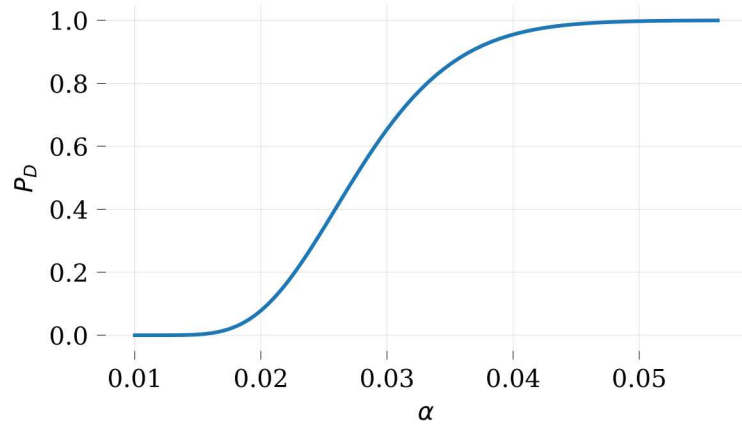

Figure E-5. Probability of detection as a function of signal level. This plot was made assuming $P^*_{FA} = 1 \times 10^{-6}$, $\alpha^* = 0.04$, and $\sum_{i=1}^{K} M_i = 27200$.

and $\alpha = 0.01$ for various values of $\sum_{i=1}^{K} M_i$. This is shown in figure E-7. If we define a reliable detection to be the point where $P_D = 0.9$, we find we need to wait for 59 949 photons (or 75 quadtree spikes), which is a little over twice as many as the 27 200 we were using before. This means that if the background rate was $\lambda_N = 1 \times 10^9$ photon arrivals per second, we could reliably detect a signal that is 1% of the background at a granularity of 16.68 kHz.

If we relax the requirement on the probability of false alarm, we can decrease the number of photon arrivals to reliably detect a signal that is 1% of the background. To show this, we set $P^*_{FA} = 1 \times 10^{-2}$, but keep $\alpha^* = 0.04$, and $\alpha = 0.01$. Similar to before, if we plot equation (E.49) for various $\sum_{i=1}^{K} M_i$, we find we only need 16 357 photon arrivals or 21 quadtree spikes for a reliable detection, which corresponds to 61.14 kHz. This is shown in figure E-8.

Suppose, instead, that we only wanted to consider time intervals at 38 kHz. Then, varying $\sum_{i=1}^{K} M_i$ corresponds to varying λ_N since $\sum_{i=1}^{K} M_i = \lambda_N / 38\,000$. If we set $P^*_{FA} = 1 \times 10^{-6}$ and let $\alpha = \alpha^*$, then we can vary λ_N so that we can choose an appropriate α^* based on the probability of detection. We show the result of doing this in figure E-9. Clearly, a larger background rate is necessary for smaller values of α^*, which is not surprising. The point here is to show that if we define our time interval and reference probability of false alarm, we can determine what the background rate needs to be for any reference signal strength.

Although there may be ways to prevent certain photon wavelengths from entering the sensor, we can't actually choose λ_N since it is depends on the brightness of the background object. However, figure E-9 would still be useful as a comparison to an existing, conventional sensor. That is, if there was a sensor with a frame rate of 38 000 kHz, then we could compare it with the SPARR sensor using the above method.

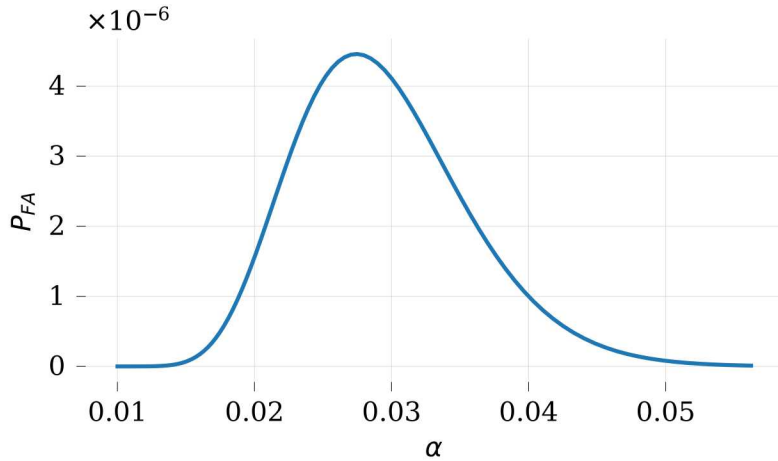

Figure E-6. Probability of false alarm as a function of signal level. This plot was made assuming $P^*_{FA} = 1 \times 10^{-6}$, $\alpha^* = 0.04$, and $\sum_{i=1}^{K} M_i = 27200$.

E.7. RECONSTRUCTION ALGORITHM

We are now able to define a reconstruction algorithm. As photons arrive at a pixel, we can bin the quadtree spikes according to a reconstruction frequency. Then for each bin, we can use equation (E.25) to estimate the photon arrival rate $\hat{\lambda}$ in that time bin. Further, we can overlap each time bin to make a smoother photon rate curve. When there is no event of interest, the computed photon rate is $\hat{\lambda}_N$, which is updated every time bin with a running mean over K time bins. For each time bin, we can use equation (E.47) using some chosen F^*_{PA} and α^* to determine if a signal is present. If a signal is determined to be present, the previous photon rate $\hat{\lambda}_N$ is held constant so that equation (E.47) can use the correct background rate. Once the system no longer detects a signal of interest for a long enough period of time, the background rate is no longer held constant and each time bin starts updating $\hat{\lambda}_N$ again.

E.8. PERFORMANCE SIMULATION

In order to simulate this algorithm, we ran three scenarios in `sparrow`. In all scenarios, the true background photon rate was $\lambda_N = 1 \times 10^{10}/11 \approx 9.091 \times 10^8$ Hz, the capacitor well size was $N = 800$ electrons, the quadtree decimation was $D_i = 1$ (which corresponds to no decimation) for each time bin, and a cosine signal was input between times 0.1 and 0.4 seconds. The reconstruction frequency was always 38 kHz with $P^*_{FA} = 1 \times 10^{-6}$ for $\alpha^* = 0.04$. The amplitude of the cosine varied between scenarios.

In the first scenario, the amplitude was ten times the background rate, which gives $\lambda_S = 1 \times 10^{11}/11 \approx 9.091 \times 10^9$ Hz at the peak of the cosine signal. The signal reconstruction is shown in figure E-10. Here, we see that the reconstruction is very close to the input signal everywhere, and the algorithm detects the signal almost everywhere. The detection criterion fails

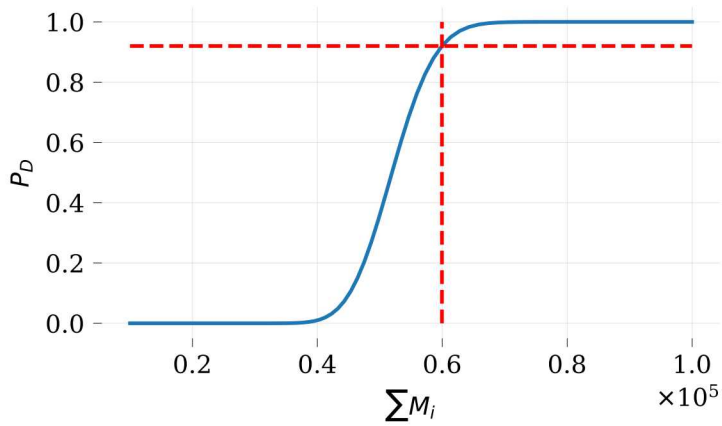

Figure E-7. Probability of detection vs. the number of photon arrivals in the interval assuming $P^*_{FA} = 1 \times 10^{-6}$, $\alpha^* = 0.04$, and $\alpha = 0.01$. As can be seen, if we wait to receive $59\,949$ photons in an interval, we will detect the signal 90% of the time.

when the signal approaches zero, making the photon rate almost equal to the background rate. Mitigating this behavior would not be difficult. We could simply force the system to continue monitoring a pixel that has a detection for some period of time. That is, if we expect all signals of interest to be shorter than some period of time, we can force the system to save all data from a pixel that detected for that period of time independent of whether it continues to make a detection.

In the second scenario, the amplitude was equal to the background rate, which gives $\lambda_S = 2 \times 10^{10}/11 \approx 1.812 \times 10^9$ Hz at the peak of the cosine signal. The reconstruction for this scenario is shown in figure E-11. We can see that the reconstruction algorithm again does well both in reproducing the signal and registering a detection. We also notice, however, that there is more uncertainty in the reconstruction as evidenced by the broader blue line as compared to the previous scenario. Also, the time during which the detection criterion is not met is longer. However, this is mitigated using the same strategy as before.

In the final scenario, the amplitude was set to 0.04 times the background rate, which gives $\lambda_S = 1.04 \times 10^{10}/11 \approx 9.4545 \times 10^8$ Hz at the peak of the cosine signal. This reconstruction is shown in figure E-12. We see that the uncertainty has increased considerably, but more importantly, we do not register a detection until about 0.033 seconds after the signal begins. Again, the failure of the detection criterion toward the end of the signal can be compensated for with the correction mentioned earlier. The failure at the beginning of the signal is not as simple. If we keep the P^*_{FA} and α^* set to their current values, there is likely not much to be done about missing the beginning of the signal. After all, missing detections is the tradeoff for having a low probability of false alarm. If, however, we can tolerate more false alarms, then we can relax either P^*_{FA} or α^* so that the threshold computed by equation (E.46) results in more detections.

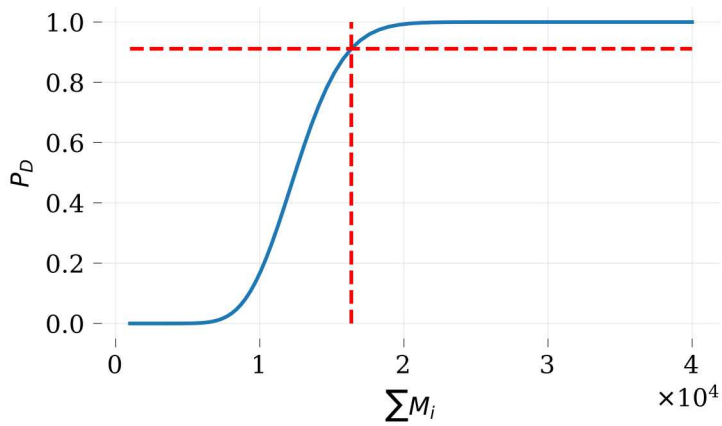

Figure E-8. Probability of detection vs. the number of photon arrivals in the interval assuming $P^*_{FA} = 1 \times 10^{-2}$, $\alpha^* = 0.04$, and $\alpha = 0.01$. As can be seen, if we wait to receive 16357 photons in an interval, we will detect the signal 90% of the time.

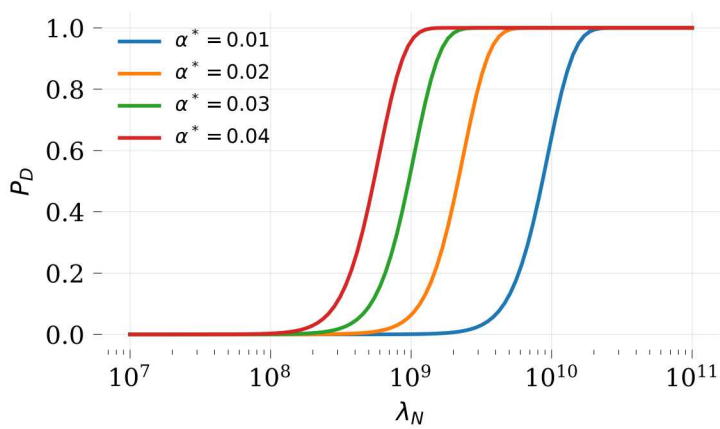

Figure E-9. Probability of detection vs. background photon arrival rate assuming $P^*_{FA} = 1 \times 10^{-6}$, $\alpha = \alpha^*$, and $\sum_{i=1}^{K} \Delta t_i = 1/38000$. As can be seen, increasing α^* results in a greater P_D for a given λ_N.

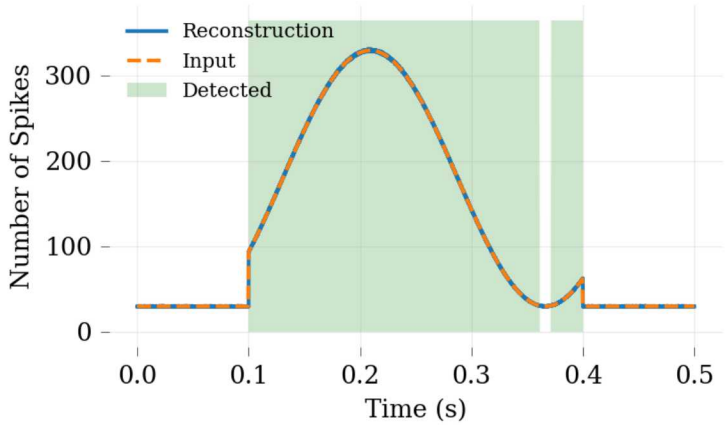

Figure E-10. Signal reconstruction and detection on a target signal with peak magnitude ten times the background.

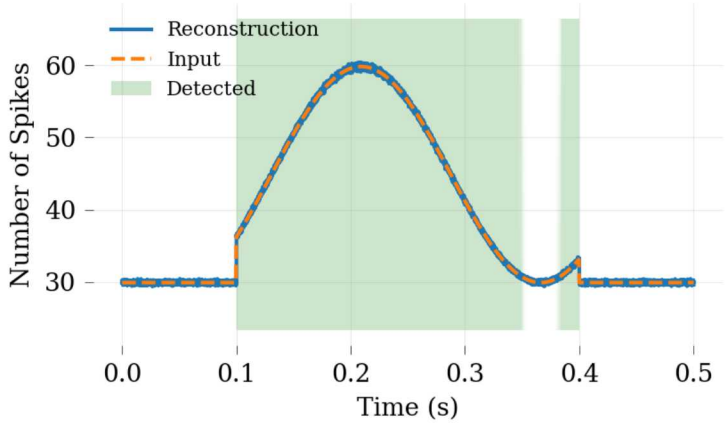

Figure E-11. Signal reconstruction and detection on a target signal with peak magnitude at parity with the background.

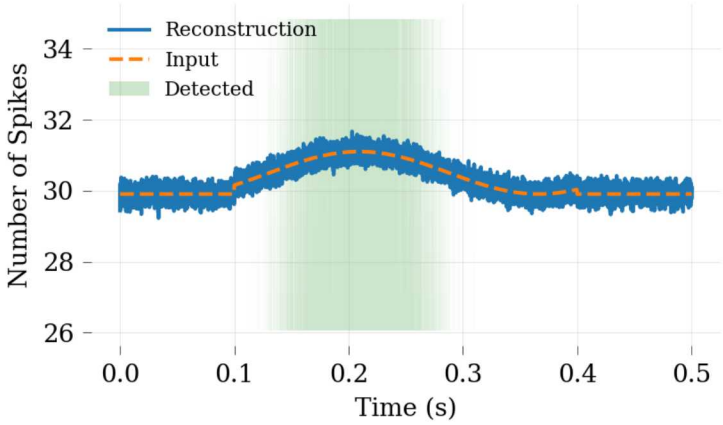

Figure E-12. Signal reconstruction and detection on a target signal with peak magnitude at 4% of the background.

APPENDIX F. TIME DECODING MACHINE FOR RECONSTRUCTING THE INPUT SIGNAL

To create low-power circuitry designs, engineers have begun exploring time encoding machines (TEMS) and time decoding machines (TDMS). TEMS are a way to encode incoming signals as spike trains, which use less power. The complement of time encoding, TDMS are circuits that reconstruct the spike trains into the original waveform. Many such TEM circuits exist, including asynchronous sigma-delta modulators, frequency modulation, IAF neurons, and others [22].

Lazar [21] discusses how to use an IAF model as a TEM. His model, as shown in figure F-1, imposes a continuous signal $x(t)$ on a IAF neuron. The signal $x(t)$ is bandlimited to $[-\Omega,\Omega]$ (in radians per second), and $|x(t)| \leq c$ for some constant c. Furthermore, there is a constant bias signal $b > 0$ in the system with $c < b$. The sum of the incident signal and the bias is divided by a constant κ and then integrated. When the result of this surpasses a threshold δ, a spike occurs. The IAF model may include a refractory period, where the neuron is not integrating while it is dumping charge.

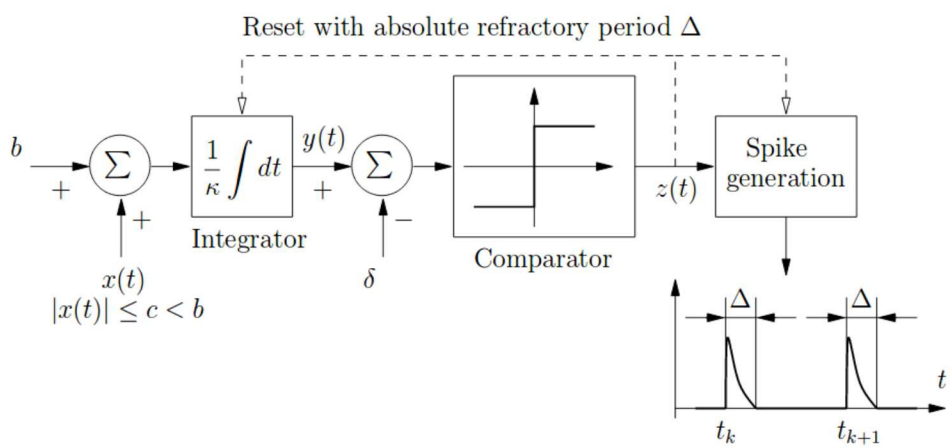

Figure F-1. An IAF neuron. This figure was taken reference [22].

The IAF model discussed needs some modifications to work with SPARR. Begin with

$$i(t) = x(t) + b(t), \tag{F.1}$$

where $i(t)$ is the continuous signal incident on a pixel, $x(t)$ is the signal of interest we care to reconstruct, and $b(t)$ is the background signal. Since these are all measurements of incident light,

$x(t) \geq 0$, $b(t) \geq 0$, and thus $i(t) \geq 0$. $x(t)$ must be bandlimited to $[-\Omega, \Omega]$. For ease of computing theoretical bounds, we assert that $x(t)$ is bounded, that is there exists some constant c such that $x(t) < c$. Finally, we will let $\kappa = 1$ and δ be the number of integrated electrons required to cause the capacitor to dump its charge by spiking. Our system has a refractory period after a spike as well, so will leave the nomenclature of Δ for this.

F.1. AN OVERVIEW OF TDMS

TDMs rely on irregular sampling theory and frame theory. While we attempt to give the reader enough working knowledge in this section, [19] provides a more in-depth analysis of the topic.

Given a chronologically sorted spike train produced by a TEM, a possible representation for $x(t)$ is

$$x(t) = \sum_{k \in \mathbb{Z}} c_k g(t - m_k), \tag{F.2}$$

where $g(t) = \sin(\Omega t)/(\pi t)$ is a low-pass filter with ideal cutoff Ω, the c_k's are the reconstruction coefficients, and each m_k is associated with spike k, but not necessarily the time-stamp of spike k. This is the standard sinc reconstruction for irregular sampling.

For perfect reconstruction with TDMs, k must span all integers, which is not a realistic requirement. Relax this condition and let $k = 0, 1, 2, \ldots, K-1$ for some $K \geq 20$ or so. In practice, this causes inaccuracies (Gibb's phenomenon) in the reconstruction close to the boundaries–typically within two or three samples–of sharp signal amplitude changes. Considering K spikes, the reconstruction can is trustworthy from time s_M to s_{K-1-M} where $2 \leq M \ll K$.

Now, write both sides of equation (F.2) at times t_j for $j = 0, 1, \ldots, J$, and reformulate the equation to be

$$\mathbf{x} = \mathbf{G}\mathbf{c}, \tag{F.3}$$

where $\mathbf{x} \in \mathbb{R}^J$, $x_j = x(t_j)$, $\mathbf{G} \in \mathbb{R}^{J \times K}$, $G_{jk} = g(t_j - m_k)$, and $\mathbf{c} \in \mathbb{R}^K$ is the unknown coefficient vector. To solve for $\mathbf{c}$, we use

$$\mathbf{c} = \mathbf{G}^\dagger \mathbf{x}, \tag{F.4}$$

where $\mathbf{G}^\dagger$ is the pseudo-inverse of $\mathbf{G}$. If $\mathbf{G}$ is square (i.e., $J = K$) and invertible, then $\mathbf{G}^\dagger = \mathbf{G}^{-1}$. Each t_j can be an arbitrary time (i.e., not associated with any spike time s_k), although in practice t_j often describes a regular series of times.

Figure F-2 shows the process for signal reconstruction. A continuous time signal $x(t)$ is run through a generic TEM, which results in a monotonically increasing series of spike times. These spike times are used to compute the coefficients for a weighted time series. This weighted time series passes through a low-pass filter with ideal cutoff Ω. The output of this low-pass filter is the reconstructed signal $x(t)$.

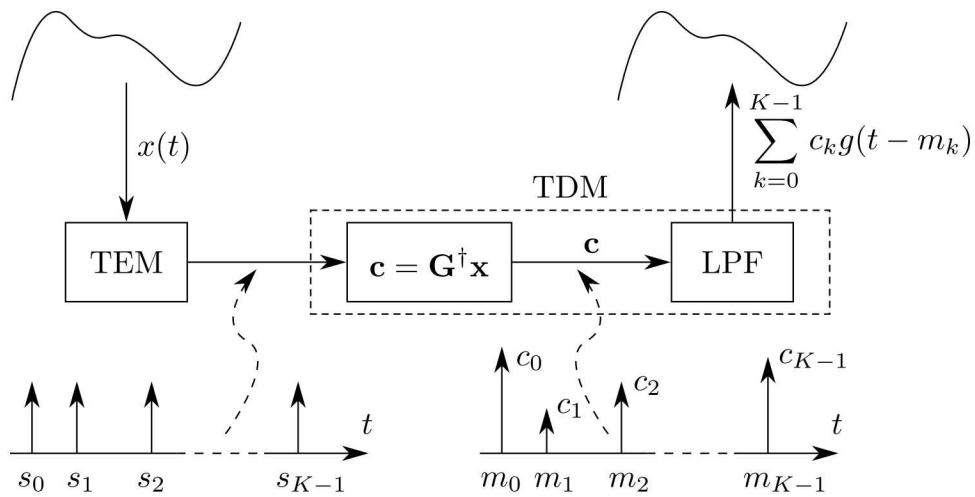

Figure F-2. TDM reconstruction. This figure was taken from reference [22].

F.2. INTEGRATE AND FIRE RECONSTRUCTION

Our model integrates $i(t)$ until a spike occurs. We denote the time of spike ℓ as s_ℓ for $\ell = 0, 1, 2, \ldots, K-1$. Mathematically, we write

$$\begin{aligned} \delta &= \int_{s_\ell+\Delta}^{s_{\ell+1}} i(t)\,dt \\ &= \int_{s_\ell+\Delta}^{s_{\ell+1}} [x(t) + b(t)]\,dt, \end{aligned} \tag{F.5}$$

where δ is the constant integrated amount of signal $i(t)$ between the end of the ℓ's refractory period $(s_\ell + \Delta)$ and the time of the next spike $(s_{\ell+1})$. In most of our applications, the background is a slowly varying signal $b(t)$ and can be treated as piecewise constant. The background contributing to spike ℓ is b_ℓ. For notational convenience, we will drop the subscript here and assume a constant background across all times. It is trivial later on to allow for a slowly moving background. With this simplifying assumption, equation (F.5) reduces to

$$\int_{s_\ell+\Delta}^{s_{\ell+1}} x(t)\,dt = \delta - b(s_{\ell+1} - s_\ell - \Delta)\,. \tag{F.6}$$

To solve for $x(t)$ given the spike train from our IAF TEM, substitute equation (F.2) into (F.6), and manipulate to get

$$\sum_{k=0}^{K-1} c_k \int_{s_\ell+\Delta}^{s_{\ell+1}} g(t - m_k)\,dt = \delta - b(s_{\ell+1} - s_\ell - \Delta), \tag{F.7}$$

where $m_k = (s_k + s_{k+1})/2$ is the midpoint between spikes. Interchanging summation and integration in this case is allowable because the both the integral and summation converge under our constraints. (F.7) then becomes

$$\hat{\mathbf{x}} = \hat{\mathbf{G}}\mathbf{c}, \tag{F.8}$$

where $\hat{\mathbf{x}} \in \mathbb{R}^K$, $\hat{x}_\ell = \delta - b(s_{\ell+1} - s_\ell - \Delta)$, $\hat{\mathbf{G}} \in \mathbb{R}^{K\times K}$, $\hat{G}_{\ell k} = \int_{s_\ell+\Delta}^{s_{\ell+1}} g(t - m_k)\,dt$, and $\mathbf{c} \in \mathbb{R}^K$. Solving for $\mathbf{c}$ and substituting the result into equation (F.3) finds

$$\mathbf{x} = \mathbf{G}\hat{\mathbf{G}}^{-1}\hat{\mathbf{x}}, \tag{F.9}$$

which is our reconstructed signal.

Equation (F.9) is only a good reconstruction of $x(t)$ under certain Nyquist-like constraints. Lazar derives this Nyquist-like constraint in [21], and we will not rederive it here. However, we have modified his derivation for our system. Equation (F.9) is a perfect reconstruction of $x(t)$ if

$$\frac{\delta}{b} + \Delta < \frac{1-\epsilon}{1+\epsilon}\frac{\pi}{\Omega}, \tag{F.10}$$

where $\epsilon^2 = \Delta(b+c)/(\delta + \Delta(b+c))$. Another way of thinking about this is that the maximum bandwidth the system can reconstruct is directly related to the background. We can rearrange equation (F.10) to be

$$\Omega < \left(\frac{1-\epsilon}{1+\epsilon}\right)\left(\frac{b}{\delta + \Delta b}\right)\pi, \tag{F.11}$$

which means that to maximize the system's signal bandwidth, we should minimize the refractory period and minimize the number of electrons per spike. Both of these result in a greater pixel spike frequency, allowing for a shorter average inter-arrival time, and thus more information.

F.3. MODIFICATIONS FOR REAL-TIME

Section F.1 tells us the reconstruction only perfectly recovers $x(t)$ if $\mathbf{G}$ and $\mathbf{x}$ have infinite dimensions, though equation (F.9) produces a good approximation to $x(t)$ for finite matrices given the Nyquist-like constraint of equation (F.10). However, so far the methods have violated causality – we have all of the spike times, s_ℓ, before attempting to reconstruct. To use this algorithm for (near) real-time applications, we introduce a mechanism to adapt Lazar's work for a streaming system [20].

To deal with streaming data, we update the nomenclature slightly. We window the streaming data into sections each containing L spikes, starting at spike m. As such, we define $\mathbf{s} = [s_m, s_{m+1}, \ldots, s_{m+L-1}]$. With this nomenclature, we compute $\hat{\mathbf{x}}_{m,L}$ and $\hat{\mathbf{G}}_{m,L}$ as used in equation (F.8) to create the reconstruction $\mathbf{x}_{m,L}$ using equation (F.9). The resulting reconstruction is imperfect, but nevertheless good except at times near the boundaries of $\mathbf{s}$. In other words, the resulting reconstruction is good over a reduced range $[s_{m+M}, s_{m+L-M}]$ for some $M \geq 2$ [20]). We address this issue by building overlapping consecutive windows of spikes and blending the reconstructions from these overlapped regions.

We formalize the idea of a window, $w = w_n(t, K, M, O)$, where K is the number of spikes in a window, M is the number of spike times around the edge of a window judged to be poorly reconstructed, and O is the number of spikes used for overlapping consecutive windows. For convenience, we define variables (see [20])

$$R = K - 2M - O$$

$$\tau_n = s_{nR+M}$$
$$\sigma_n = s_{nR+M+O},$$

where R represents the number of spikes used exclusively for reconstruction of only one window (i.e. the number of spikes that do not overlap with any other window and are not within the region around the edge of the window), τ_n is the spike time of the beginning of the nth window, and σ_n is the spike time of the first non-overlapping spike in the nth window. Now, we reconstruct $x(t)$ for particular choices of K, M, and O by creating a weighted sum of all contributing windows (see [20])

$$x_{K,M,O}(t) = \sum_{n\in\mathbb{Z}} w_n(t,K,M,O)x_{nR,K}(t). \tag{F.12}$$

We restrict the windowing function w_n to have the form (see again [20])

$$w_n(t,K,M,O) = \begin{cases} 0 & \text{if } t \notin (\tau_n,\sigma_{n+1}], \\ \theta_n(t) & \text{if } (\tau_n,\sigma_n], \\ 1 & \text{if } (\sigma_n,\tau_{n+1}], \\ 1-\theta_{n+1}(t) & \text{if } (\tau_{n+1},\sigma_{n+1}] \end{cases}. \tag{F.13}$$

This windowed-and-stitched reconstruction method can produce $x_{K,M,O}(t)$ with bandwidth in excess of Ω, the bandwidth of $x(t)$. For example, let the Fourier transform of $w_n(t,K,M,O)$ be $W_n(t,K,M,O)$, which is 0 outside of $\pm\Omega_w$. The bandwidth of $x_{nR,K}$(t) is bounded by Ω. However, the bandwidth product of $x_{nR,K}(t)\times w_n(t,K,M,O)$ in equation (F.12) is $\Omega+\Omega_w$, where Ω_w is the bandwidth of the windowing function w. We choose $\theta_n(t)$ to minimize Ω_w. One standard choice recommended by Lazar [20] is

$$\theta_n(t) = \sin^2\left(\frac{\pi}{2}\cdot\frac{t-\tau_n}{\sigma_n-\tau_n}\right). \tag{F.14}$$

Increasing K increases both the accuracy per reconstruction, as well as broadens $w_n(t)$, resulting in a reduced Ω_w. However, this also increases the computational complexity of the inverse of $\hat{\mathbf{G}}_{m,L}$ in equation (F.9), as $\hat{\mathbf{G}}_{m,L}$ is larger.

Passing $x_{K,M,O}(t)$ through a low-pass filter with a cutoff frequency of Ω will restore the original bandwidth of the signal. If required, the samples $x_{K,M,O}(nT_s)$ for $T_s \leq \pi/(\Omega+\Omega_w)$ can be low-pass filtered with digital cutoff frequency $\pi/(1+\Omega_w/\Omega)$ [20]. With the finite overlapping windows and filtering, we can obtain near real-time reconstruction of the input signal.

F.4. DECIMATION IN TIME DECODING MACHINE ALGORITHM

Sparr is a decimating system – the processing in the Haar transform and potential decimation in the quadtree spikes reduces the readout complexity, collisions, and power consumption of the system. Commonly, each spike coming from sparr represents several asp spikes. The decimation is deterministic if no collisions occur, so a weight can be assigned to each read-out spike. For example, if a spike train of four successive spikes passes through 2:1 decimation, then only the second and the fourth spike would be output. Each of these spikes would be associated with a

weight of 2. Weighting, done off-array, represents the internal information while also reducing the rate of readout. For the purposes of the TDM algorithm, quantization-and-weight can be treated as quantization noise. The resulting reconstruction has larger errors and narrower bandwidth since it reduces the available Nyquist rate by throwing away spikes. The equations require some modification to make use of the spike weight, or amplitude.

Define the amplitude of spike k as A_k. Equation (F.6) then becomes

$$\int_{s_\ell + A_\ell \Delta}^{s_{\ell+1}} x(t)\,dt = \delta - b(s_{\ell+1} - s_\ell - A_\ell \Delta). \tag{F.15}$$

Note that we integrate from $s_\ell - A_\ell \Delta$ instead of $s_\ell - \Delta$ to accommodate for the added refractory periods lost with the decimation of a spike. While this codifies the integration of the proper *duration* of time, it does not codify the correct *exact* time of $x(t)$. We would prefer

$$\int_{s_\ell + \Delta}^{s_{\text{decimated}}} x(t)\,dt + \int_{s_{\text{decimated}} + \Delta}^{s_{k+1}} x(t)\,dt. \tag{F.16}$$

However, we no longer know $s_{\text{decimated}}$, so we approximate this with equation (F.15). In practice, Δ is usually small enough that the resulting error is small. With these modifications, we can update

$$\hat{G}_{\ell,k} = \int_{s_\ell + A_\ell \Delta}^{s_{\ell+1}} g(t - s_k)\,dt \tag{F.17}$$

and

$$\hat{x}_\ell = A_\ell \delta - b(s_{\ell+1} - s_\ell - A_\ell \Delta) \tag{F.18}$$

and use equation (F.9) to recover $x(t)$.

The total number of encoded spikes is correctly accounted for, but the time of the discarded spikes is not. Our approach has effectively placed all the decimated spikes on top of the next available spike, which is mathematically and computationally straightforward but also implausible since the refractory period prevents overlapping spikes. In short, there are probably optimizations that would induce less average error at the expense of greater computational effort.

APPENDIX G. PIECEWISE CONSTANT RATE APPROXIMATION

G.1. RECONSTRUCTION

G.1.1. Estimation

We can model the arrival times of photons on a given pixel of the detector as realizations of a nonhomogeneous Poisson process with a rate function $\lambda(t)$. As such, the probability that K photons are received starting at some initial time t_I and ending at a final time t_F is

$$\Pr(K \text{ photon arrivals}) = \frac{e^{-\Lambda(t_I,t_F)}\left[\Lambda(t_I,t_F)\right]^K}{K!}, \tag{G.1}$$

where $\Lambda(t_I,t_F)$ is the expected number of photon arrivals between t_I and t_F. This is calculated via

$$\Lambda(t_I,t_F) = \int_{t_I}^{t_F} \lambda(t)\,dt. \tag{G.2}$$

Suppose that photons arrive on a given pixel at times t_i for $i = 0,1,\ldots,K-1$. If we approximate the rate function $\lambda(t)$ to be a piecewise constant, we can write

$$\lambda(t) = \sum_{i=0}^{K-1} \lambda_i \left[H(t-t_{i-1}) - H(t-t_i)\right] \tag{G.3}$$

where $H(\cdot)$ is the Heaviside step function and each λ_i is an unknown constant. If we compute $\Lambda(t_0,t_{K-1})$ using equations (G.2) and (G.3), we find

$$\begin{aligned}
\Lambda(t_0,t_{K-1}) &= \int_{t_0}^{t_{K-1}} \lambda(t)\,dt && \text{(G.4)}\\
&= \int_{t_0}^{t_{K-1}} \sum_{i=0}^{K-1} \lambda_i \left[H(t-t_{i-1}) - H(t-t_i)\right] dt && \text{(G.5)}\\
&= \sum_{i=0}^{K-1} \lambda_i \int_{t_0}^{t_{K-1}} \left[H(t-t_{i-1}) - H(t-t_i)\right] dt && \text{(G.6)}\\
&= \sum_{i=0}^{K-1} \lambda_i (t_i - t_{i-1}). && \text{(G.7)}
\end{aligned}$$

If we instead consider the interval (t_{j-1}, t_j), we find

$$\Lambda(t_{j-1}, t_j) = \sum_{i=0}^{K-1} \lambda_i (t_i - t_{i-1}) \delta_{ij} \tag{G.8}$$

$$= \lambda_j (t_j - t_{j-1}) \tag{G.9}$$

$$= \Lambda_j \tag{G.10}$$

where δ_{ij} is the Kronecker delta.

We can now use a MLE to compute each λ_j. Due to the piecewise constant model as well as the independence of the arrival times of the photons, we can maximize the joint likelihood by maximizing the likelihood of each individual piecewise approximation. As such,

$$\mathcal{L}(\lambda_i \,|\, t_{j-1}, t_j, N) = \frac{e^{-\Lambda_j} \Lambda_j^N}{N!}, \tag{G.11}$$

where N is the number of photons received in order to generate a spike. This leads to a log-likelihood of

$$\ell(\lambda_i \,|\, t_{j-1}, t_j, N) = -\Lambda_j + N \log(\Lambda_j) - \log(N!) \tag{G.12}$$

$$= -\lambda_j (t_j - t_{j-1}) + N \left[\log(\lambda_j) + \log(t_j - t_{j-1}) \right] - \log(N!) \tag{G.13}$$

In order to find the MLE, we take the derivative of equation (G.13) with respect to λ_j and set the result equation to zero. Solving for λ_j, we find

$$\hat{\lambda}_j = \frac{N}{t_j - t_{j-1}}, \tag{G.14}$$

which is exactly what we would expect. If all we know is two pixel spike times and the size of the electron well, we would estimate the photon arrival rate with equation (G.14). Hence, the best approach is to solve for each λ_j between spike times to estimate $\lambda(t)$.

We can extend this estimator to "average" over the data to have a smoother estimate of the signal. Since the estimator itself is not dependent on the specific arrival times of the spikes, but rather the difference between the arrival times (i.e., the time over which the photons have been collected by the detector), we can "downsample" and "average" the data. Rather than estimate the photon rate for every spike, we could estimate the rate over every M spikes. The denominator of G.14 would change to the time it took to collect M spikes, and the numerator would change to MN. This would result in a reduction of estimated coefficients λ_i by a factor of M. Also, as we will see in the next section, this reduces the variance of the estimator by a factor of M.

G.1.2. Performance

In order to characterize the performance of the above MLE, we will compute the CRLB. This is given by

$$\text{CRLB} = -\frac{1}{\mathrm{E}\left(\dfrac{\partial^2 \ell(\lambda_j)}{\partial \lambda_j^2} \right)} \tag{G.15}$$

$$= -\mathrm{E}\left(-\frac{\lambda_j^2}{N}\right) \tag{G.16}$$

$$= \frac{\lambda_j^2}{N}. \tag{G.17}$$

Since the CRLB is a lower bound on the variance of the MLE, we have

$$\mathrm{Var}(\hat{\lambda}_j) \geq \frac{\lambda_j^2}{N}. \tag{G.18}$$

This implies that the uncertainty in the MLE increases with the square of the rate of photon arrivals. We can decrease the variance by averaging over multiple spikes as suggested earlier. This results in

$$\mathrm{Var}(\hat{\lambda}_j) \geq \frac{\lambda_j^2}{MN}. \tag{G.19}$$

This gives a smaller variance at the cost of a more crude rate function.

If instead of measuring the point-wise variance, we can measure the variance of the estimated rate as a whole. The joint likelihood for this is given by

$$\mathcal{L}(\lambda \,|\, t_0, t_{K-1}, N) = \prod_{i=0}^{K-1} \frac{e^{-\Lambda_i}\Lambda_i^N}{N!}, \tag{G.20}$$

which yields a joint log-likelihood of

$$\ell(\lambda \,|\, t_0, t_{K-1}, N) = -\sum_{i=0}^{K-1} \Lambda_i + N \sum_{i=0}^{K-1} \log(\Lambda_i) - \log(KN!) \tag{G.21}$$

$$= -\sum_{i=0}^{K-1} \lambda_i (t_i - t_{i-1}) + N \sum_{i=0}^{K-1} \left[\log(\lambda_i) + \log(t_i - t_{i-1})\right] - \log(KN!). \tag{G.22}$$

If we take two derivatives with respect to λ_j, we again find the CRLB is given by equation (G.18).

G.1.3. Implementation

The estimation algorithm derived above is a very simple algorithm and can be implemented in silicon very easily. Everything can be done based off of the time of the spikes of a particular pixel. Since we know the value of N, the only thing required are the spike times. Hence, if we have an on-chip clock reference, then each pixel spike can be time-stamped. Any pixels that have an unusually high number of spikes for a period of time can then be transmitted to a processor for further analysis.

If we wanted to perform signal reconstruction on-chip as well, each pixel could contain a counter, a resettable clock at some fine resolution, and a divider. The clock starts at time 0. Each time M spikes occur (assuming we are using the averaging estimator outlined earlier), the clock latches, the estimate for that time is set to $MN/\Delta t$, and the clock is reset. These estimates, time-stamped at the center of the interval, could be run through a matched filter on-chip for signal-of-interest detection.

DISTRIBUTION

Hardcopy—Internal

Number of Copies	Name	Org.	Mailstop
1	D. Chavez, LDRD Office	1911	0359
1	Legal Intellectual Property	11500	0161

Email—Internal (encrypt for OUO)

Name	Org.	Sandia Email Address
Philip L. Dreike	6700	dreikepl@sandia.gov
Stephen M. Gentry	6755	smgentr@sandia.gov
Daniel A. Gonzales	6741	dgonza2@sandia.gov
Matthew J. Kagie	5448	mjkagie@sandia.gov
David B. Karelitz	9326	dbkarel@sandia.gov
Randolph “Rex” Kay	5212	rrkay@sandia.gov
Jeffrey A. Mercier	6770	jamerci@sandia.gov
John S. Mincey	5253	jsmince@sandia.gov
Mark C. Woods	6771	mcwood@sandia.gov
Technical Library	01177	libref@sandia.gov

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.