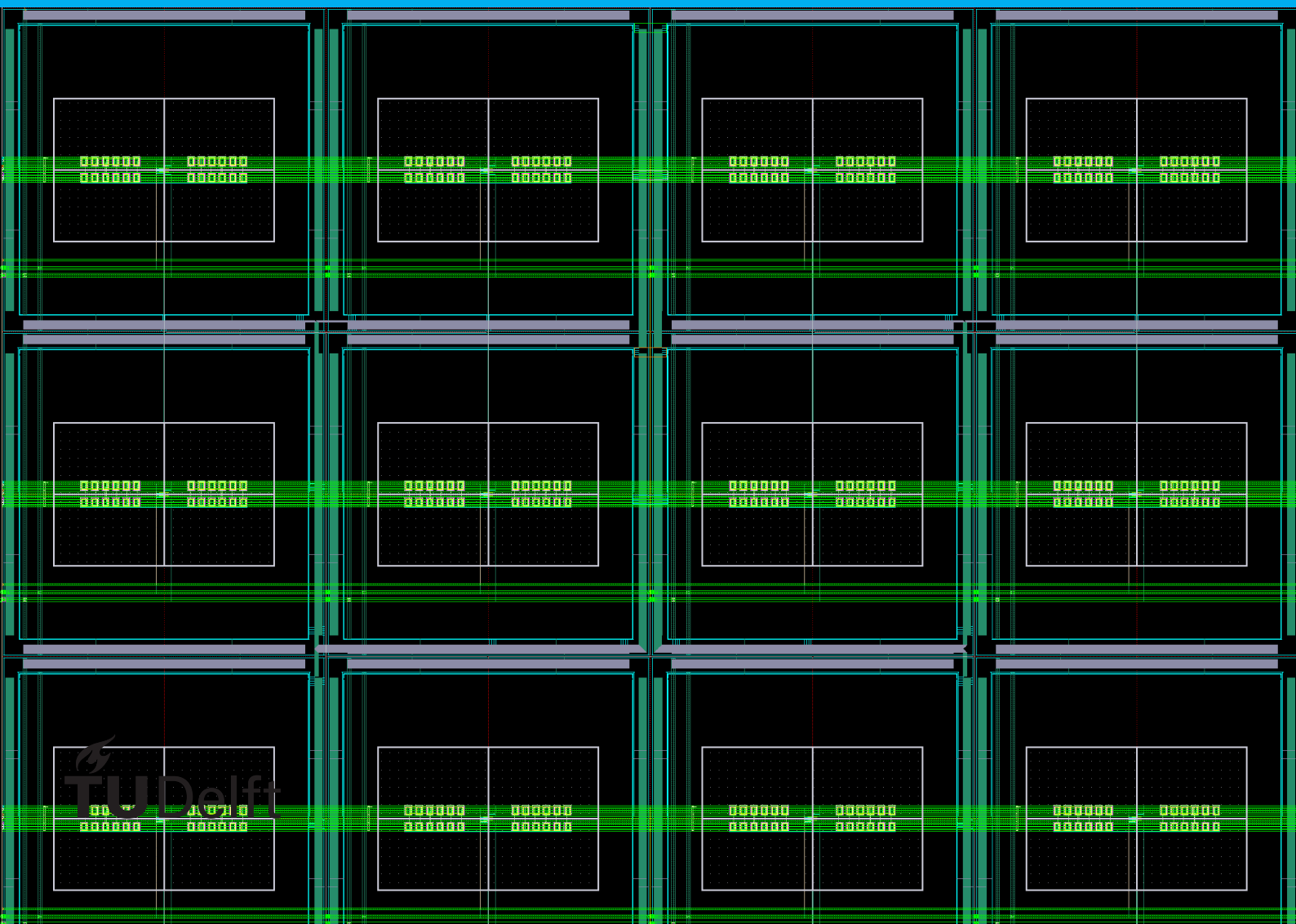
A Low-Power Detector Readout ASIC based on Sparse-Event Counting for Scanning-Electron Microscopy

A.J. Smit

ISBN 978-94-6518-266-7



A Low-Power Detector Readout ASIC based on Sparse-Event Counting for Scanning-Electron Microscopy

by

A.J. Smit

ISBN 978-94-6518-266-7

to obtain the degree of Master of Electrical Engineering
at the Delft University of Technology,
to be defended publicly on February 23, 2026

Student number:	4532023	
Project duration:	February 1, 2024 – February 23, 2026	
Thesis committee:	Prof. dr. S. Nihtianov	TU Delft, supervisor
	dr. ir. M.A.P. Pertijs	TU Delft, chair
	Prof. dr. ir. G.N. Gaydadjiev	TU Delft, core member

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Preface

This thesis was written with the help of the grammar and language tool 'Writefull'.

This thesis appeared quite challenging, as there was little to no research available on this specific subject. Additionally, Hermes Microvision inc. (HMI) was unable to provide any relevant information regarding the incidence patterns of electrons at the sensor, a subject that is at the heart of this thesis. Therefore, I set out to create a basic model to predict such incidence patterns to determine the event rates of individual detectors. This model is based of the work of Goldstein et. al [1], Koshikawa and Shimizu [2] and Niedrig [3].

A.J. Smit
Delft, February 2025

Summary

As feature sizes of integrated circuits become smaller, validation during production with existing scanning electron microscope technologies becomes more challenging. To validate integrated circuits with small feature sizes, a novel Scanning Electron Microscope (SEM) sensor architecture has been patented [4]. Rather than having only a few larger detectors, the proposed sensor architecture consists of a grid of small detectors. This significantly reduces the parasitic capacitance of the detectors, allowing for a much better signal-to-noise ratio (SNR). As a result, the primary beam energy of the SEM can be lowered, allowing imaging of much smaller features while also preventing damage to the integrated circuit. To avoid the buildup of surface charge and provide an acceptable imaging rate, the proposed SEM-sensor will have a sample rate of 400 MHz.

Compared to conventional SEM-sensor, with only 1-10 detectors, the proposed architecture will have approximately 16.384 detectors. This poses a great challenge to the backend readout of the sensor. For each clock cycle, the outputs of all 16.384 detectors have to be added together to provide the total number of detected electrons. In this thesis, various digital and analog backend readout architectures have been proposed. Three of these, namely the full adder tree, critical adder tree, and pulse counter, have been designed, implemented, and compared against one another.

The full adder tree is largely based on the Wallace Tree, which has been implemented in a pipelined fashion. The critical adder tree improves upon this design by omitting the addition of higher order bits based on the expected event rate of the system. Both of these designs are synchronous and, therefore, require a clock signal to synchronize their inputs and outputs.
The pulse counter is a novel readout architecture that allows the inputs to arrive asynchronously while providing synchronous outputs. The rising edges of the asynchronous inputs have been converted into short pulses. These pulses were counted using a modified shift register, where each pulse shifts a digital '1' one position. This has been achieved by matching the path delay between the output of the current latch and the next latch to be slightly larger than the width of the pulses (165 ps). As a result, a single pulse can only ever shift the modified shift registers by one position.
Furthermore, whenever two asynchronous inputs arrive within a short time of each other, the pulses will overlap, and the total width of the pulse as seen by the shift register will be wider than that of a single input. As the propagation delay between the latches has been matched to the width of a single pulse, any wider pulses cause the modified shift register to shift by two positions. The output of the modified shift register forms a thermometer code and has been encoded as a digital signal that is provided as a synchronous output.

All three of these architectures were implemented for a small detector matrix of 16 $\times$ 16 detectors. They have been compared with each other based on their power consumption and error probability for different expected event rates of the detectors, ranging from 0.01 to 4 total events per 2.5 ns within the 16 $\times$ 16 detector matrix. The full adder tree was demonstrated to perform reliably at all event rates with a power consumption of 62 mW, which was dominated by the clock distribution to each individual detector to provide the synchronous inputs. The critical adder tree performed reliably up to the event rate for which it was designed and had a slightly lower power consumption of 60.5 mW, once again dominated by the clock distribution. Finally, the pulse counter performed reliably up to $6.0E^5$ events per second, while only consuming 1.26 mW, attributed to the fact that asynchronous inputs do not require the clock to be routed to each individual detector.

The entire sensor has been designed as a matrix of 128 x 128 detectors. The clock distribution has been implemented using a balanced H-Tree, resulting in a clock skew within 100 ps throughout the detector matrix. The final backend readout architecture has been implemented through a pipelined design, where the results are stored using memory elements after each pipeline segment. Thus, allowing the critical path of each pipeline segment to be equal to the clock period of 2.5 ns, while guaranteeing

the readout for each clock cycle. The first pipeline segment provided the sum of 256 detectors. The outputs of four of these segments were added by the second pipeline segment to provide the sum of 1024 detectors. The outputs of four of these segments were added by the third pipeline segment to provide the sum of 4096 detectors. Finally, the outputs of four of these segments were added by the final pipeline segment to provide the sum of the complete detector matrix.
The readout architectures used for the first pipeline segment were determined based on the expected event rate of the 256 detectors within the pipeline segment. Those with an expected event rate below 0.3 events per 2.5 ns were implemented using the pulse counter architecture, while the remaining pipeline segments were implemented using the critical adder tree architecture. The second, third and fourth pipeline segments were implemented using the critical adder tree architecture.
To estimate the expected event rate throughout the detector matrix, a model has been created. This model roughly estimates the probability of electron incidence at the detector matrix based on the physical behavior of electrons in an SEM. The first pipeline segments at the center of the detector matrix were shown to have an expected event rate below 2.5 events per 2.5 ns and have therefore been implemented using the critical adder tree architecture to minimize their error rate. The remaining first pipeline segments were shown to have an expected event rate below 0.15 events per 2.5 ns and therefore have been implemented using the pulse counter architecture to minimize their power consumption .

To further reduce the power consumption of the system, it has been designed using double data rate (DDR) memory, where data is stored at both rising and falling clock edges. This allowed the system's clock to run at 200 MHz rather than 400 MHz, reducing the power consumption of the clock distribution from 0.616 W to 0.446 W.
The total power consumption of the system using a mix of pulse counters and critical adder trees for the first pipeline segments was found to be 0.9 W, with an error rate of 232 parts per million. Compared to the power consumption of the system using only critical adder trees, which was 4.3 W with an error rate of 0 parts per million. This significant reduction illustrates the trade-off that can be made between the power consumption and error rate.

Contents

List of Figures

List of Tables

List of abbreviations

ADC - Analog to Digital Converter
ASIC - Application Specific Integrated Circuit
BSE - Back-Scattered Electrons
CCRO - Current Controlled Ring Oscillator
CDF - Cumulative Distribution Function
CMOS - Complementary Metal Oxide Semiconductor
CRAT - Critical Redundancy Adder Tree
CSA - Carry Save Adder
DDR - Double Data Rate
DETDFF - Dual Edge Triggered D-flipflop
DFF - D-flipflop
DLT - Digital Logic Trench
DWF - Dense Wire Fabric
EMI - Electro-Magnetic Interference
FRC - Front-end Readout Circuit
HMI - Hermes Microvision inc.
IC - Integrated Circuit
JIT - Just-In-Time
LiDAR - Light Detection and Ranging
PCIB - Pulse Counter with Input Buffer
PE - Primary Electron
PDK - Process Design Kit
PLL - Phase Locked Loop
ROIC - Readout Integrated Circuit
RMS - Root Mean Square
SCL - Standard Cell Library
SE - Secondary Electrons
SEM - Scanning Electron Microscope
SNR - Signal to Noise Ratio
SPAD - Single Photon Avalanche Diode
SRAM - Static Random Access Memory
SUT - System Under Test
TDC - Time to Digital Converter
TSMC - Taiwan Semiconductor Manufacturing Company
TSV - Through Silicon Via
TTL - Through-the-Lens

1 Introduction

This thesis focusses on the design of an application-specific read-out integrated circuit (ASIC) for a sparse Bernoulli matrix. A Bernoulli source has a certain probability p that an event occurs, resulting in a digital high output. In case no event occurs, the output is a digital low. Given a matrix containing $m \times n$ Bernoulli sources, the total number of events might be of interest. If the number of events is small compared to the number of elements within the matrix, it is referred to as a sparse matrix [5]. The Bernoulli sources are spaced at a constant pitch. The Read-Out ASIC should determine the total number of events throughout the matrix at a fixed sample rate. Furthermore, each Bernoulli source within the matrix is reset after a sample period.

Such an ASIC could be applied in scanning electron microscopy (SEM), a type of microscope that greatly exceeds the resolution of conventional optical microscopes. The resolution of optical microscopes is limited by the wavelength of light that is used for imaging and has a theoretical limit of approximately 250 nm [6]. An SEM, however, uses electrons for imaging by focusing a continuous beam of electrons down to a very small point on the surface of the sample where they interact with the atoms of this sample [1]. The electron beam scans the sample surface in a grid-like pattern, and the electron interactions are measured for every point on this grid. As a result, a highly detailed image can be formed with a resolution of 1-20 nm [7], as illustrated in Figure1.1.

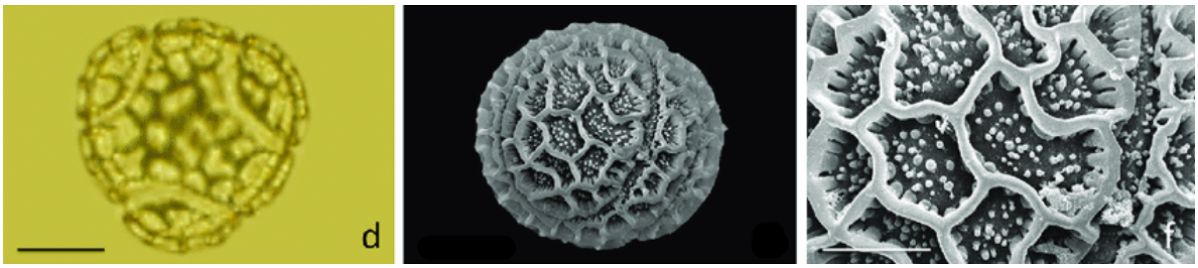


Figure 1.1: Image of pollen using an optical microscope (left) and SEM (center, right)[8]

To facilitate higher resolution SEM's, a detector capable of detecting a single low charge electron was designed by Disi, Zaki, and Nihtianov [9] [10]. This detector can be used to construct an SEM sensor by arranging the detectors in an array, as proposed by Wang et al. [4]. Any charge present on the surface of the sample will deflect electrons, resulting in a visible distortion of the image, as illustrated in figure 1.2. [11]. Consequently, the sample rate of this system must be large enough to prevent surface charging from the SEM beam and to provide an acceptable scanning speed. An individual detector can be described as a low probability Bernoulli source, where an event represents a detected electron. As a result, the sensor can be abstracted and seen as a sparse matrix of Bernoulli sources with a high sample rate. This specific application will be the subject of the read-out ASIC designed in this thesis.

In the first section of chapter 2 the above-mentioned SEM sensor is described in more detail and the starting point for this thesis is defined. Next, the requirements for the read-out ASIC are presented.

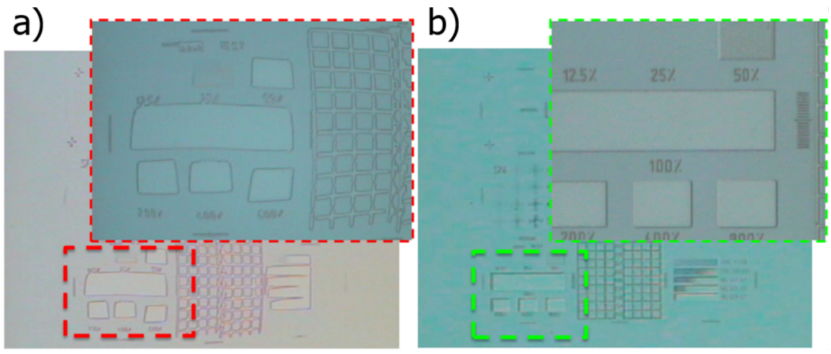


Figure 1.2: SEM scan of sample with surface charge (A) and without surface charge (B) [12]

Using these requirements, the read-out ASIC is broken down into different subsystems, and the preliminary high level architecture of the ASIC is presented. The remainder of the chapter focuses on the backend readout subsystem within the ASIC by discussing existing readout architectures and proposing various novel readout architectures. The chapter is concluded by evaluating the proposed readout architectures on the basis of the requirements. Three of the proposed architectures are identified as suitable candidates. Chapter 3 explores the three readout architectures in more detail, starting with their design and then evaluating their error performance and power consumption based on their simulation results. In chapter 4 the design of the ASIC is covered. The simulation results of the ASIC are presented and discussed in chapter 5.2. Finally, the thesis is concluded in chapter 6 with recommendations for future work.

Although this thesis will not focus on the design of an SEM, those interested in the operating principles of an SEM are encouraged to read appendix A. The concepts discussed there are used to derive a statistical model of electron incidence on a sensor in appendix 3.1. For those who are not familiar with integrated circuit layout, Appendix C provides the background required to understand chapter 4

2

Background

This chapter introduces the system that will be designed and how its role within an application specific integrated circuit (ASIC). First, the starting point of this work will be defined along with the requirements of the system and the ASIC in section 2.1. With these requirements, the system is broken down into subsystems, allowing us to identify the back-end readout subsystem as the most challenging to implement. In section 2.2, various back-end readout architectures of image sensors are presented, and their relevance to the proposed system is evaluated. This will illustrate that the proposed sensor has little to no similarities to existing imaging sensors, and therefore the backend readout architecture cannot borrow from existing systems. Consequently, in section 2.3 various analog and digital backend readout architectures are proposed and evaluated based on the system requirements.

2.1. Proposed readout ASIC

This thesis will focus on the exploration, design, and implementation of an ASIC that provides the readout functionality for an event counting based scanning electron microscope sensor. Within the context of this thesis, the readout is achieved by adding the outputs of each Bernoulli source and providing the total sum within the associated sampling period.

A SEM operates by focusing a primary electron (PE) beam onto the imaging area in a grid-like pattern. Each location on this grid is referred to as a pixel. The value of each pixel is determined by the number of electrons that are reflected through elastic scattering, referred to as back-scattered electrons (BSEs), and inelastic scattering, referred to as secondary electrons (SEs) [1]. This behavior is explained in more detail in appendix A. Conventionally, reflected electrons are detected using large detectors, whose voltage increases with the number of incident electrons. This voltage is used as the pixel value.

The SEM sensor proposed in US Patent 0123152A1 [4] consists of a matrix of many small detectors (the detector matrix) that are capable of detecting individual low-energy electrons. Consequently, a matrix of these detectors can discretely count the number of incident electrons within its area. Within the proposed SEM sensor the discrete number of incident electrons would be used directly as the pixel value, rather than the analog voltage. In the following sub-section the starting point for this work will be presented, followed by the requirements that must be met by the ASIC.

2.1.1. Foundation for the proposed work

The detectors, along with three separate front-end readout circuits (FRC), have been designed by Disi, Zaki, and Nihtianov [9] [10] [13]. These are capable of detecting a single low-energy electron. The detector itself is located on the back-side of the silicon is 165 μm x 165 μm. The FRC is located on the front-side of the silicon and occupies approximately $73\mu m$ x $113\mu m$. The general architecture of the FRCs is shown in figure 2.1 and is described below:

- **CSA + Active Shaper**: Incident electrons on the semiconductor detector charge the parasitic capacitance of the detector. The CSA amplifies the voltage over this capacitance. As a result,

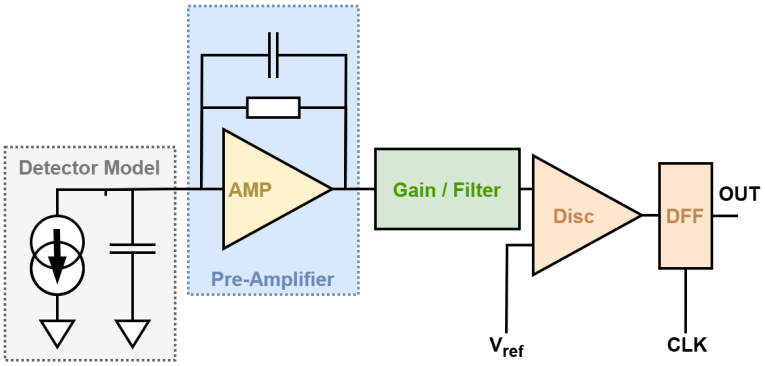


Figure 2.1: General architecture of the front-end readout circuit. The detector is modeled as a current source with some parasitic capacitance. The current produced by incident electrons is amplified by the pre-amplifier, whose output is processed using either a passive or active filter. The output of the filter is connected to a discriminator which converts the analog signal to a digital one which is stored using a DFF.

the rise time at the CSA output is very fast, but the charge slowly leaks away, resulting in a slowly falling tail. To filter out only the high-frequency components, an active shaper is used. The output of this active shaper is processed to a digital signal using a discriminator.

- **CSA + Passive Shaper**: Operates using the same principle as the CSA + Active Shaper. However, the shaper is replaced by a passive shaper to reduce power consumption. Consequently, the discriminator has to be updated to reduce the input-referred noise and offset. This is done using the autozeroing technique every 90 μs for 10 ns. During these 10 ns the FRC is not able to process the signals from the detector.

- **TIA**: The current from the semiconductor detector is converted to a voltage using a transimpedance amplifier. This voltage is then amplified and processed to a digital signal using a discriminator.

All three of these FRC architectures provide a similar output, either a digital '1' or a digital '0' for the duration of each sampling period. The output can be said to have probability p of being '1' and probability $1 - p$ of being '0'. Therefore, the detector + FRC output can be modeled using a Bernoulli distribution as given by equation 2.1 [14].

$$X_n \sim Bernoulli(p) \rightarrow P(X_n = 1) = p \tag{2.1}$$

Figure 2.2 shows a single detector with an FRC and illustrates how the detector matrix is constructed. The event rate at the detectors near the center of the detector matrix is expected to be six events every 100 ns. The 3σ interval for the total number of events within the detector matrix over 2.5 ns is expected to be 50 (i.e. 99.7% of the time less than 50 events occur within 2.5 ns) while the mean is expected to be 10.

Each FRC has the following connections:

1. **Supplies:** VDD, VSS, VDD_AN, VSS_AN
2. **Digital:** CLK, RST'
3. **Output:** EVNT
4. **Analog Reference:** Vb1, Vb2, Vnet_lsh, Vth4, Vth2

The analog references are the same for all front-end readout circuits and must be as stable as possible. For that reason, these must be properly shielded, and special care must be taken to ensure that fast switching (digital) signals do not run parallel to any of these reference supply lines for long distances. Analog circuits have a certain degree of supply noise rejection; therefore, noise on the analog supply is preferred to noise on the reference voltages.

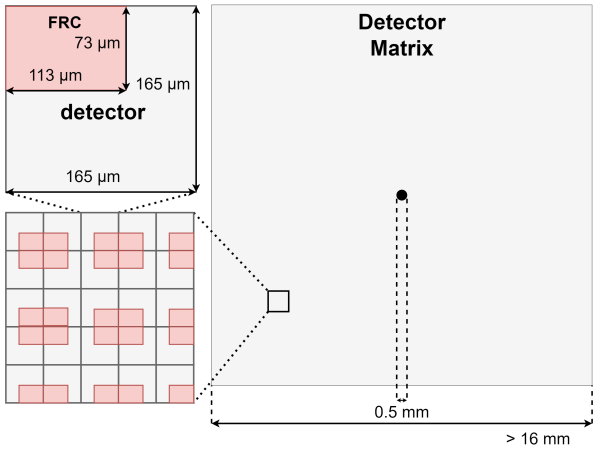


Figure 2.2: Illustration of the proposed SEM sensor using a detector matrix consisting of individual detectors located on the back-side of the silicon, with a front-end readout circuit (red) on the front-side of the silicon.

2.1.2. System requirements

Within the list of requirements the 'system' refers to the system which is designed within this thesis, while the 'ASIC' refers to requirements that are related to the final ASIC:

Requirement 1 The system must count the total number of events within the matrix for every 2.5 ns sample period (400 MHz).

Requirement 2 The system must perform the readout continuously without stalling.

Requirement 3 The root mean square (RMS) voltage of the noise introduced on the FRCs Analog Reference voltages by the system, measured at the FRC input, must be less than 10 mV.

Requirement 4 The error rate introduced by the system must be less than 1000 ppm.

Requirement 5 The ASIC digital supply must draw less than 5 W.

Requirement 6 The system must be implemented using the TSMC 40 nm PDK.

Requirement 7 The ASIC must fit within an area of 25 mm × 25 mm.

Requirement 8 The system must implement a detector matrix must at covering an area with a diameter of at least 16 mm.

Requirement 9 The system must have a gap in the center of the detector matrix for the primary electron beam with a diameter of 0.5 mm.

Requirement 10 The clock skew within the system must be less than 100 ps.

Requirement 11 The ASIC must provide storage for the results.

Requirement 12 The ASIC must provide an IO interface for external hardware.

Requirement 13 The ASIC must be provide configuration registers for the host system.

2.1.3. High-Level System Architecture

Based on the requirements in section 2.1.2 the high-level system architecture shown in figure 2.3 is proposed with the following subsystems:

- **Host System**: This is the system responsible for controlling the SEM, including the PE beam position.This sub-system falls beyond the scope of this work.
- **External PLL**: The external Phase Locked Loop provides the 400 MHz clock signal to both the host system, and the ASIC. This sub-system falls beyond the scope of this work.

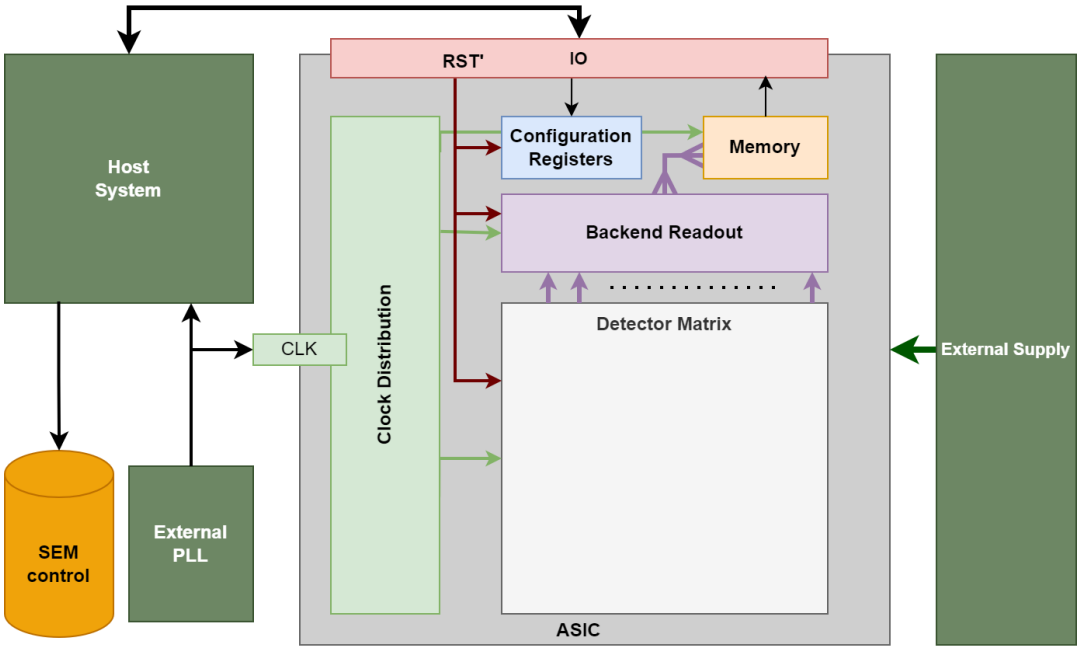


Figure 2.3: High-Level system architecture of the ASIC showing the different sub-systems and their connections

- **External Supplies**: Analog reference voltages, analog supplies, and digital supplies are expected to be supplied to the ASIC externally. This sub-system falls beyond the scope of this work.
- **Detector Matrix**: The matrix of detectors as described above. This subsystem must ensure that *Requirement 8* is met.
- **Backend Readout**: This subsystem reads the output of every detector in the sensor matrix and outputs a binary integer representing the total number of events. This subsystem will be the main focus of this thesis, as the other subsystems can all be based on existing implementations. This subsystem must ensure that *Requirement 1*, *Requirement 2* and *Requirement 4* are met.
- **Memory**: Serves as a buffer for transmitting the data to the host system. This subsystem must ensure that *Requirement 11* is met.
- **Configuration Registers**: Allows the host system to configure environment parameters such as the width and height of the SEM image (in number of samples). The configuration register subsystem must ensure that *Requirement 13* is met.
- **IO**: Provides the communication interface for data to and configuration from the host system and allows receiving control signals from the host system (such as a reset signal). This subsystem must ensure that *Requirement 12* is met.
- **Clock distribution**: The clock signal is provided by an external PLL. The clock distribution must ensure that *Requirement 10* and *Requirement 1* are met.

Although the ASIC contains many subsystems, this thesis will primarily focus on those closely related to the event-counting-based SEM sensor. That is, the detector matrix, the backend readout, and the clock distribution. The remaining subsystem are a crucial part of the ASIC, however, their design and implementation provides no novelty, and can be borrowed from many other systems. Therefore, their design is not covered within this thesis.

2.2. Prior Art

Most of the literature on state of the art imaging sensor readout architectures focuses on area scan sensors containing pixels with a multi-level analog output that requires conversion to a digital value

and is stored per individual pixel [15] [16] [17] [18].
In the following sections, readout architectures for conventional CMOS image sensors, X-Ray sensors, SEM sensors and single photon imaging sensors will be introduced, and their relevance to our application will be evaluated.

2.2.1. CMOS image sensor readout architectures

In general, CMOS image sensors are used for area scan imaging in which an entire image is captured in a single frame. The area is subdivided into pixels, and each pixel has a photodetector that generates a current depending on the amount of incident light. The current is integrated during the sampling period and converted to a voltage. The voltage is converted to a digital signal which is used to form the digital image. [19] State of the art CMOS image sensors readout circuits can be categorized into two groups, namely, digital and analog [15]. The main architectures of the digital category are the sequential readout, the per column analog-to-digital (ADC) converter, and the per pixel ADC [15] [19]. In-pixel analog memory is the most notable advancement in analog read-out architectures [20] [21].

None of these techniques are suitable for our application. The digital readout circuits are mainly focused on the conversion of the pixel voltage to a digital value, and the parallelization of this. The analog memory architectures, while interesting, are also not relevant, as the detector outputs in our application are already digitized. Therefore, D-flip-flops can be used for storage. Furthermore, storing detector values for later processing would require the SEM to stop scanning while processing the stored results, which is in direct conflict with *Requirement 2*.

2.2.2. X-ray sensor readout architectures

X-ray detectors have two main variants. Firstly, there are indirect detectors; here the X-ray photons are converted into visible light, which is measured using conventional photo detectors. Secondly, there are direct detectors, where the X-ray photons are directly converted to an electrical charge. This charge is then either integrated or processed based on the amount of charge generated, which is known as a pulse processing system. This signal processing method is similar to that of the detector in our application. However, in X-ray detectors the detection grid is still subdivided into pixels, and the individual pixel values are of interest. Therefore, many techniques discussed in 2.2.1 are also applicable to the readout ASIC of X-ray detectors.

Kleczek et al. [22] propose the use of a counter that keeps track of the number of pulses seen during the sample period and outputs a single integer value. Erdinger [23] illustrated the use of in-pixel static random access memory (SRAM) to store intermediate samples. Using this method, a 4096 pixel X-ray detector was designed which was able to achieve a burst frame rate of 4.5 Mfps. This is similar to the concept proposed by Sugiyama et al. [21] and used by Kleinfelder et al. [20] and El-Desouki et al. [15] but then within the digital domain.

Although the front-end readout circuits are similar to those described in [9], the back-end readout systems are not compatible with the requirements of our system. Our system has individual detectors that output a binary value and we are interested in the sum of all the detectors. Although there is overlap in the concepts of photon-counting and electron-counting between our application and the X-ray sensor, the X-ray sensor is an area scan device, while our application is a pixel scan device.

2.2.3. SEM sensor readout architectures

The most common way to detect electrons in SEMs used to be the Everhart-Thornley detector [24]. A scintillator is used, which is a material that can convert ionizing particles to photons. A very thin metal coating is used on the scintillator, to which a large voltage (in the order of 10 kV) is applied. A Faraday cage is placed around the scintillator to prevent any deflection of the electron beam, as is illustrated in figure 2.4. The low-energy electrons from the specimen are accelerated toward the scintillator because of the electric field on the metal coating. The photons generated from the electrons by the scintillator are transmitted to a photomultiplier, which converts photons into current pulses. These current pulses are then amplified and processed for every sample period. This results in an analog signal that is related to the amount of incident electrons during a sample period.

A variation of the original Everhart-Thornley detector is Through-the-Lens (TTL) detection. This type of

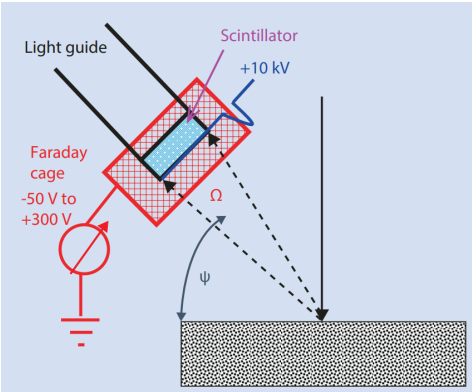


Figure 2.4: Schematic of an Everhart-Thornley detector showing the primary electron beam incidence on the imaging surface. A voltage of 10 kV is applied to a metal coating on the surface of the scintillator, creating an electric field that attracts low energy electrons. [1]

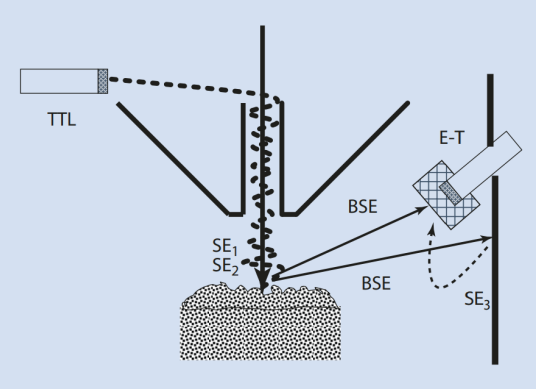


Figure 2.5: Schematic of Through-the-Lens (TTL) Everhart-Thornley detector where a snorkel lens traps the SEs, and causes them to accelerate in the opposite position of the PE beam. The Everhart-Thornley detector is therefore placed above the objective lens to detect the SEs. [1]

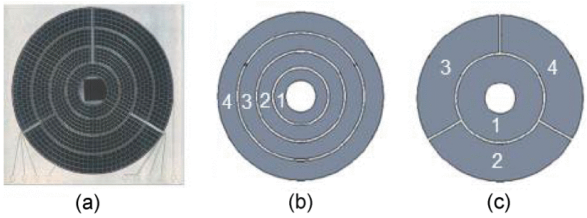


Figure 2.6: 8-segment detector proposed by Sakic et al. [25] (a) and its concentric back-scattered (b) and angular back-scattered (c) configurations

detection is suitable for SEM's that use a snorkel objective lens for the electron beam. The magnetic field of such a lens reaches the specimen and is able to trap electrons, causing them to accelerate up the PE beam column. An Everhart-Thornley detector is then placed above the objective lens to detect these electrons as illustrated in figure 2.5. Signal processing follows the same steps as for the conventional Everhart-Thornley detector.

Through technological advancements, SEM sensors can also be implemented on silicon. The detector is placed under the objective lens and therefore requires a gap in the center for the electron beam, commonly referred to as the through-wafer aperture. The detector is generally divided into different sections, so that each section can be measured independently [1]. This allows the specimen to be analyzed under different illumination angles relative to the electron beam. Furthermore, the detector area is located on the back of the silicon wafer, allowing direct implementation of signal processing, and readout circuits on the front of the silicon wafer.
Sakic et al. [25] propose the fully configurable detector of eight segments shown in Figure 2.6. The detector segments can be read individually or can be configured to form a larger detector. This results in images of the specimen with different illumination angles, contrast and resolution as illustrated in figure 2.7 [26]. The backend readout architecture takes the analog outputs of each segment and converts them to digital values. These digital values can then be combined in post-processing to obtain the total signal for different configurations.

Both the Everhart-Thornley and semiconductor readout architectures are not applicable to our system. Both of these architectures have detectors that output an analog signal, while our system only has digital outputs from the detectors.

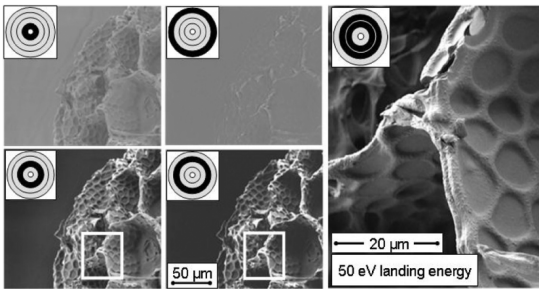


Figure 2.7: Image of uncoated pollen sample using different configurations of the detector proposed by Sakic et al. [25]

2.2.4. Single photon imaging

Single photon imaging sensors are widely used within light detection and ranging (LiDAR) systems. Here, a short laser pulse is generated. The photons from this laser are reflected by an object, and detected using a single photon avalanche diode (SPAD). With a time to digital converter (TDC) the time of flight is determined, which is then used to determine the distance of the object that reflected the light [27] [28] [29]. These LiDAR imaging systems have some similarities to the system proposed in 2.1. SPADs generate a binary output whenever a single photon is detected, similar to the electron detectors. Furthermore, the LiDAR imaging systems are implemented as an array of SPADs, similar to how our system is implemented as an array of electron detectors. However, LiDAR imaging systems are area scan devices, where each SPAD represents a specific depth pixel in the final image.
An interesting approach used by LiDAR imagers is that they are designed so that multiple SPADs can share a single TDC. The reference time for each of the SPADs is the same, namely when the laser pulse is emitted. Therefore, the TDCs are designed in such a way that they can convert an arbitrary number of input pulses to a digital time stamp. [30]

2.3. Conceptual readout architectures

This section presents several architecture concepts for the back-end readout subsystem, using analog and digital readout techniques. The architectures presented are compared and evaluated based on the requirements provided in Section 2.1, identifying the Pipelined Adder Tree (AT), Critical AT, and Pulse Counter as the most promising. These architectures will undergo further design and simulation to validate their suitability.

2.3.1. Current Controlled Ring Oscillator

A ring oscillator is formed by connecting an uneven number of inverters to each other in series. The output of such a ring oscillator will oscillate at a certain frequency that can be adjusted using tuning capacitors, as shown in Figure 2.8. Wall et al. [31] presents a ring oscillator of which the output frequency depends on the input current. Such a current-controlled ring oscillator (CCRO) could be used to add a number of binary signals together. Each high signal would source or sink some bias current to or from the CCRO input. As a result, the output frequency of the CCRO would then be directly related to the number of high inputs.

However, accurately measuring the output frequency is not a trivial task. Furthermore, the CCRO output frequency must be larger than the system clock frequency in order to provide a reliable result each clock period. This results in a large dynamic power consumption and also increases electromagnetic interference (EMI) due to the high frequencies required, which could negatively impact the analog front-end readout circuits.

2.3.2. Current addition

Cini and Morgül [32] propose a current-mode adder circuit that takes a 7-bit input and has a 3-bit output. The digital inputs are first converted to a current using an NMOS transistor as a current source. Currents from all high inputs (I_{sum}) are fed through a resistor R, which is implemented using a PMOS transistor with its bulk and drain connected together. The design is differential, so the currents from all low inputs (I'_{sum}) are fed through a different resistor R'. The supply voltage is 1.8 V; therefore, whenever the voltage across the resistors exceeds 0.6 V, the PMOS acts as a diode, which exhibits as clipping

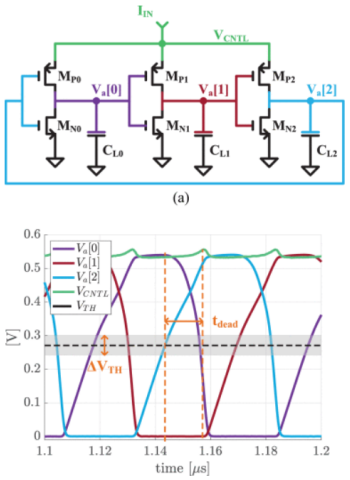


Figure 2.8: (a) Ring Oscillator with tuning capacitors (b) Output waveforms of the Ring Oscillator [31]

of the voltage at the PMOS drain around 1.2 V.

The outputs are determined by comparing the voltages V_i and V_i' at the negative terminals of resistors R and R', respectively, to predetermined levels. If $V_i > V_i'$ the comparators check against V_i. However, if $V_i < V_i'$, it indicates that more than 3 inputs are high. Due to the clipping that occurs, the precise sum of the inputs cannot be obtained from V_i, instead the comparators check against V_i' to determine how many inputs are low. Finally, the comparator outputs are converted to digital values.

A similar circuit could be used to add binary signals with a low probability of being high. To clarify, when given a thousand binary signals with the probability of more than three of these being high at the same time being close to zero, these signals can be added using a current adder. The current adder would be designed to add up to three currents together as illustrated in figure 2.9. Such a circuit would only require a 2-bit output. In contrast to the circuit proposed by Cini and Morgül, this circuit does not use low inputs to generate a complementary signal, as this would increase the static power consumption and possibly conflict with *Requirement 5*. Instead, only the enabled inputs are summed, and the voltage at V_o is converted to its digital value using an ADC.

For such a design, the current sources should be located close together to minimize the parasitic capacitance of the shared bus at the negative terminal of the resistor. Within the context of the detector matrix the outputs of the detectors would be routed to some central point where the current addition circuit is located. Depending on the distance this could require buffering of the detector outputs, possibly even using clocked memory elements like d-flipflops (DFF).

Unfortunately, the TSMC40 nm process is intended to use a supply voltage no higher than 1.2 V. Therefore, the dynamic range of the voltage at V_o is limited. Implementing the resistor using a PMOS would further reduce the dynamic range. Consequently, there is an upper limit to the number of high inputs that can be summed that is related to the bias current and the noise margin of the ADC.

Moreover, the bias currents must be significantly large to negate the parasitic capacitance of V_o. The operation of this circuit relies on a stable bias current for each input, so each input would require its own dedicated current source. Enabling and disabling this current source within the 2.5 ns required by *Requirement 1* could cause problems with settling of the currents. Finally, the ADC needs some conversion time, during which the voltage V_o would have to remain stable. This could be difficult to achieve with the required sample rate of 400 MHz. One advantage of this implementation is that it does not require many digital circuits, resulting in less EMI for the analog reference voltages.

2.3.3. Current integration

The current integration architecture aims to solve the stability concern of the ADC input voltage in the current addition architecture. The binary inputs are first stored using DFFs or latches. During the next clock phase (or period if a DFF is used), the stored signals are used to sink a small current from a large capacitor, causing the voltage to slowly decrease. At the end of the clock period, the remaining voltage over the capacitor is converted using an ADC. The voltage across the capacitor at the end of the clock

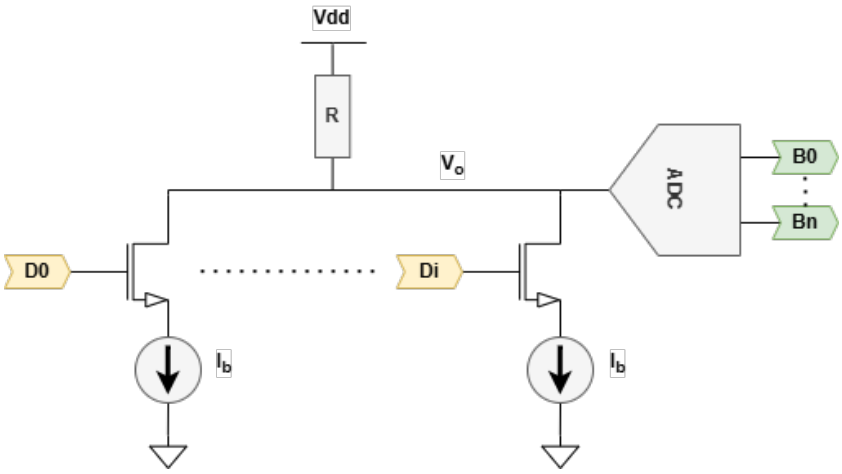


Figure 2.9: Current Addition architecture concept. Detector outputs D_n with an event enable a bias current through R. The voltage drop caused by the sum of currents through R is associated to the number of enabled detectors, and is converted to a binary representation of this using an ADC.

period is directly related to the number of events. Depending on the upper limit to the event probability of the cluster, the ADC only requires a few bits at the output. The system operates over three clock cycles:

1. **Pre-charge**: The capacitor is pre-charged to VDD.
2. **Current integration**: The capacitor is discharged by I_s.
3. **Conversion**: The voltage across the capacitor is converted to a digital value.

The circuit for this concept is shown in figure 2.10. Small 3-bit flash ADCs are able to operate at frequencies well above 400 MHz [33] , and will therefore not be a limitation for this concept. As an alternative to current sources, the sink current can also be generated by using a minimum-size NMOS.

The main advantage of this system is that power consumption can be kept relatively low compared to the current addition concept. Furthermore, this system is much less susceptible to EMI noise because the current is integrated over 2.5 ns. Unfortunately, such a system would be very sensitive to PVT variations and would require careful design to account for these variations. In addition, the ADC requires at least one (or more, depending on the ADC architecture) reference voltage. Finally, as the voltage across the capacitor decreases, the current through the NMOS also decreases, resulting in a nonlinear relationship between the number of detectors with an event and the final voltage across the capacitor. To avoid this the NMOS transistors sinking current would have to be kept in saturation by keeping the drain voltage sufficiently high. This reduces the dynamic range in which the ADCs would have to operate, leading to a reduced SNR.

2.3.4. Adder tree

A full adder (FA) is a digital circuit that takes three binary inputs (with the same weight) and combines them into a 2-bit output, the sum (S), with the same weight as the inputs, and the carry (C), with twice the weight of the inputs [34]. It is the key building block of arithmetic units. Connecting multiple FA's in parallel, with the carry outputs connected to the third input of its neighbor, allows for the addition of two larger digital numbers. Such a configuration is also known as a ripple-carry adder since the carry has to propagate from the least significant adder through all the intermediate adders to the final adder [35]. However, our application can be described as the addition of many single bits with the same weight. A carry-save adder (CSA) is better suited for such an input [36]. A CSA performs additions for bits of the same weight in parallel and saves the carry output for the next addition.

A FA can also be viewed as a three-to-two compressor circuit. It takes three binary inputs of the same weight and compresses them into two binary outputs with different weights [37]. In a similar fashion a half adder (HA) can be described as a circuit that takes two same weight binary inputs and converts these into two different weight outputs. Using these concepts, a multi-operand addition can

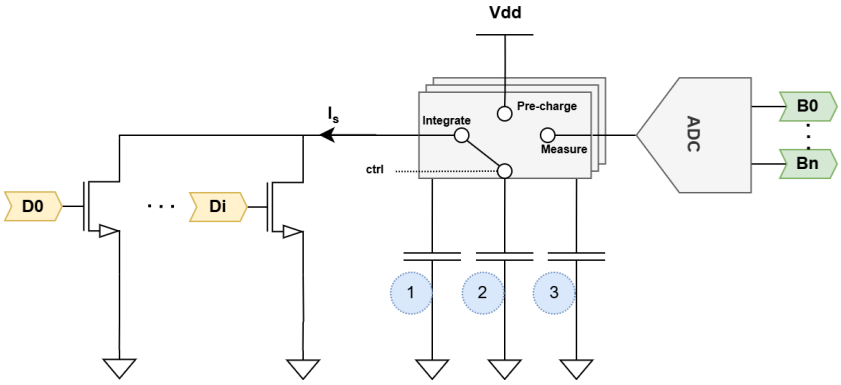


Figure 2.10: Current integration architecture concept. A capacitor is pre-charged to Vdd. The detector outputs D_i with an event enable the minimum size NMOS to sink current from the pre-charged capacitor over an entire or half clock cycle. The capacitor is connected to the ADC, which converts the voltage into a digital representation of the number of detectors with an event.

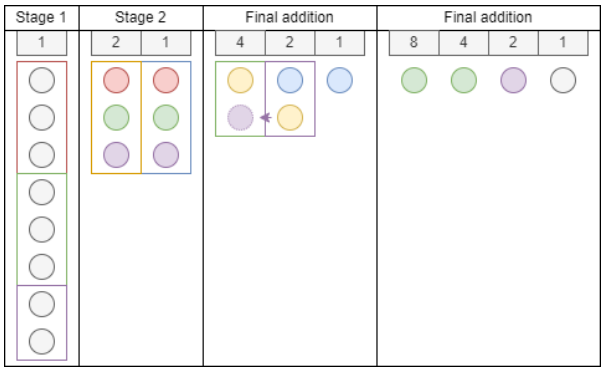


Figure 2.11: CSA for an input of eight equally weighted bits. Each dot represents a bit with the weight indicated by the column. Bits are added by Full-Adders (bounding box around three bits) and half adders (bounding box around two bits)

be split up into different stages. This is illustrated for the addition of eight single bits with the same weight in figure 2.11. Each dot represents a single bit. In the first stage two FA's and a HA can be used, indicated by a bounding box around three and two bits, respectively. In the next stage the sum and carry outputs are color-coded accordingly. In stage 2 two FA's can be used. The final addition is done by using two HA's to create what is known as a ripple carry adder to produce the final result. The resulting circuit is known as a Wallace tree [38].

The main advantage of a Wallace tree adder is that after each stage the results can be stored using Latches or DFF's, and a pipelined structure can be created [39]. Although this significantly increases the delay from the input to the output, it reduces the critical path delay to the delay of a single full adder. Depending on the clock period, the location of the DFF's can be altered to be every couple of stages, as long as the critical path delay between DFF's remains smaller than a single clock period.

An adder tree can be used to add up many inputs to a single binary integer output, which is exactly what the backend readout subsystem must implement. However, the critical path delay through an entire adder tree quickly becomes larger than the 2.5 ns clock period. In order to meet requirements *Requirement 1* and *Requirement 2* the adder tree must be pipelined using either DFF's or latches. This allows the total addition time to extend past the current sample period, while still being able to process the data from the following sample period. Depending on the critical path delay of a single stage of the adder tree, the addition of the entire sensor matrix can be performed over a number of clock cycles. Figure 2.12 illustrates the pipelined adder tree using rising edge triggered DFF's.

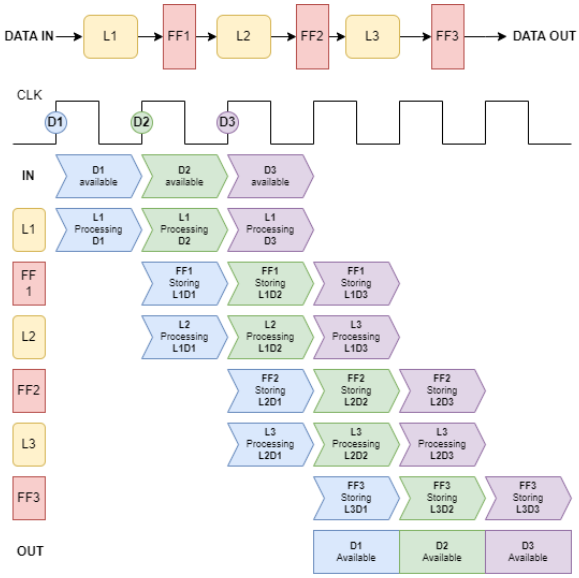


Figure 2.12: Illustration of data propagation through a pipelined Tree adder.

2.3.5. Critical-redundancy Adder Tree

The adder tree described in section 2.3.4 assumes that it is possible for all inputs to be high. However, as stated in section 2.1 the upper limit to events per clock cycle across the entire detector array is 50. If an adder tree is designed for 10.000 inputs, then the output width would be 14 bits. To store the maximum number of 50 events as given in Section 2.1, only 6-bits are required. Therefore, the adder tree contains redundant logic, wasting power (through leakage currents) and area.

Instead of implementing all this redundant logic, we may observe that given the upper limit of 50 events, there will only ever be one level within the logic tree which outputs a high signal on the 6th bit. Consequently, OR gates can be used instead of FA's and HA's. Furthermore, any higher-weight bit is simply excluded from the adder tree. The resulting adder tree has a critical redundancy for the given application. This concept can be extended further to each level of the adder tree if the probabilities of the detectors within the sensor matrix are known.

2.3.6. Pulse counter

For all previously proposed architectures, it is assumed that the detector output is synchronized with the global clock. To be specific, if an event occurs at a given detector, the output of the detector becomes a logical '1' at the next rising edge of the clock. However, the system could also be designed asynchronously. Namely, as soon as an event occurs at a given detector, its output becomes a logical '1'. Both these cases are illustrated in figure 2.13.

With the use of asynchronous detectors the outputs are routed to a central location that the propagation delay is matched. At the centralized location, the rising edges of the detector output can be converted to pulses. Similarly to X-ray pixel readout architectures, these pulses can then be counted [22]. These pulses could be used to trigger a binary counter, however, a binary counter would require some degree of combination logic. This would limit the interval at which the pulses could be processed. Therefore, to maximize the speed at which pulses are processed, a shift register is used instead, where the input of the first register is connected to the VDD. This results is a logical '1' that propagates through the shift register with each pulse. The shift register output then represents the number of events as a thermometer code. Using an encoder as proposed by Lavania et al.[40] this thermometer code can be converted to a binary integer. This system is shown in figure 2.13.

The width of the pulse generated from the rising edge of a detector output determines the range that can be detected. If the output pulse width is for instance 500 ps, then no more than 4 events can be detected within 2.5 ns. Furthermore, if the events occur within this 500 ps window of one another, they will not trigger the shift register. Therefore, to maximize the range of detectable events, the output pulse

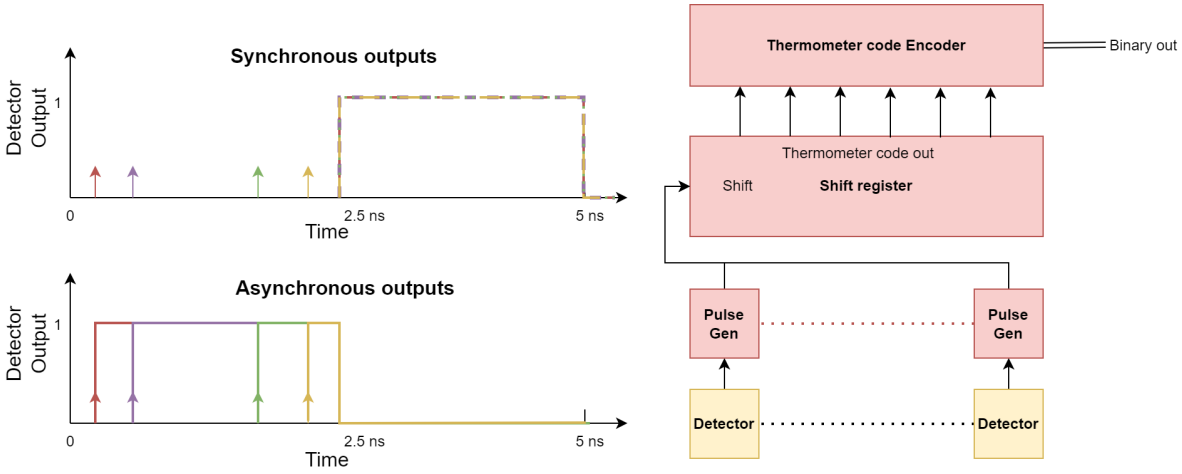


Figure 2.13: Back-end readout architecture concept where the detectors have asynchronous outputs instead of clocked (left). The asynchronous outputs are routed to a central location within a cluster of detectors, where the rising edges are converted to short pulses. These pulses are used to activate a shift register, moving a bit one position for each pulse. The shift register output represents the number of events as a thermometer code, which is converted to a binary integer using an encoder.

Readout Architecture	**Requirement**						**Design Effort**
	1: Period	2: Readout	3: EMI	4: Error	5: Power	6: PDK	
CCRO	?	✓	✗	?	✗	✓	+++
Current Addition	?	✓	?	?	✗	✓	+
Current Integration	?	✓	✓	?	?	✓	++
Pipelined AT	✓	✓	?	?	✓	✓	-
Critical AT	✓	✓	?	?	✓	✓	- -
Pulse Counter	?	✓	?	?	✓	✓	- -

Table 2.1: Comparison of the different architectures presented based on the requirements presented in section 2.1, and the estimated design effort required.

width must be minimized. The size of the shift register is also directly related to the maximum number of expected events of the connected detectors. Due to these limitations, this circuit is only suitable for use within clusters that have a very low event probability.

2.4. Evaluation of readout Architectures

In the previous sections, many different readout architectures have been proposed. These will now be compared against each other using the requirement formulated in Section 2.1. Not all of these requirements are related to the back-end readout subsystem, only *Requirement 1 - Requirement 6* are relevant. In Table 2.1 each back-end readout architecture discussed in this chapter is related to the relevant requirements, and the required design effort is estimated. At this point, there is no method to reliably quantify the error introduced by these architectures. As a result, the entries for *Requirement 4* are left as unknown and will be evaluated in Chapter 3.1. From all the architectures, the Current integration, Pipelined AT, Critical AT and Pulse Counter seem to be best suited for the back-end readout subsystem. However, the current integration architecture is expected to have a much larger design effort compared to the other three. Therefore, the Pipelined AT, Critical AT and pulse counter architectures are selected to be design and evaluated in the following Chapter.

3

Sparse Matrix readout Architectures

At the end of Chapter 2 three architectures were identified as promising candidates for the backend readout of the detector matrix. That is, the pipelined adder tree, the critical redundancy adder tree, and the pulse counter. The first part of this chapter presents equations that are used to model a sparse Bernoulli matrix. In the following sections, the detailed design of these three architectures is presented and equations are derived to estimate the error probability each architecture introduces. Finally, the designs are implemented and evaluated for a detector matrix of 16 × 16 detectors. These results will be used in chapter 4 for the design of the backend readout subsystem.

3.1. Bernoulli matrix model

In order to gain more insight into possible architectures, we will take a closer look at the properties of a Bernoulli matrix. This allows us to model any Bernoulli matrix and determine the error probabilities that different architectures might introduce. First, we will take a closer look at the behavior of a single Bernoulli source, which will then be expanded to a Bernoulli matrix. Finally, we will look at the definition of a sparse matrix.

In the context of this work, a Bernoulli source is any kind of sensor element whose output X is either a logical one or zero during a specific sampling period n. For every sampling period, the probability that Bernoulli source outputs a logical one follows a Bernoulli trial [14] with some probability p:

$$X[n] \sim Bernoulli(p) \rightarrow P(X[n] = 1) = p \tag{3.1}$$

and a probability of $1 - p$ of outputting a logical zero:

$$P(X[n] = 0) = 1 - p \tag{3.2}$$

Given a Bernoulli matrix B containing M independent Bernoulli sources where each individual source has its own probability p_m

$$B = (B_m)^M, \quad B_m \sim Bernoulli(p_m), \quad m \in \{1, \dots, M\} \tag{3.3}$$

The probability that k of these M sources produce a logical one within the sampling period n can be determined using the Poisson Binomial distribution [41]:

$$P(B[n] = k) = \sum_{A \in F_k} \prod_{i \in A} p_i \prod_{j \in A^c} (1 - p_j) \tag{3.4}$$

where F_k is the set that contains all subsets with k integers from A, and A^c is the complement of subset A. The mean and variance of the Poisson binomial distribution are defined as follows:

$$\mu_{pb} = \sum_{i=1} p_i \tag{3.5}$$

$$\sigma_{pb}^2 = \sum_{i=1} (1 - p_i)p_i \tag{3.6}$$

Calculating the probability using equation 3.4 directly is only possible for small values of k and M as the size of F_k increases factorially with the size of M. Biscarri et al. [42] propose a different method for calculating this probability using a Fourier transform. This method is implemented using the Matlab script in Appendix G.2.7.

Another alternative approach uses the Poisson distribution to estimate the Poisson binomial distribution [43]. For a single Bernoulli source, every sample period occurs at a fixed interval of T with a probability p of an event occurring during that entire sampling period. Rather than expressing the probability of an event occurring during a single sampling period, we can express the expected event rate of the source, and model its output as a Poisson distribution Y. Given the Bernoulli probability p the event rate per interval λ can be expressed as:

$$\lambda = p \tag{3.7}$$

With the sampling period T, the expected event rate per second R can be expressed as

$$R = \frac{p}{T} \ \ s^{-1} \tag{3.8}$$

Following this, the Poisson distribution for a single Bernoulli source can be used to determine the probability of k events occurring within the time interval t:

$$Y \sim Pois\left(R = \frac{p}{T}\right) \rightarrow P(Y[t] = k) = \frac{(Rt)^k e^{-Rt}}{k!} \tag{3.9}$$

Furthermore, given a matrix C of M independent Poisson distributions,

$$C = (C_m)^M, \quad C_m \sim Pois(R_m), \quad m \in \{1, \dots, M\} \tag{3.10}$$

then the sum of these distributions is

$$D = \sum_{i=1}^{M} C_i \tag{3.11}$$

which is also a Poisson distribution with an expected event rate equal to the sum of event rates.

$$D \sim Pois\left(\sum_{i=1}^{M} R_i\right) \tag{3.12}$$

To estimate the probability in equation 3.4 we define the following Poisson distribution:

$$E \sim Pois\left(R = \frac{1}{T}\sum_{i=1}^{M} p_i\right) \rightarrow P(E[t] = k) = \frac{(Rt)^k e^{-Rt}}{k!} \tag{3.13}$$

where p_i are the probabilities from B. A good estimator should be unbiased, therefore, the mean of the estimator should equal the mean of the estimated data. For the Poisson distribution, the mean is given by:

$$\mu_{pois} = Rt = \frac{t}{T}\sum_{i=1}^{M} p_i \tag{3.14}$$

For the time interval $t = T$ the mean of the Poisson distribution E reduces to 3.5

$$\mu_{pois} = \sum_{i=1}^{M} p_i = \mu_{pb} \tag{3.15}$$

proving that E is an unbiased estimator of B. To be a good estimator, the variance must also be similar. The variance of the Poisson distribution is given by:

$$\sigma_{pois}^2 = Rt = \frac{t}{T}\sum_{i=1}^{M} p_i \tag{3.16}$$

A matrix with M elements can be characterized by its sparsity S as:

$$S = 1 - \frac{\sum p_i}{M} \tag{3.17}$$

For $S > 0.95$ a matrix is said to be sparse. This work focuses on the readout of a sparse matrix. Therefore, it is safe to assume that

$$p_i \ll 1 - p_i \tag{3.18}$$

and consequently

$$(1 - p_i)p_i \approx p_i \tag{3.19}$$

Therefore, the variance of the Poisson binomial distribution in equation 3.6 and the variance of the Poisson distributions in equation 3.16 are almost equal as long as equation 3.18 is valid.

$$\sigma_{pois}^2 = \sum p_i \approx \sum (1 - p_i)p_i = \sigma_{pb}^2 \tag{3.20}$$

To conclude, the Poisson distribution E is a good estimator for the Poisson Binomial distribution B for sparse matrices.

With the Poisson distribution E the cumulative distribution function (CDF) [14] in equation 3.21 can be used to determine the probability of more than k events occurring within a given time period t. This will be used throughout this chapter to determine the error probabilities of the different backend readout architectures.

$$P(E[t] \geq k) = 1 - e^{-Rt}\sum_{i=0}^{k-1} \frac{(Rt)^i}{i!} \tag{3.21}$$

3.2. Full Adder Tree

The full adder tree can handle as many inputs as required. The adder tree is implemented using an algorithm. The signals within the adder tree have an associated weight i, such that the decimal value of the signal corresponds to 2^i. Furthermore, the signals are separated for each layer of the adder tree. Therefore, the number of inputs for each weight in the first layer of the adder tree is indicated by $b_{i,1}$ (i.e., $b_{i,1} = [16, 6, 1]$ would indicate 16 inputs with a weight of 1, 6 inputs with a weight of 2, and 1 input with a weight of 3). The number of outputs for each weight in the first layer (which are also the inputs to the second layer) is indicated by $b_{i,2}$.

Given the inputs to the adder tree $b_{i,1}$, the width of the output of the adder tree can be determined. The largest possible sum that can be produced by the input signals is given by:

$$s = \sum_{i=0} b_{i,1} 2^i \tag{3.22}$$

The width of the output such that s can be represented is then determined with

$$outputs = \lceil \log_2(s + 1) \rceil \tag{3.23}$$

The algorithm determines the number of FAs and HAs required for each layer and determines the number of signals at the output of that layer until each weight only has a single signal. Similarly to the number of signals within layer j, $fa_{i,j}$ and $ha_{i,j}$ indicate how many FAs and HAs are required for a specific weight i within the layer.

$b_{i,j}$, $fa_{i,j}$ and $ha_{i,j}$ are determined iteratively. The algorithm first fully compresses the lowest weight signal to a single output, then continuous to compress the next weight signals to a single output. This continues until the signals with the weight determined in equation 3.23 are compressed to a single output. For each iteration, the following steps are followed:

1. Check if a HA is required. This is the case if the current layer m contains only two signals of the current weight n and one signal of each lower weight:
   if $(\sum_i = 0^n b_{i,m} == n+1$ AND $b_{n,m} == 2) \quad \rightarrow \quad ha_{n,m} = 1$
2. Determine the number of FAs required. This is done by taking the integer floor of the number of signals at the input of the current layer m of with the current weight n divided by 3 (as explained in section 2.3.4 a FA can be seen as a three-to-two compressor):
   $fa_{n,m} = \left\lfloor \frac{b_{n,m}}{3} \right\rfloor$
3. Determine the number of outputs of the current layer m (the outputs are stored as inputs to the next layer $m+1$) with the same weight n. Some signals at the input of the current layer cannot be compressed and thus will carry through to the next layer without passing through an FA or HA. A single FA will reduce the number of signals of the same weight by two, while an HA reduces the number of signals of the same weight by one:
   $\Delta b = -2fa - ha.$
   Furthermore, the FA and HA carry signals assigned in the next step should also be taken into account:
   $b_{n,m+1} = b_{n,m+1} + b_{n,m} - 2fa_{n,m} - ha_{n,m}$
4. Determine the number of outputs of the current layer m (the outputs are stored as inputs to the next layer $m+1$) with the next weight $n+1$. These are the carry signals generated by the FAs and HAs. Both have only one carry signal:
   $b_{n+1,m+1} = b_{n+1,m} + 2fa_{n,m} + ha_{n,m}$

3.2.1. Full Adder Tree Implementation

The algorithm to generate the full adder tree is implemented in the SKILL script in appendix H.1. This script first determines the structure of the full adder tree and then generates the Virtuoso schematic. A simplified layout is created and extracted for the basic components (FA and HA) where the input and output net wires have a length equal to four standard cell widths to account for the parasitics associated with the interconnect. However, this remains an approximation, so the final power consumption can deviate from the results obtained using these models.
Using the SKILL script, an full adder tree is generated for 256 inputs with a weight of 0. Figure 3.1 shows the structure of the generated full adder tree.

3.2.2. Full Adder Tree error probability

The full adder tree architecture does not have an associated error source in its design. No matter the number of events at its input, its output will accurately represent them. The only source of errors present in this architecture arises from race conditions, which can be fully mitigated through proper design. Therefore:

$$\epsilon_{at} = 0 \tag{3.24}$$

Technically, an error also occurs whenever a single detector has more than one event within a single sampling period. However, such errors will not be taken into account.

3.3. Critical Adder Tree

For a sparse matrix, an upper event limit a_m can be defined such that the following holds:

$$P(X > a_m) \approx 0 \tag{3.25}$$

Using the upper event limit, the total number of bits required for the readout of the matrix can be bounded as follows:

$$outputs = \lceil \log_2(a_m) \rceil \tag{3.26}$$

Weight	Cell	IN	L1		L2		L3		L4		L5		L6		L7		L8		L9		L10		L11		L12		L13		L14	
			Gates	Output:	Gates	Output:	Gates	Output:	Gates	Output:	Gates	Output:	Gates	Output:	Gates	Output:	Gates	Output:	Gates	Output:	Gates	Output:	Gates	Output:	Gates	Output:	Gates	Output:	Gates	Output:
b0	FA	256	85	86	28	30	10	10	3	4	1	2		1		1		1		1		1		1		1		1		1
	HA												1																	
b1	FA			85	28	57	19	29	9	14	4	7	2	4	1	2		1		1		1		1		1		1		1
	HA																1													
b2	FA					28	9	29	9	20	6	12	4	6	2	3	1	2		1		1		1		1		1		1
	HA																		1											
b3	FA							9	3	12	4	10	3	8	2	6	2	3	1	2		1		1		1		1		1
	HA																				1									
b4	FA									3	1	5	1	6	2	4	1	4	1	3	1	2		1		1		1		1
	HA																						1							
b5	FA											1		2		4	1	3	1	2		3	1	2		1		1		1
	HA																								1					
b6	FA																	1		2		2		3	1	2		1		1
	HA																										1			
b7	FA																									1		2		1
	HA																												1	
b8	FA																													1
	HA																													
FA's		243	85		56		38		24		16		10		7		4		2		1		0		0		0		0	
HA's		5	0		0		0		0		0		1		0		1		1		1		1		0		0		0	

Figure 3.1: Adder tree generated by the SKILL script in appendix H.1. The columns correspond to the layer of the full adder tree, while the rows correspond to the weights of the signals. For each layer and weight the number of FAs, HAs and output signals are shown.

In the previous section, with the design of the full adder tree, this was not considered. However, this allows for a significant improvement in the readout architecture. Instead of performing all additions using FAs and HAs, only signals with a weight of $b_0 - b_{outputs-2}$ are added in this way. The signals with weight $b_{outputs-1}$ can be added using an OR gate, as there will never be more than one high input of that weight. For this critical adder tree, four-input OR gates will be used instead of two-input OR gates, as this reduces the number of levels required in the critical adder tree.

Similarly to the full adder tree in section 3.2, the critical adder tree (CRAT) is implemented using an algorithm. The algorithm determines the number of FAs, HAs, and OR-gates required for each layer, and determines the number of signals at the output of that layer until each weight only has a single signal. $b_{i,j}$ indicates how may signals of weight i are present at the input of layer j. Similarly, $fa_{i,j}$, $ha_{i,j}$ and $or4_j$ indicate how many HAs, FAs and OR-gates are present in layer j with weight i. Their values are determined iteratively by first compressing the signals with the lowest weight down to a single output, and then doing the same for the remaining weights through the following steps:

1. Check if a HA is required. This is the case if the current layer m contains only two signals of the current weight n and one signal of each lower weight:
   if $(\sum_i = 0^n b_{i,m} == n+1$ AND $b_{n,m} == 2) \quad \rightarrow \quad ha_{n,m} = 1$
2. Determine the number of FAs required. This is done by taking the integer floor of the number of signals at the input of the current layer m of with the current weight n divided by 3 (as explained in section 2.3.4 a FA can be seen as a three-to-two compressor):
   $fa_{n,m} = \left\lfloor \frac{b_{n,m}}{3} \right\rfloor$
3. Determine the number of outputs of the current layer m (the outputs are stored as inputs to the next layer $m+1$) with the same weight n. Some signals at the input of the current layer cannot be compressed and thus will carry through to the next layer without passing through an FA or HA. A single FA will reduce the number of signals of the same weight by two, while an HA reduces the number of signals of the same weight by one:
   $\Delta b = -2fa - ha.$
   Furthermore, the FA and HA carry signals assigned in the next step should also be taken into account:
   $b_{n,m+1} = b_{n,m+1} + b_{n,m} - 2fa_{n,m} - ha_{n,m}$

Postion	Cell	IN	L1 Gates	L1 Outputs	L2 Gates	L2 Outputs	L3 Gates	L3 Outputs	L4 Gates	L4 Outputs	L5 Gates	L5 Outputs	L6 Gates	L6 Outputs	L7 Gates	L7 Outputs	L8 Gates	L8 Outputs	L9 Gates	L9 Outputs	L10 Gates	L10 Outputs	L11 Gates	L11 Outputs	L12 Gates	L12 Outputs
b0	FA	256	85	86	28	30	10	10	3	4	1	2		1		1		1		1		1		1		1
	HA												1													
b1	FA	0		85	28	57	19	29	9	14	4	7	2	4	1	2		1		1		1		1		1
	HA																1									
b2	FA	0				28	9	29	9	20	6	12	4	6	2	3	1	2		1		1		1		1
	HA																		1							
b3	FA	0						9	3	12	4	10	3	8	2	6	2	3	1	2		1		1		1
	HA																				1					
b4	FA	0								3	1	5	1	6	2	4	1	4	1	3	1	2		1		1
	HA																						1			
b5	4-OR	0										1	1	2	1	3	1	2	1	2	1	2	1	2	1	1
Total	FA	243	85		56		38		24		16		10		7		4		2		1		0		0	
	HA	5	0		0		0		0		0		1		0		1		1		1		1		0	
	OR	7	0		0		0		0		0		1		1		1		1		1		1		1	
Total Signals				##		##		77		53		37		27		19		13		10		8		7		6

Figure 3.2: Critical adder tree generated by the SKILL script in appendix H.2. The columns correspond to the layer of the critical adder tree, while the rows correspond to the weights of the signals. For each layer and weight the number of FAs, HAs and output signals are shown. For the final weight the number of 4-input OR gates is shown.

4. Determine the number of outputs of the current layer m (the outputs are stored as inputs to the next layer $m+1$) with the next weight $n+1$. These are the carry signals generated by the FAs and HAs. Both have only one carry signal:
   $b_{n+1,m+1} = b_{n+1,m} + 2fa_{n,m} + ha_{n,m}$

5. For signals with a weight of $output - 1$ in the current layer m add the OR gates (if less than 4 signals are available, the remaining inputs of the 4-input OR gate are tied to the ground).:
   if $b_{outputs,m} > 1 \rightarrow or4_m = \lceil \frac{b_{outputs,m}}{4} \rceil$

3.3.1. Critical Adder Tree Implementation

The algorithm to generate the critical adder tree is implemented the SKILL script provided in appendix H.2. This script first determines the structure of the critical adder tree and then generates the Virtuoso schematic. A simplified layout is created and extracted for the basic components (FA and HA) where the input and output net wires have a length equal to four standard cell widths to account for the parasitics associated with the interconnect. However, this remains an approximation, so the final power consumption can deviate from the results obtained using these models.
Using the SKILL script, a critical adder tree is generated for 256 inputs with a weight of 0. Figure 3.2 shows the structure of the generated full adder tree.

3.3.2. Critical Adder Tree error probability

The Critical Redundancy Adder Tree introduces an inherent error source through its architecture. This error occurs when the critical redundancy adder tree is designed for a given upper event limit, but the number of events during a given sample period exceeds this designed value (an overflow). The maximum number of events at the input of the critical redundancy adder tree is limited by its width b as:

$$X_{crat,max} = 2^b - 1 \tag{3.27}$$

Using equation 3.21 the error probability due to an overflow of a section of the critical redundancy adder tree can be calculated using:

$$\epsilon_{crat,of} = P(X > X_{crat,max}) = 1 - e^{-r_{clst}T} \sum_{i=0}^{2^b} \frac{(r_{clst}T)^i}{i!} \tag{3.28}$$

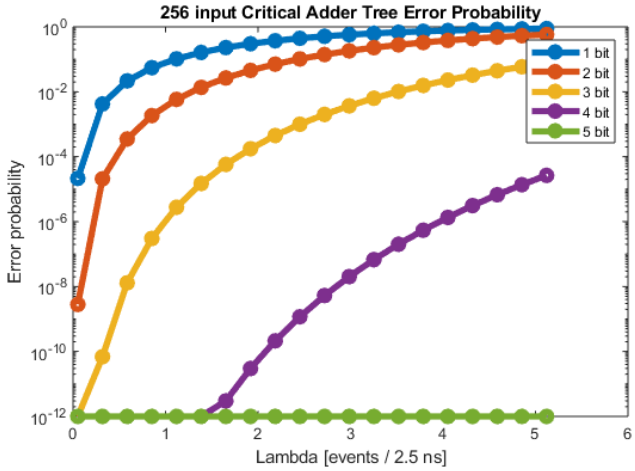


Figure 3.3: Theoretical error probability for the 256 input critical adder tree for a width of 1-5 bits as a function of the event rate per time interval of 2.5 ns λ. To prevent issues with displaying error probabilities of zero within a logarithmic plot, the probabilities are configured to saturate at 10^{-12}

where T is the sample period. By carefully choosing the width of the critical redundancy adder tree, the error $\epsilon_{crat,of}$ can be minimized. This is illustrated in Figure 3.3.

3.4. Pulse Counter

The concept of a pulse counter was first introduced in Section 2.3.6. The asynchronous outputs of the detectors are routed to a central location within the cluster. Here, the rising edges are converted into short pulses, which trigger a shift register with its input connected to VDD. The shift register output then forms a thermometer code that represents the number of events. Using an encoder, the thermometer code is converted to a binary integer. The design of the pulse counter is divided into the components shown in Figure 3.4 with the following responsibilities:

- **Input Buffer**: This component ensures that a sufficient margin is maintained between the asynchronous inputs and the clock edges to ensure that they are counted within the correct sampling period.
- **Pulse Generator**: This component is responsible for converting the rising edges at its input to pulses with a short pulse width at its output.
- **Merger**: This component takes the outputs of each pulse generator and merges the signals onto a single wire that is connected to the shift register.
- **Pulse Router**: The pulse router connects the merger output to either the top or the bottom pulsed shift register, depending on its control signal. One of the pulsed shift registers will detect the pulses, while the other is resetting. At the next clock edge, they switch roles. This is done to ensure that pulses can be detected continuously.
- **Pulsed Shift register**: This component shifts a logical '1' over one position for each pulse received.
- **Encoder**: This component converts the thermometer code output from the shift register to a binary integer.
- **Controller**: The controller regulates the control signals required for the pulse counter. It takes a clock signal as input and outputs the clear signal for the shift register and the delayed clock signals for the memory elements.

As the design of the pulse generator, merger, and shift register can be greatly influenced by the layout parasitics, the simulations for these components are performed using the post-layout parasitics.

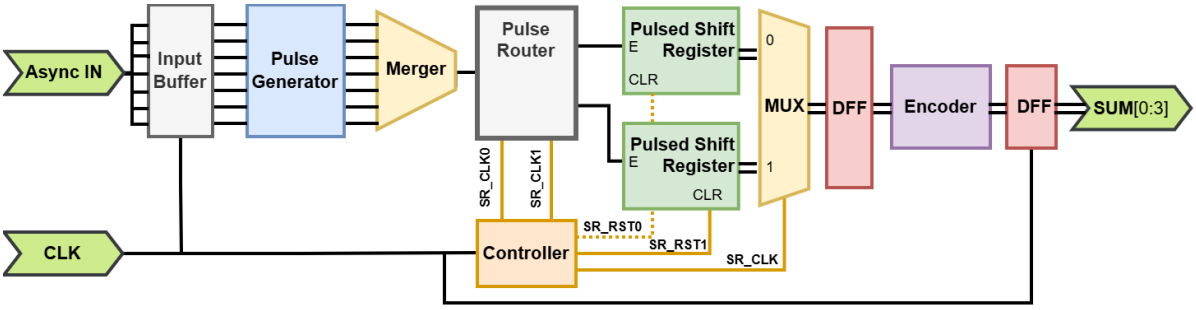


Figure 3.4: Overview of the pulse counter architecture.

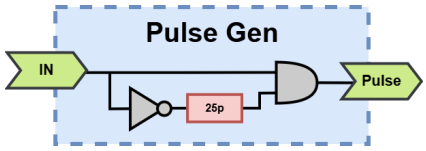


Figure 3.5: Pulse Generator that converts a rising edge at its input to a short pulse at its output.

Furthermore, the entire system is implemented using a double data rate (DDR) architecture, the reason for which will become clear later in Section 4.4.

Pulse Generator

The pulse generator is implemented using an inverter chain with a NAND gate, as shown in Figure 3.5. The input of the pulse counter is connected directly to the NAND gate and an inverter with a delay element at its output. The output of the delay element is initially low, which causes the NAND gate to output a high signal. After the input propagates through the inverter and delay element, the output of the NAND gate becomes low. This results in a short pulse at the output of the NAND gate, the width of which can be adjusted by increasing the delay of the delay element.

The post-layout Monte Carlo simulation shown in Figure 3.6 shows the distribution of the output pulse width when varying the process and the local device mismatch. The average pulse width is 48 ps, and the 3σ limits are 35 ps and 61 ps.

Input Buffer

As the pulse counter operates with asynchronous inputs, it is important to provide a sufficient margin between inputs that arrive close to the clock edge. This ensures that the system counts these inputs within the correct sampling period. To achieve this, the rising edges from asynchronous inputs that occur just after a clock edge are delayed using a tri-state buffer. The clock edges are used to generate

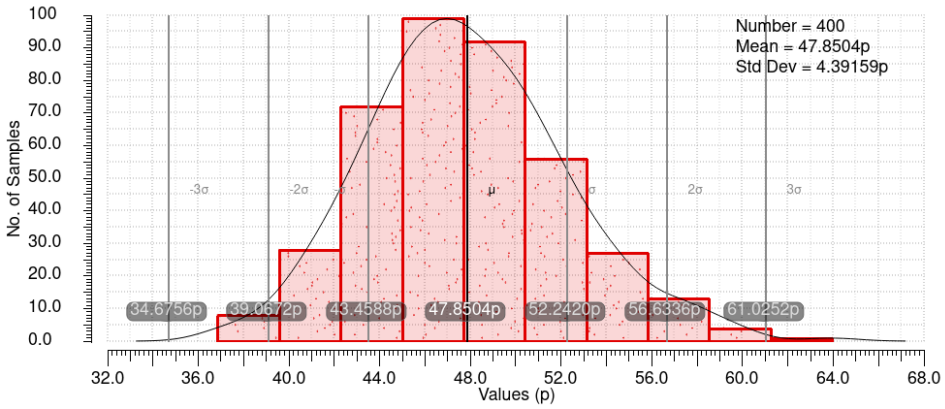


Figure 3.6: Distribution of pulse-width at the output of the pulse generator resulting from a Monte Carlo simulation. The mean pulse width is 48 ps, and the 3σ limits of the pulse width are 35 and 61 ps.

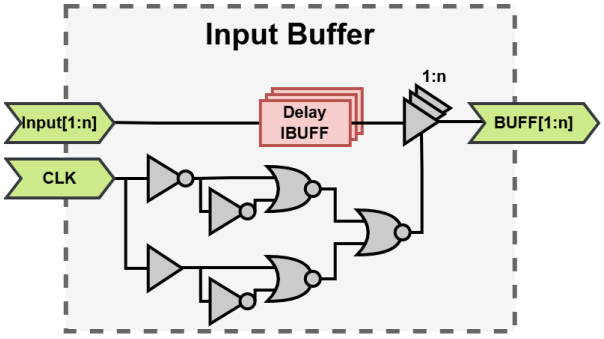


Figure 3.7: Schematic of the input buffer.

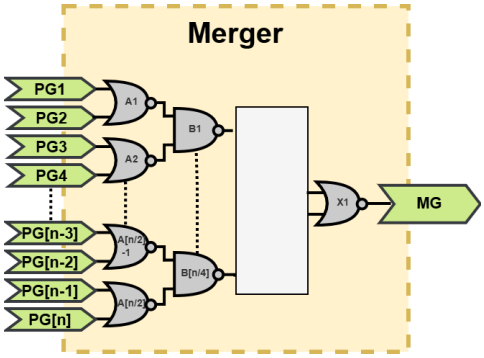


Figure 3.8: Schematic of the merger.

a short pulse, during which the tri-state buffer's output is high impedance. This ensures that the outputs that were already high stay high, while rising and falling edges at the buffer input are only able to pass through once the tri-state buffer is enabled. The input buffer circuit is shown in Figure 3.7.

Merger

The merger is responsible for combining the outputs of each pulse generator into a single signal. This can be achieved by using a tree-like OR gate structure. However, in CMOS technology, using noninverting gates is less efficient compared to their inverting counterpart, as the noninverting behavior is implemented by adding an inverter to the output of the equivalent inverting gate. Therefore, the merger is implemented using alternating NOR and NAND gates as illustrated in Figure 3.8. The number of layers required depends on the number of inputs the NOR and NAND gates have. Two variations of the merger are tested using two and four input gates, respectively.

The merger schematics are simulated for the different process corners with input pulse widths of 35 ps (the shortest pulse width observed in Figure 3.6), the results of which are shown in Figure 3.9. It can be observed that the implementation using 4-input gates regularly fails to propagate pulses to its output. This can be attributed to the additional parasitic capacitance resulting from the increased widths of the transistors required to provide sufficient drive strength in gates with a large fan-in [44]. The merger implemented using 2-input gates performs significantly better and is able to merge pulses with a much smaller width. However, pulses that are too close together are seen as a single long pulse at the output. This must be considered during the design of the shift register.

The final merger is designed for 256 inputs using two input gates so that the pulse counter can be compared directly to the full adder tree and the critical adder tree. To characterize the design, a test bench is created that connects the output of the pulse generators to the input of the merger. Using a Monte Carlo simulation, the pulse width distribution is obtained at the output as shown in Figure 3.10. These results are for both variations in the process corner, representing die-to-die variations, and local mismatch of device dimensions, representing the on-die variation [45].

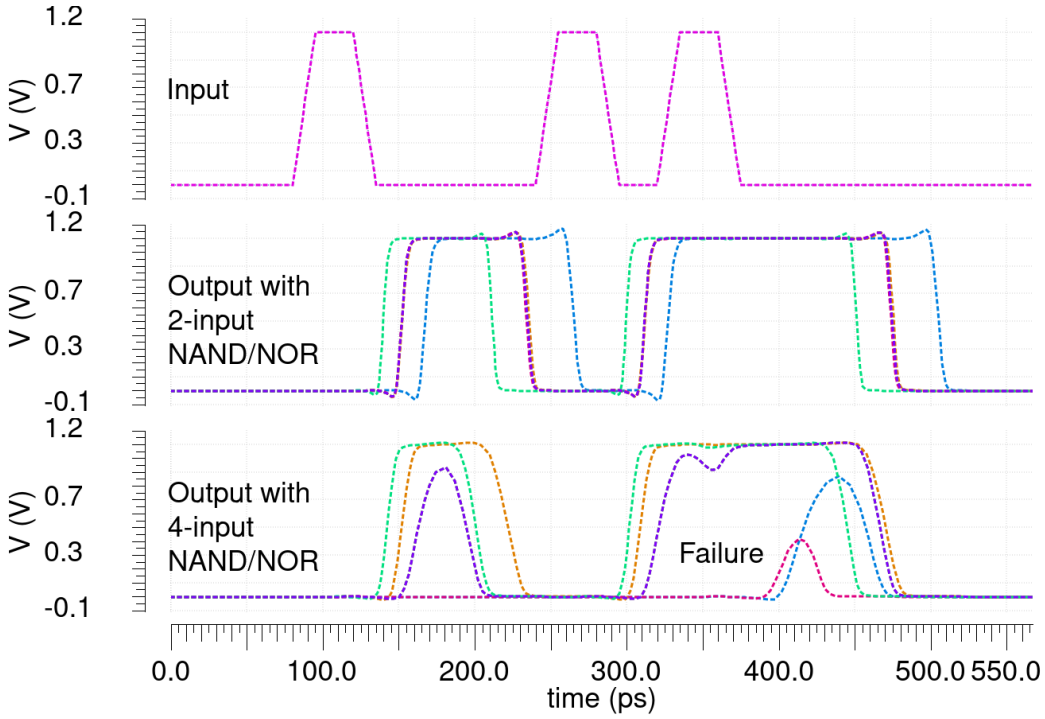


Figure 3.9: Schematic simulation results of 64-input mergers implemented using 2-input (center) and 4-input (bottom) gates. The merger implemented using 4-input gates fails to produce a output pulse at multiple process corners, and has less steep edges compared to the merger implemented using 2-input gates. The merger implemented using 2-input gates is able to propagate the 40 ps pulses seen at its input, to its output as 60-100 ps pulses

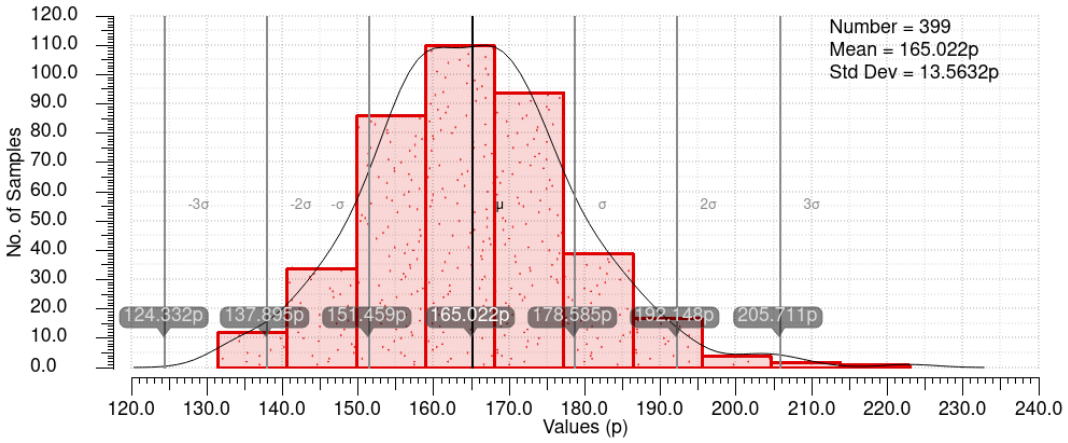


Figure 3.10: Post layout Monte Carlo simulation results of the pulse generator connected to the input of the 256-input merger. The mean output pulse width of the merger is 165 ps, and the 3σ limits are 124 ps and 205 ps.

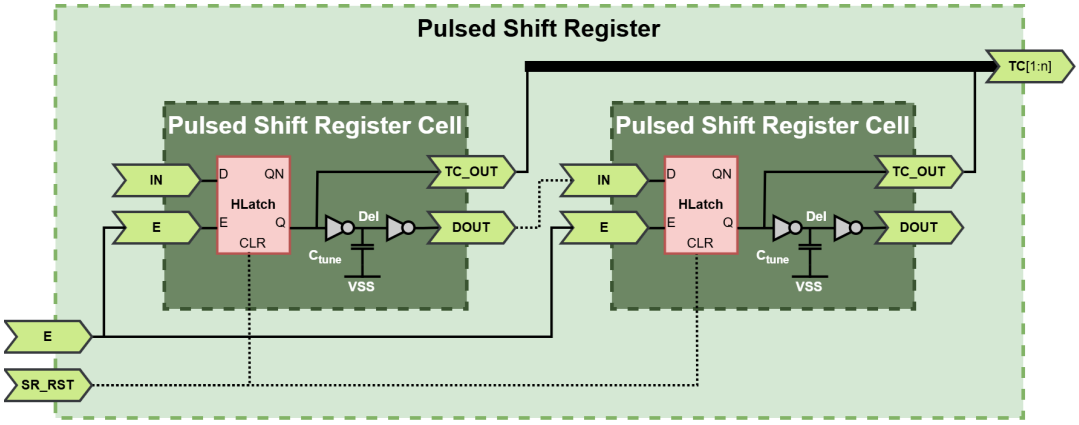
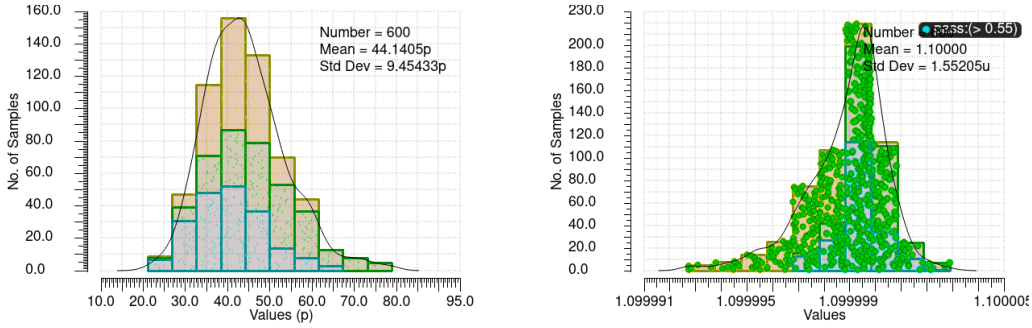


Figure 3.11: Schematic of a pulsed shift register with two cells.



(a) Monte Carlo simulation for the overlap time t_2 of the inputs to the second cell at the TT (blue) SS (green) and FF (yellow) process corners with device mismatch. This shows that the pulsed shift register operates reliably within the different process corners

(b) Monte Carlo simulation results showing the voltage difference between the TC-output of the first and second cell. These results illustrate that the shift register operates as expected even with locally mismatched devices.

Figure 3.12: Monte Carlo simulation results of the pulsed shift register. The input is provided to the pulse generator, which is connected to the merger. This covers not only the corners and mismatch within the pulsed shift register, but also for the pulse generator and merger.

Pulsed Shift Register

Figure 3.11 shows two cells of the shift register. The shift register is designed using High-Enabled Latches, with an asynchronous clear. These are provided within the SCL and can operate with pulses as short as 42 ps. A high signal on the **E** allows the input signal from **D** to propagate to **Q**. A small inverter is used to buffer the output. The capacitor C_{tune} is slowly discharged. Once discharged sufficiently, the second inverter output changes. When the **CLR** input is asserted the outputs **Q** and **QN** reset to '0', and '1' respectively.

The value of the tuning capacitor C_{tune} should be matched with the largest expected pulse width at the input. Typically, within the same die, the mismatch between the device dimensions is the dominant source of variation [45]. To ensure that a single pulse at **E** is never registered as multiple events, the value of C_{tune} must be selected such that a single pulse at the input is never registered as two events due to the local mismatch of the devices. This is verified using a Monte Carlo simulation where the device mismatch is varied within the TT, FF, and SS process corners. The circuit is considered to pass the test if only the first output is high while the second output is low. The results in Figure 3.12a show the time that the inputs to the second cell are both high. Figure 3.12b shows the histogram of the voltage difference between the output of the first and second cells, showing that the mismatch within a single process corner does not negatively impact this design.

Encoder
Thermometer-to-binary encoders are well-studied [46]. There are four main architectures, namely a lookup table, a tree of OR gates, a Wallace tree, and finally a multiplexer-based design. Depending on the application, certain architectures are preferred. As this application prioritizes low power, the pulse counter encoder will be implemented using the multiplexer-based design borrowed from Sireesha and Kumar [47] expanded to a 4-bit output.

Controller
The pulse counter requires a controller that provides the control signals for the Pulse Router, Pulsed shift registers, TC-Mux and DFF's. If a single shift register is used, there would be approximately 100 ps for every sample period where no events can be detected, as this is the time required for the pulsed shift register to reset. This would lead to an increased detection error rate. To solve this, the pulse counter architecture presented in figure 3.4 incorporates two pulsed shift registers. While the system clock is high, the output of the merger is connected to the bottom shift register, and while the clock is low, it is connected to the top shift register. The inactive shift register can be reset while the active shift register counts the pulses, allowing the pulse counter to continuously detect events. By adding a clock divider at the input of the pulse counter the 400 MHz global clock can be reduced to 200 MHz clock. The alternating phases of the 200 MHz clock then align with the sampling periods. Alternatively, the pulse counter can be directly implemented within a DDR system (moreover in section 4.4).

Figure 3.13 shows the schematic of the control logic. The incoming clock signal is delayed to match the delay of the input buffer in Figure 3.7. The clock is then split into three branches; one for the top shift register, one for the bottom shift register, and one for the TC-Mux and DFF's. Each has their own delay so these can be tuned individually, as rising and falling edges do not always have the same propagation delay.

The reset signals for the pulsed shift registers are generated through the following boolean equations:

$$RST0 = \overline{(\overline{RST} + (\overline{SR\_CLK0}\ \ SR\_CLK))} = RST(SR\_CLK0 + \overline{SR\_CLK}) \tag{3.29}$$

$$RST1 = \overline{(\overline{RST} + (\overline{SR\_CLK1}\ \ \overline{SR\_CLK}))} = RST(SR\_CLK1 + SR\_CLK) \tag{3.30}$$

Furthermore, a delay is added from SR_CLK0 and SR_CLK1 to the combinational logic that implements these functions. This delay can be used to keep the shift registers disabled for some time after the pulse are routed to them, which ensures that asynchronous inputs arriving just before a clock edge are not counted by both the pulsed shift registers. The following list provides an overview of the tunable controller parameters.

- **IBUFF**: This delay must be matched with the propagation delay from an asynchronous input to the output of the input buffer stage.
- **CLK[0-1]**: If this delay is too small, asynchronous inputs that arrive just before a clock edge are not properly detected, as the pulse router is switched to early. If this delay is too large, the mergers output stays connected to the wrong pulsed shift register for too long. However, post-clockedge inputs are slightly delayed by the input buffer. Therefore, this delay is ideally set to some value in between the propagation delay from the pulse generator input to the merger output, and the pulse width of the input buffers tri-state control signal as is illustrated in Figures 3.14 and 3.15.
- **DFF**: This delay does not require accurate tuning, as long as it is larger than the propagation delay from the input of the pulse generator to the output of the TC-Mux. This ensures that no race conditions can occur when storing the thermometer code.
- **RST[0-1]**: This delay must be tuned quite accurately. Figure 3.10 presented the pulse widths at the output of the merger. Therefore, if an input would arrive just before a clock edge, a pulse would be seen at the merger's output, which extends into the next sample period. This pulse must only be registered by the pulsed shift register that was active before the clock edge, while the pulsed shift register that is active after the clock edge must not count this pulse. This is done by delaying the enabling of the shift registers (enabled when the reset signal is high) so that the

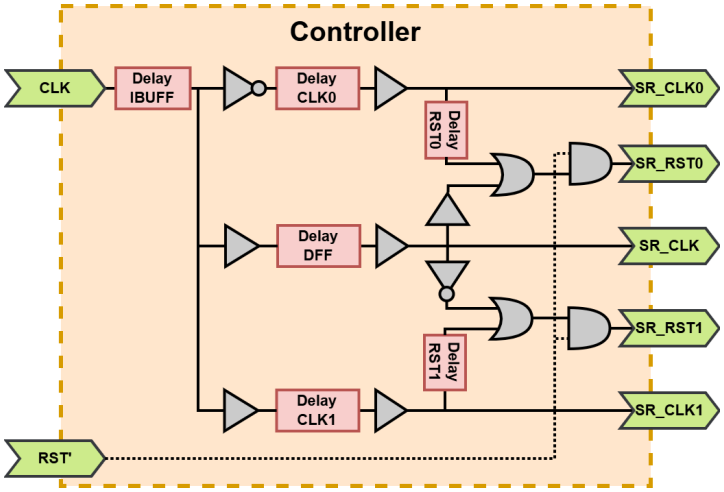


Figure 3.13: The pulse counter controller delays the clock signal to match the propagation delay through the pulse generator, merger and shift register in Figure 3.4 for use in the thermometer code flip flops.

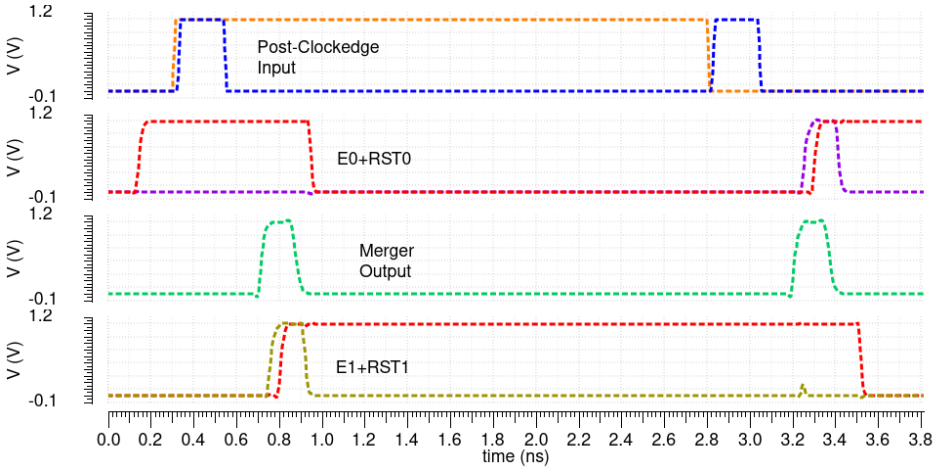


Figure 3.14: Simulation showing Post-edge events are delayed so that they are seen at **E** just as the pulsed shift register is enabled

trailing pulses from the previous sampling period are not registered. Because of the input buffer, inputs that arrive just after a clock edge are only converted into pulses after the tri-state buffers are enabled again. This allows for an acceptable margin while tuning this delay. This delay must be configured such that the sum of **CLK[0-1]** and **RST[0-1]** is larger than the propagation delay from the rising edge at the input of the pulse generator to the falling edge at the output of the pulse router. In this way, the pulsed shift register is only enabled after the pulse from the preclockedge input has decayed, as illustrated in Figure 3.15.

3.4.1. Pulse Counter Implementation

Two pulse counters are implemented, both containing 15 cells in the shift register, which allows up to 15 events to be tracked per sampling period. The first implementation has an input buffer to separate events from the clock edges, while the second implementation does not use an input buffer. As the tri-state buffers in the input buffer significantly increase the clock load, it must be verified that this load increase is a worthwhile trade-off to decrease the error probability.

3.4.2. Pulse Counter error probability

The pulse counter architecture introduces multiple inherent error sources. The first error source introduced by the pulse counter is due to the overlap of input signals. The second error source is similar to

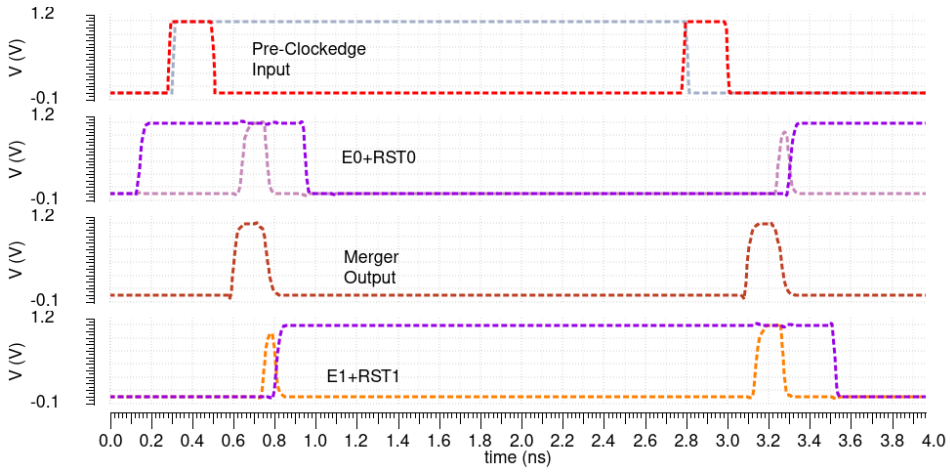


Figure 3.15: Simulation showing pulses resulting from Pre-edge events are seen by both **E0** and **E1**, but due to the delayed enabling of the pulsed shift registers they are not registered.

the one introduced by the critical redundancy adder tree, namely an overflow.

Overlap error

The pulse counter is capable of detecting partially overlapped pulses. Given that the mean pulse width at the output of the merger is $\tau_{mg,\mu}$, the mean delay of a single shift register cell is $\tau_{sr,\mu}$, and the hold time for the latch is τ_l. The pulse width at the output of the merger and the delay of a single shift register cell are matched to each other. However, this is not an exact match. The mismatch can be defined as follows:

$$\tau_{mm} = \tau_{sr,\mu} - \tau_{mg,\mu} \tag{3.31}$$

A negative mismatch would indicate that a single pulse at the output of the merger would most likely shift the pulsed shift register by two positions instead of one. In case of a positive mismatch two overlapping pulses can be registered correctly as separate events. For this, the time t_2 between the two pulses must be larger than the sum of the mismatch and the hold time to ensure the next latch is triggered correctly:

$$t_2 > \tau_{mm} + \tau_l \tag{3.32}$$

Similarly, three overlapping pulses can only registered as separate events if the third pulse arrives after the second pulse has been registered and t_2 seconds have passed. The time t_3 between the first and third pulses must therefore satisfy:

$$t_3 > \tau_{sr,\mu} + \tau_{mm} + \tau_l \tag{3.33}$$

Following this, an overlap error occurs when either of the following occurs:

1. More than one event occurs within an interval t_2 as defined by equation 3.32.
2. More than three events occur within the interval t_3

Given a detector matrix whose output is characterized by equation 3.13, the overlap error due to two or more events occurring within an interval t_2 can be found using equation 3.21:

$$P(E[t_2] \geq 2) = 1 - e^{-R \times t_2} \sum_{i=0}^{1} \frac{(R \times t_2)^i}{i!} \tag{3.34}$$

The probability that no more than one event occurs within any interval t_2 during a sample period T is found using complementary probability:

$$P_{succes1} = (1 - P(E[t_2] \geq 2))^{T/t_2} = \left(e^{-R \times t_2} \sum_{i=0}^{1} \frac{(R \times t_2)^i}{i!} \right)^{T/t} \tag{3.35}$$

The probability that more than two events occur within the interval t_3 can also be found using equation 3.21:

$$P(E[t_3] \geq 3) = 1 - e^{-Rt_3} \sum_{i=0}^{2} \frac{(Rt_3)^i}{i!} \tag{3.36}$$

The probability that no more than two events occur within any interval t_3 during a sample period T is also found using complementary probability:

$$P_{success2} = (1 - P(E[t_3] \geq 3))^{T/t_3)}) = \left(e^{-Rt_3} \sum_{i=0}^{2} \frac{(Rt_3)^i}{i!} \right)^{T/t_3} \tag{3.37}$$

$P_{success1}$ and $P_{success2}$ are independent. The probability that no overlap errors occur during an interval T is therefore given by the product:

$$P_{pass} = P_{success1} \times P_{success2} \tag{3.38}$$

Consequently, the probability of an overlap error occurring during an interval T is given by:

$$\epsilon_{pc,ol} = 1 - P_{pass} \tag{3.39}$$

$$\epsilon_{pc,ol} = 1 - \left(\left[ ]e^{-R \times t_2} \sum_{i=0}^{1} \frac{(R \times t_2)^i}{i!} \right]^{T/t} \right) \left(\left[ ]e^{-Rt_3} \sum_{i=0}^{2} \frac{(Rt_3)^i}{i!} \right]^{T/t_3} \right) \tag{3.40}$$

Overflow error

If the total number of events during a sample period at the input of the pulse counter exceeds the number of positions b in the shift register, an overflow occurs. In fact, the number of events that can be detected by the pulse counter within an interval T is further limited by the width of the pulses generated from the detector outputs as:

$$b = \left\lfloor \frac{T}{\tau_{mg,\mu}} \right\rfloor \tag{3.41}$$

The error probability due to an overflow can then be expressed as:

$$\epsilon_{pc,of} = P(E[T] \geq b) = 1 - e^{-RT} \sum_{i=0}^{b} \frac{(RT)^i}{i!} \tag{3.42}$$

Total error

The total error probability of the pulse counter architecture is determined using equations 3.40 and 3.42:

$$\epsilon_{pc} = \epsilon_{pc,ol} + \epsilon_{pc,of} \tag{3.43}$$

Given the error probability from equation 3.43, on average one out of $\frac{1}{\epsilon_{pc}}$ sampling periods will have an error. On average, each sampling period will have λ successfully detected events. Under the assumption that whenever an error occurs, only a single error occurs, the error probability can be expressed as the error rate in parts per million (ppm) ε through

$$\varepsilon = \frac{\lambda}{\epsilon_{pc}} \tag{3.44}$$

The pulse counter designed in this section has 256 inputs, a pulsed shift register with a width of 15, an overlap margin $t_2 = 55\ ps$ (see figure 3.12a) and a merger output pulse width of $\tau_{mg,\mu} = 165\ ps$ (see figure 3.10). Using equation 3.43 the probability of an incorrect measurement within a sample period of 2.5 ns is determined for a range of event rates λ. The resulting graph is shown in figure 3.16, illustrating how the pulse counter excels with high-sparsity matrices but sees a degradation in performance as the event rate increases.

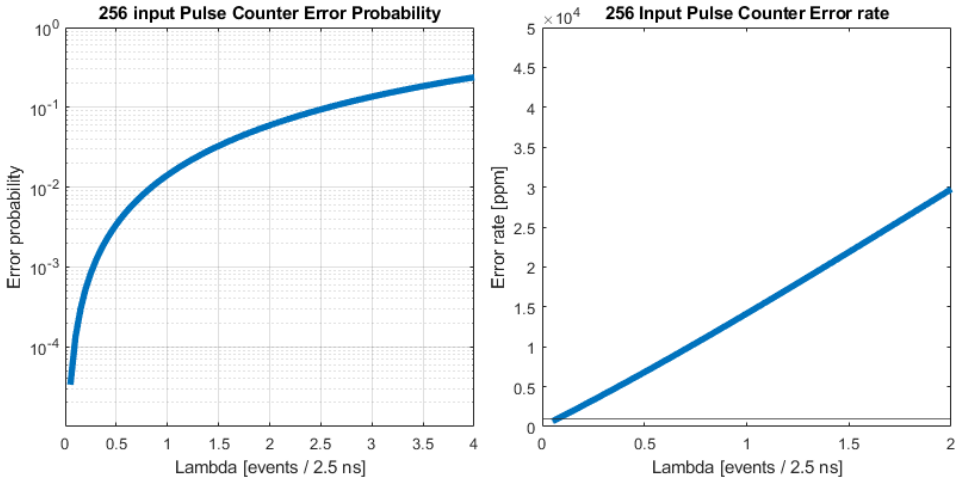


Figure 3.16: Theoretical error probability (left) and error rate (right) for the 256 input pulse counter with a width of 15 as a function of the event rate λ per 2.5 ns.

3.5. Comparison

To compare the three read-out architectures, they have been implemented for a detector matrix containing 256 sources and simulated over a range of expected event rates. A detector matrix with 256 sources 256! possible input combinations. It would be impossible to cover every input combination during the evaluation of an architecture. Instead, proper coverage is obtained using random inputs. For this, three Verilog-AMS components and a test bench are used.

The first Verilog-AMS component models the 256 sources within the cluster as Poissonian sources with an expected event rate λ per 2.5 ns. To model detectors with an asynchronous output, the output is asserted for a configured amount of time before returning to a digital low output. To model detectors with a synchronous output, the output is asserted only at a clock edge, and returned to a digital low at the following clock edge. The Verilog-AMS code for this component is provided in Appendix I.1.

The second Verilog-AMS component is a counter, which keeps track of the number of rising edges that occur within a sample period and outputs this as an 8-bit digital value after the next clock edge. The true count is used as a reference to verify the accuracy of the output from the readout architectures. The code for this component is given in Appendix I.2.

The third Verilog-AMS component takes the output of the counter and the output of the system under test (SUT), and determines the absolute error. The absolute error can be used to further investigate the accuracy of the different readout architectures. In addition, the total number of events is tracked along with the total number of missed events, allowing the relative error to be determined. The code for this component is given in Appendix I.3. Figure 3.17 shows the qualification of a Poissonian cluster connected to a counter. The inputs are shown to follow a Poissonian distribution, and the counter directly registers each rising edge. At the next clock edge, the true count is outputted as a binary integer.

Using these three Verilog-AMS components, the test bench shown in figure 3.18 is created. The output of the Verilog-AMS detector cluster is connected to the inputs of both the system under test (SUT) and the counter. The outputs of the SUT and counter are connected to DFF's to synchronize the results to the clock signal. The output of the Verilog-AMS counter serves as the ground truth and is compared against the output of the SUT using the error tracker to determine its performance.

3.5.1. Results

For the comparison of the four architectures, the testbench presented earlier in figure 3.18 is used. The Verilog-AMS clusters in each test bench are configured to use the same seed to ensure that each architecture is provided with the exact same inputs. The simulation is run for 2.5 μs, using a DDR clock signal of 200 MHz. For these simulations, the typical process corner is used for both NMOS and PMOS devices and the expected event rate λ is swept from 0.3 to 4.

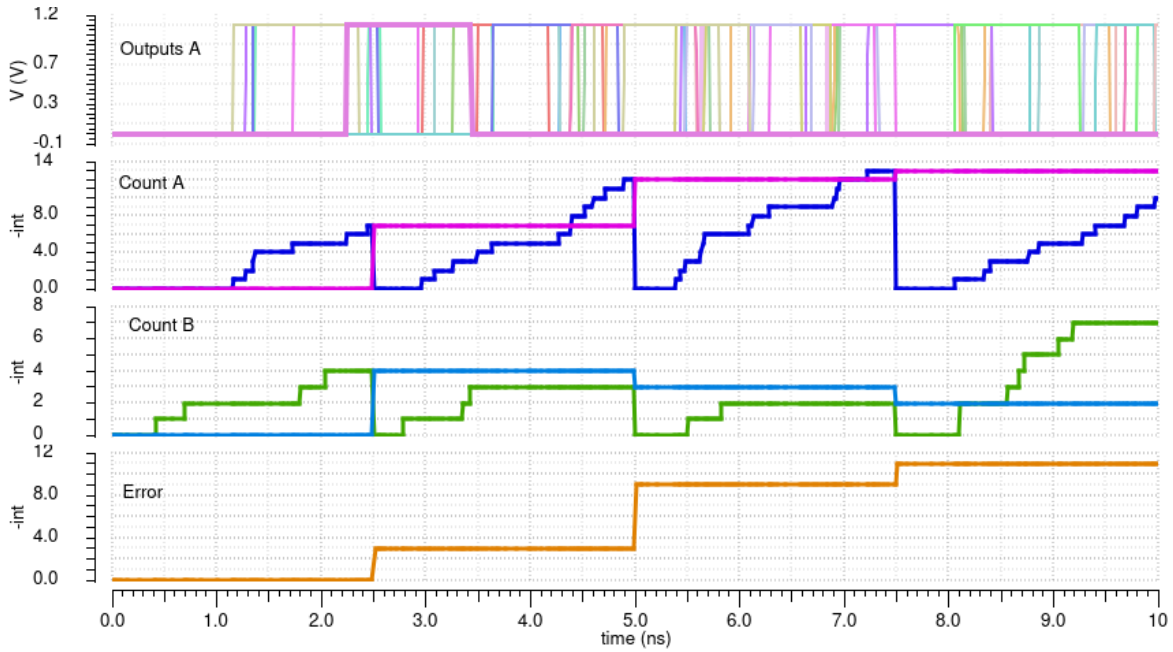


Figure 3.17: Simulation results with two Verilog-AMS clusters (with an assertion time of 1.2 ns) and counters connected to a single error tracker with a 400 MHz clock signal. The top graph shows the asynchronous outputs of the Verilog-AMS cluster. The second graph from the top shows the internal counter (Dark Blue) of the first Verilog-AMS counter, and the decimal value of its output (Magenta) provided at the rising edge of the clock. The third graph from the top graph shows the internal counter (Green) of a second Verilog-AMS counter, and the decimal value of its output (Blue) provided at the rising edge of the clock. The bottom graph shows the decimal value of the error signal (Orange) at the output of the Verilog-AMS error tracker (i.e. the difference between the outputs of the two Verilog-AMS counters)

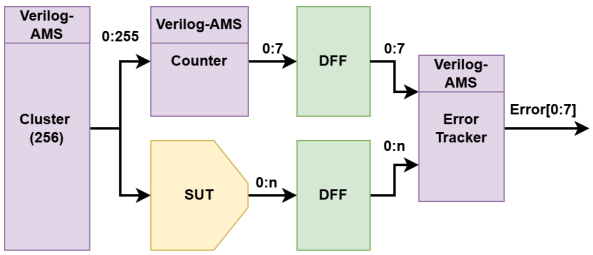


Figure 3.18: Diagram of the test bench used to evaluate the back-end readout architectures (System Under Test (SUT) in the diagram).

In the results a distinction is made between two different error measurements. Namely, the error probability and the error rate. The error probability indicates how likely it is that the backend readout architectures output does not match the actual number of events within a specific sampling period. The error rate indicates how many events are missed on average and is expressed in parts per million (ppm).

The error probability and error rate of the backend readout architectures for a simulation time of 2.5 μs (1000 sampling periods) are shown in figure 3.19. The full and critical adder trees are not included in the error probability plots, as none occurred. The comparison of the simulation results to the error probability and error rate derived in equations 3.43 and 3.44 respectively, show that the error probability more or less matches for the pulse counter with an input buffer. However, while deriving equation 3.44 the assumption was made that only a single error would occur. This assumption holds for smaller values of λ, but quickly diverges from the simulation results as λ increases.

It is clear that the input buffer improves performance, as the 1000 ppm error rate limit without input buffer is reached rather quickly as an event occurred near the clock edge. With input buffer, the error rate limit is reached at $\lambda = 0.437$. This comes at the cost of a significantly higher power consumption of 1.3 mW, whereas the pulse counter without input buffer only draws 0.3 mW.

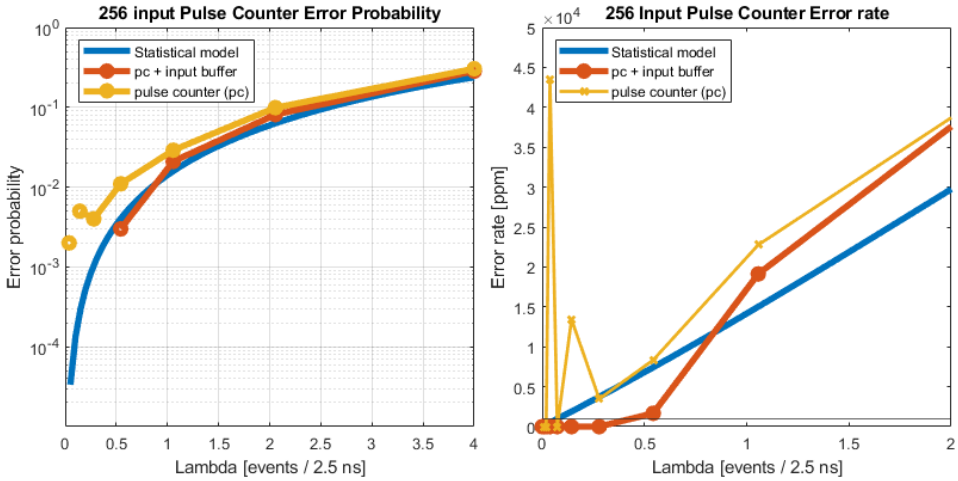


Figure 3.19: The simulated and calculated error probabilities (left) and error rates (right) of a 256 input pulse counter.

The RMS power consumption from these simulations is shown in figure 3.20.
The best performing architecture appears to be the critical adder tree, performing identically to the full adder tree, with a slightly lower power consumption. Not much can be said regarding the overflow error probability in figure 3.3 as the simulation time only covers 1000 sampling periods,. Therefore, the chance of observing an overflow error for a critical adder tree of 4 bits within the provided expected event range is very small.

The results might lead to the conclusion that the critical adder tree is superior in every way. However, the power consumption results only account for the combinational logic. The pulse counter architecture does not require the clock to be distributed to the detectors, which will have to be taken into account in the following chapter when deciding which backend readout architectures should be used. In Section 4.2 the clock distribution will be designed for the sensor matrix which will provide insight into the power consumption associated with the clock distribution.

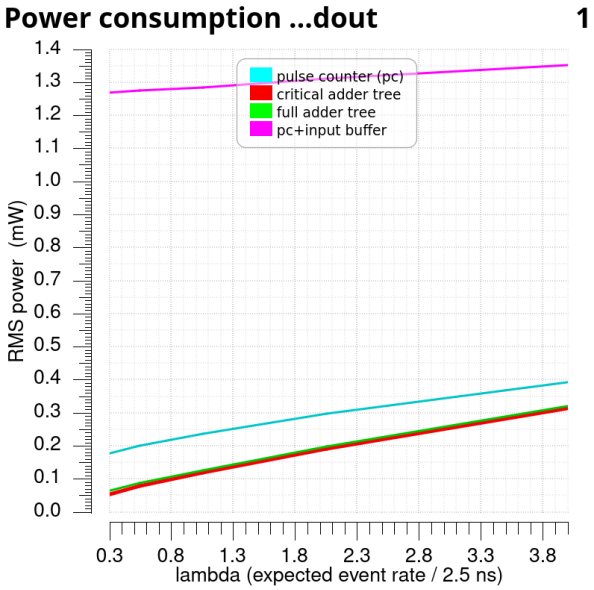


Figure 3.20: RMS power for each of the four readout architectures. These results are obtained for the TT process corner, and a simulation time of 2.5 μs.

4

ASIC Design

In Section 2.1 the system was broken down into subsystems. These subsystems are briefly repeated:

1. **Front-End Readout circuit (Detector):** Each front-end readout circuit needs to be provided with a clock signal, a digital and analog supply, and five reference voltages that are used within the analog circuits. Additionally every detector requires a reset signal
2. **Sensor matrix:** The sensor matrix consists of rows and columns of detectors and their front-end readout circuits, with the dimensions provided in Section 2.1. Furthermore, the sensor matrix must be comply with *Requirement 8*
3. **Clock Distribution:** The clock must be distributed to the front-end readout circuits according to *Requirement 10*. In addition, the clock must be distributed to all clocking elements in the back-end readout system.
4. **Back-End Readout circuit:** The back-end readout circuit is responsible for summing the output of each individual detector within the detector grid according to *Requirement 1* - *Requirement 6*.
5. **Memory:** The data outputted by the Back-End Readout Circuit need to be stored in a buffer before being sent to the host system according to *Requirement 11*.
6. **Off-Chip IO:** In order to communicate with the host system, and to receive the global clock and control signals, various Off-Chip IO's need to be implemented according to *Requirement 12*.
7. **Configuration registers:** These registers can be written by the host controller to specify the grid dimensions of the PE beam.

A diagram of the ASIC and the connection to the host system is provided in figure 4.1. This chapter will start by defining the structure of the pipeline for the backend readout subsystem using the readout architectures discussed in the previous chapter. At this point, not all the information required for the implementation of the backend readout subsystem has been presented. Therefore, the technique used for the clock distribution will be presented, followed by the floor-planning and signal routing within the ASIC. After this the double data rate memory architecture will be introduced. Finally, all the information required for the implementation of the backend readout subsystem is known. This chapter will then be concluded with the implementation of the backend readout subsystem.
The off-chip-io-, memory-, and configuration-subsystems provide no novelty, and can be borrowed from many other systems. Therefore, their design is not covered within this thesis.

4.1. Backend Readout Design

The concept of a pipelined architecture was introduced in section 2.3.4. In short, the addition of a large number of signals to a single integer is separated into pipeline segments. Each pipeline segment performs a small part of the addition over an entire clock cycle and presents its result as an input to the next pipeline segment during the next clock cycle. In this way, the complete addition is spread over

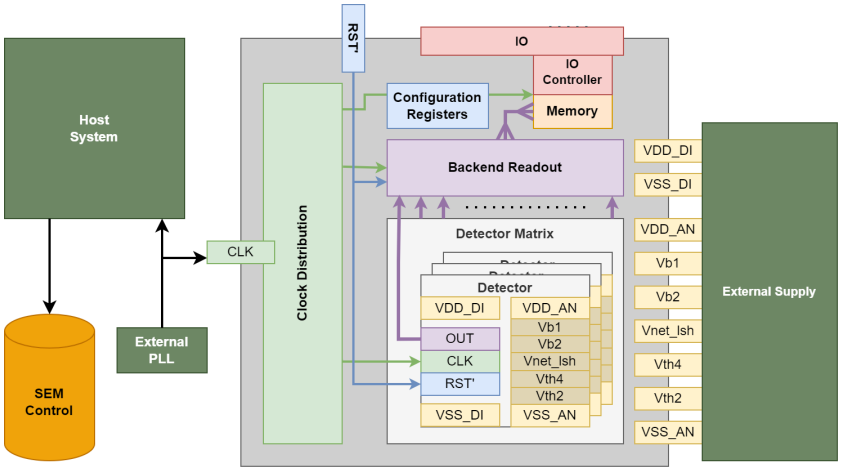


Figure 4.1: Diagram of the subsystems within the ASIC and their connections

multiple clock cycles, while being able to process new inputs for every clock cycle.

Within this system, the inputs to the pipeline come from the detectors. The physical detector pitch is 165 μm therefore, there is a significant challenge regarding the signal routing and propagation delays. In this section, we will look at various design considerations that will affect the power consumption, complexity, and performance of the back-end readout architecture. Using the design choices that best suit the system requirements presented in Section 2.1, the general structure of the pipeline is determined.

In the previous chapter the full adder tree, critical adder tree and pulse counter architectures were designed. The simulation results in figure 3.19 showed that the suitability of these architectures is highly dependent on the expected event rate of the matrix for which the readout is implemented. In this section the backend readout pipeline will be designed using these architectures. First, the structure of the readout pipeline will be determined. This will be followed by the detailed design of each pipeline segment.

4.1.1. Pipeline Structure

Appendix E shows the detailed derivation of the backend readout pipeline for a detector matrix of 128 × 128 detectors. Figure 4.2 shows the floor-plan of the backend readout pipeline. The readout pipeline consists of four pipeline segments L1-L4. The L1 pipeline segments produce the sum of 256 (16 ×16) detectors. The readout architecture used for the L1 pipeline segments will be determined based on the expected event rates of the detectors for which they perform the readout. Pipeline segments L2-L4 produce the sum of 1024, 4096 and 16384, respectively, and use the critical adder tree architecture.

4.1.2. L1 pipeline design

Equations 3.24, 3.28, and 3.43 showed that the error probabilities of each architecture are highly dependent on the event rate of the sparse matrix for which they are implemented. In order to quantitatively determine which of these architectures is most suitable for implementation within the ASIC, the event distribution of the sensor matrix must be known. Therefore, a model is created to estimate the incidence of particles within the sensor matrix. In Appendix B the system statistics presented in Section 2.1 are used together with the operating principles of a SEM (presented in Appendix A) to model the incidence of electrons at the sensor matrix. This is used to determine the event-rate at every single detector within the sensor matrix.

The derivation and detailed results of this model are presented and discussed in appendix G.2. A summary of the results is provided here in figures 4.3a and 4.3b. The total size of the modeled detector matrix is 128 by 128 detectors, the detailed design of which is discussed later in section 4.3.2. These

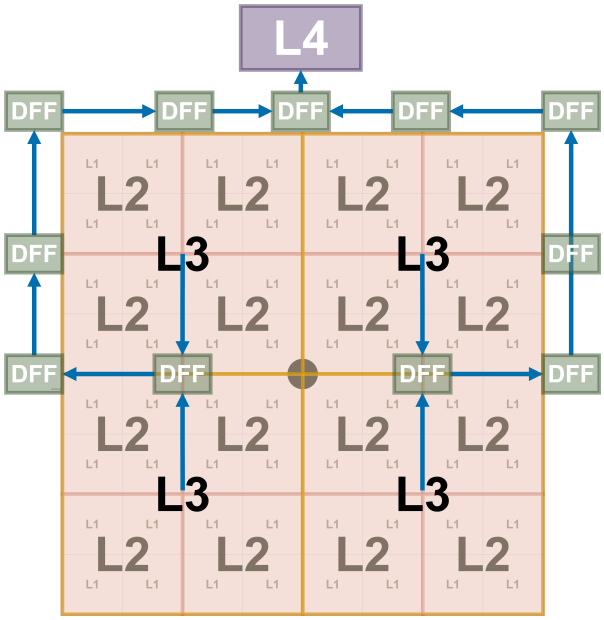


Figure 4.2: Structure of the backend readout pipeline. The individual pipeline segments are denoted by L1-L4, with L1 containing 256 individual detectors.

results will be used to determine the minimum size of the critical adder tree and the pulse counter for implementation within a cluster using equations 3.28 and 3.42 respectively.

For the pulse counter architecture, one of the major advantages is that the clock does not need to be routed to each individual detector. From Figure 3.19 we conclude that the pulse counter with an input buffer performs reliably up to an event rate of approximately 0.3 / 2.5 ns. Therefore, the choice is made to use the pulse counter architecture with input buffer for all clusters with an event expectation lower than 0.25 / 2.5 ns, and the critical adder tree architecture for all clusters with a larger event expectation. Using the results of the model from figures 4.3a and 4.3b the L1 pipeline segments are implemented as illustrated in figure 4.4.
The four L1 pipeline segments in the center of the detector matrix are implemented using the critical adder tree architecture, as have the largest expected event rate. The combinational logic is generated using the critical adder tree algorithm from section 3.3 . This critical adder tree is generated with an input of $[256]$ and an output width of 4 bits allowing the detection of up to 15 events.
All remaining L1 pipeline segments are implemented using the pulse counter architecture with an input buffer.

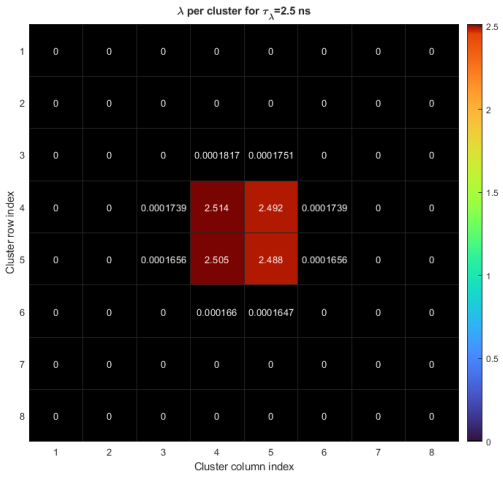
4.1.3. L2-L4 pipeline design

The combination logic of L2-L4 is generated using the critical adder tree algorithm from section 3.3. The combinational logic of the L2 pipeline segment is generated for an input of [4,4,4,4] and an output width of 5 bits. The combinational logic of the L3 pipeline segment is generated for an input of [4,4,4,4,4] and an output width of 6 bits. The combinational logic of the L4 pipeline segment is generated with an input of [4,4,4,4,4,4] and an output width of 6 bits.

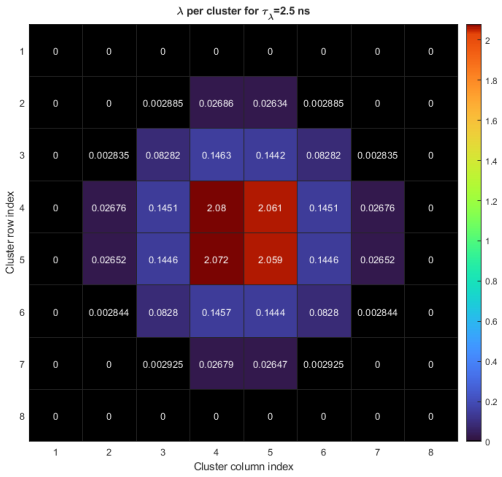
4.2. Clock Distribution

The clock distribution for the detector matrix and the backend readout subsystem is designed separately from that of the memory, configuration registers, and communication interfaces. In this way, the clock of the sensor matrix and other subsystems can be controlled independently from each other. Appendix F presents commonly used clock distribution techniques.

The analog circuits are sensitive to interference from nearby signals. For that reason, clock distribution techniques that require the clock signal to be routed over analog islands do not meet *Requirement 3*. Similarly, clock distribution techniques that have a large variation in clock skew are less likely to meet



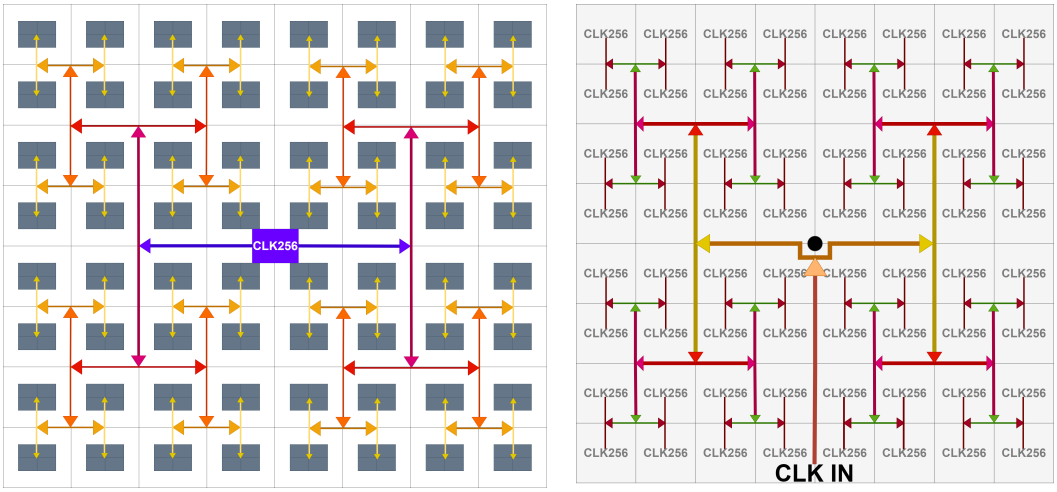
(a) Event expectation λ per 2.5 ns within clusters of 16 x 16 detectors resulting from the Matlab model in Appendix G.2. These result are obtained from the model using a bias voltage of 6000 V, and a working distance of 5 mm, with the sensor statistic presented in section 2.1.



(b) Event expectation λ per 2.5 ns within clusters of 16 x 16 detectors resulting from the Matlab model in Appendix G.2. These result are obtained from the model using a bias voltage of 6000 V, and a working distance of 15 mm, with the sensor statistic presented in section 2.1.

PCIB	PCIB	PCIB	PCIB	PCIB	PCIB	PCIB	PCIB
PCIB	PCIB	PCIB	PCIB	PCIB	PCIB	PCIB	PCIB
PCIB	PCIB	PCIB	PCIB	PCIB	PCIB	PCIB	PCIB
PCIB	PCIB	PCIB	CRAT	CRAT	PCIB	PCIB	PCIB
PCIB	PCIB	PCIB	CRAT	CRAT	PCIB	PCIB	PCIB
PCIB	PCIB	PCIB	PCIB	PCIB	PCIB	PCIB	PCIB
PCIB	PCIB	PCIB	PCIB	PCIB	PCIB	PCIB	PCIB
PCIB	PCIB	PCIB	PCIB	PCIB	PCIB	PCIB	PCIB

Figure 4.4: Overview of readout architectures used for the implementation of the L1 pipeline segment. Pulse counter with input buffer (PCIB) or critical adder tree (CRAT)



(a) Clock distribution using a balanced H-Tree for a cluster of 16×16 detectors

(b) Clock distribution using a balanced H-tree for the entire sensor matrix

Figure 4.5: Clock distribution to the sensor matrix using a balanced H-Tree

Requirement 10. Finally, the power consumption of the clock distribution technique must be taken into account to comply with *Requirement 5*.

In Table 4.1 the different clock distribution techniques are evaluated according to their compliance with these requirements. From this comparison, the balanced H-Tree is identified as the most suitable clock distribution technique within the sensor matrix. The detector matrix has a regular structure, making it a perfect candidate for a balanced H-tree clock distribution. The implementation of the H-Tree for a matrix of 16×16 detectors is shown in figure 4.5a, and the implementation for the entire sensor matrix is shown in figure 4.5b.

Clock Distibution Technique	Requirement		
	EMI	clock skew	power
Clock Grid	X	X	X
Unconstrained Tree	✓	X	✓
Balanced H-Tree	✓	✓	✓
Binary Tree	✓	X	✓

Table 4.1: Comparison of the different clock distribution techniques for the detector matrix according to their compliance with requirements *Requirement 3*, *Requirement 10* and *Requirement 5*

4.3. Floor-planning and Signal routing

In the following subsections the routing of all the signals, and floor-planning of the combinational logic is discussed.

4.3.1. PDK characteristics

As stated in *Requirement 6* the ASIC must be implemented using the TSMC 40nm PDK. The TSMC 40nm PDK process design kit (PDK) provides a Standard Cell Library (SCL) containing standard logic gates. Due to a non-disclosure agreement, details of the PDK cannot be shared. Instead, the PDK will be characterized by extracting the critical length, distributed RC, repeater drive strength, signal power consumption, and propagation delay as defined in appendix C. The PDK has a shrinkage factor of 0.9, which means that a feature of 1 μm in the layout will become a feature of 0.9 μm in the physical IC. From hereon all reported dimensions related to the layout dimension before the shrinkage factor are denoted as μm_{pdk}. The build-up used contains seven copper interconnect layers (denoted M1-M7).

Metal Layer	Sheet Resistance [$\frac{m\Omega}{\square}$]	Min Width [μm_{pdk}]
M1	225,0	0.07
M2	210,0	0.07
M3	210,0	0.07
M4	210,0	0.07
M5	210,0	0.07
M6	22,9	0.4
M7	5.06	225,0

Table 4.2: Available metal layers in the TSMC N40 PDK with their sheet resistance and minimum wire width

An overview of the sheet resistance and the minimum trace width of each metal layer is given in Table 4.2. The dynamic power consumption of any digital circuit depends mainly on the capacitive load, as the energy dissipated due to a transition is:

$$E = \frac{CV^2}{2} \tag{4.1}$$

As explained in Appendix C the capacitance of a wire depends on its width. As a consequence, metal layers with a larger minimum wire width will have a larger parasitic capacitance. Routing any signal that experiences large voltage swings and high switching frequency through these layers is therefore undesirable, as this negatively impacts the power consumption. Metal layers M6 and M7 should, therefore, primarily be used to route supplies or other nets that do not experience large voltage swings.

An important characteristic of a PDK is the rise time at the output of an inverter chain, as this limits the maximum length of the wire due to the effects of the transmission line, as explained in Appendix C.1.3. For the used PDK the rise time is approximately 5 ps. From Equation C.15 the maximum length for which transmission line properties do not need to be accounted for is:

$$L_{max} = ct_r = 1.5mm \tag{4.2}$$

In appendix G.3.1 the PDK is characterized. For metal layers 2 - 5 the following capacitance per unit length is found:

$$c_{wire,2-5} = 0.17348\frac{fF}{\mu m_{pdk}} \tag{4.3}$$

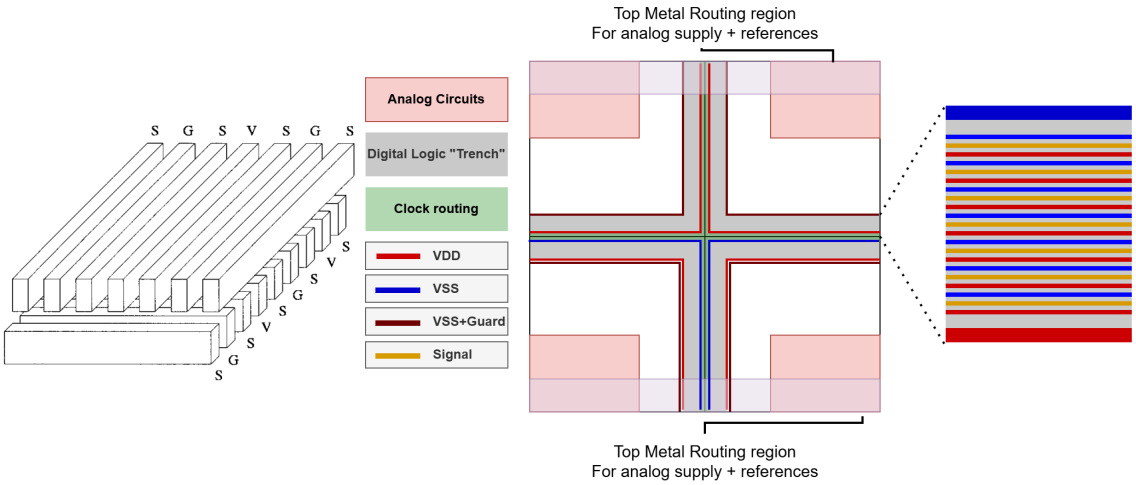
Furthermore, routing signals through metal layers 2-5 using an inverter with drive strength D8 is shown to be the most power efficient, and requires an inverter within every 234 μm. Routing signals through metal layer 6 with an inverter with drive strength D24 results in the smallest propagation delay per μm, and requires an inverter within every 740 μm.

4.3.2. Floor-planning of the Detector Matrix

During the floor planning of the detector matrix routing of the following signals must be taken into account

- Analog supplies for the FRCs.
- Analog reference for the FRCs.
- Digital supplies for the backend readout circuits, and FRCs.
- Digital outputs of the FRCs, and digital signals of the backend readout circuits.
- Clock signal.
- Reset signal.

It is desirable for the dimensions of the detector matrix to be a power of two, as this allows the detector matrix to be designed iteratively. First, a matrix of four detectors is designed, then these



(a) Dense Wire Fabric routing strategy [48] almost completely eliminates crosstalk between signal wires (S = Signal, V = VDD, G = GND)

(b) Overview of the layout of four adjacent detectors, and the shielding and supply framework provided within the digital logic trenches. Analog supplies are routed horizontally in the areas shown at the top and bottom.

Figure 4.6: Dummy detector layout in sensor matrix with framework for digital logic and signal routing.

are combined to create a matrix of eight detectors, and so on. The nearest power of two that meets *Requirement 8* is

$$N_{matrix} = 2^{\left\lceil \log_2\left(\frac{2\times 8mm}{165\mu m}\right)\right\rceil} = 128 \tag{4.4}$$

Thus, the detector matrix will be 128 by 128 detectors, resulting in a total of $n_{matrix} = 16384$ detectors. The detectors are placed so that the FRCs form an analog island. This way, there can be as much distance as possible between the analog and digital circuits. The digital circuits will be located along the detector boundaries. In order to mitigate noise injection into the analog substrate as much as possible, the digital signals and devices are surrounded by a VSS guard ring, providing a low-resistance return path for any charge injected into the P-type substrate through MOS devices and parasitic cross-coupling capacitance from interconnect wires.

A group of four detectors using this layout is shown in figure 4.6b. The area enclosed by the VSS guard ring is where all digital logic and signals will be located and will be referred to as the Digital Logic trench (DLT). In the center of the DLT, on the border of the detectors, an area is reserved for clock routing. The digital signals routed in the DLT will be located close together and will run in parallel for long distances, leading to data-dependent delays due to crosstalk (discussed in more detail in appendix C.1).

The dense wire fabric (DWF) proposed by Khatri et al. [48] is a crosstalk-free interconnect technique. DWF (shown in figure 4.6a) uses minimum-width wires routed in parallel at the minimum pitch distance. Between the signal wires, alternate shielding wires to VDD and VSS are provided. The routing direction also alternates between layers. Signal routing is achieved by adding vias to the desired locations. Although this routing method significantly increases the capacitive loads of the wires, it almost completely eliminates crosstalk. Therefore, the sensor matrix provides the shielding wires required to route digital signals similar to the DWF. Using the DWF the DLT can support the routing of a maximum of 128 signals.

The analog supply and references are routed over the analog regions (ROIC's) as much as possible. This minimizes the parasitic cross-coupling with digital signals and logic located in the DLT. In addition, analog signals are shielded using the ground of the analog supply (*VSS_AN*) as illustrated in Figure 4.7. The digital supply and the reset signal are routed over the DLT so that the noisy digital supply does not cross over the sensitive analog supply and references, as illustrated in Figure 4.8. This routing method also prevents accidentally creating ground loops that could harm system performance.

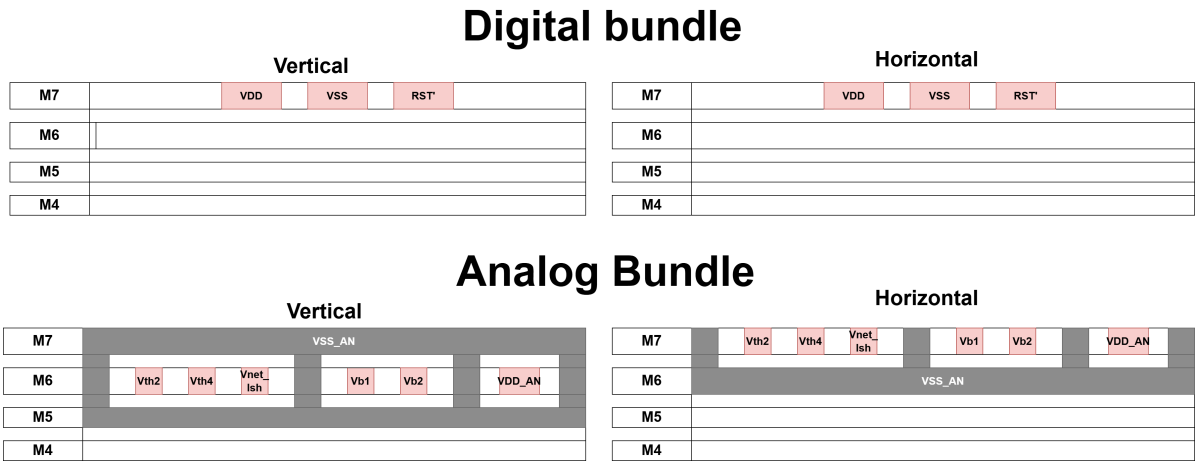


Figure 4.7: Top-Level routing structure of analog and digital supplies. The analog references and supplies are shielded using *VSS_AN*. The digital supplies and reset signal are not as sensitive and do not require shielding.

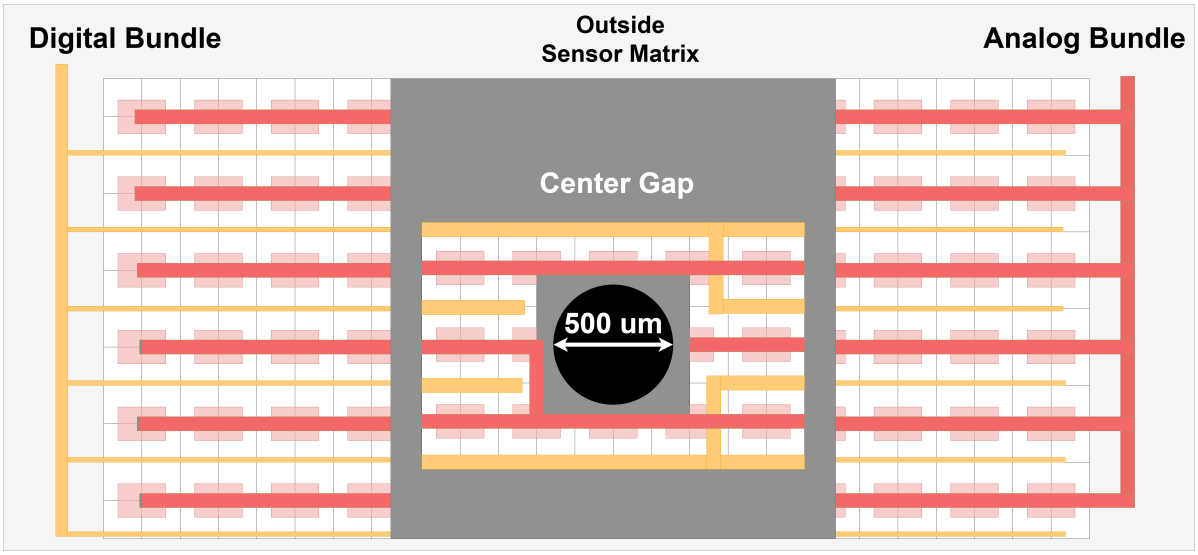


Figure 4.8: Routing of supplies, analog references and reset signal throughout the sensor matrix, and around the center gap required by *Requirement 8*

4.3.3. Routing backend readout signals

Due to the power budget given by *Requirement 5*, and the interconnect being the largest contributor to power consumption (as explained in more detail in Appendix C), an inverter with drive strength D8 is preferred for signal routing in the first pipeline segments. The critical length of the interconnect wire determined in Appendix D is slightly larger than the width and/or length of a single detector. To simplify the design, repeaters will be placed at every detector pitch ($185\ \mu m_{pdk}$).

After pipeline segment L3 the number of signals that have to be routed is relatively small, while the distance over which they have to be routed is very large. Therefore, as of pipeline segment L3 the signals will be routed through metal layer 6 using repeaters with drive strength D24

4.3.4. Routing the clock

Routing signals through M6 results in a much shorter propagation delay compared to M2-M5 at the cost of a higher power consumption. Considering that the clock signal experiences a large rail-to-rail voltage swing each clock period, routing through M6 or M7 should be avoided due to the limited power budget from *Requirement 5*. In addition, minimizing the propagation delay from the clock source to the endpoint is of little interest. Rather, minimizing the skew between the different endpoints should be the main focus as defined in *Requirement 10*.

The clock will be routed mainly through M3-M5 and will use balanced clock buffers with a drive strength of D12. Although the EDP is larger compared to other drive strengths, D12 is able to maintain shorter rise and fall times, which reduces clock jitter and short circuit power consumption. To maintain these rise and fall times, a clock buffer will be inserted at the same pitch as the detectors ($185\ \mu m_{pdk}$). To route the clock from the L4 pipeline segment to L3 metal layer 6 will be used in combination with balanced clock buffers with a drive strength of D24. Otherwise additional memory elements would have to be added to the pipeline due to the increased propagation delay of the clock signal.

4.3.5. Floor-planning of backend readout

Each pipeline segment can process its input independently of the other segments. The combinational logic for each segment can be distributed throughout the detector matrix, or the combinational logic can be placed at a central location outside the detector matrix, as illustrated in Figure 4.9. The advantages and disadvantages of each are discussed in detail in the following subsections.

Centralized Combinational logic

Placing the logic outside the detector matrix has the advantage that most of the heat dissipation occurs away from the sensitive detectors, reducing thermal noise. Furthermore, this allows the combinational logic of the adder tree to be described using an HDL (like VHDL or Verilog) and automatically synthesize the layout using a tool like Genus. The disadvantage is that this would require more chip area and increase the total power consumption, as all detector outputs need to be routed over long distances and buffered multiple times to reach the combinational logic. Furthermore, signal routing would significantly increase the design effort due to repeater placement. Using Figure D.1 and Equation D.2 with a D8 inverter as repeater, the distance a signal can travel within the sampling period of 2.5 ns is:

$$L_{wire,D8,max} = \frac{2.5E^{-9}L_{crit}}{t_{p,crit}} \approx 8.5\ mm_{pdk} \tag{4.5}$$

If All the Combinational logic of the adder tree were to be placed at the top center of the detector matrix, the longest possible distance a signal would have to travel would be one and a half time the number of rows/columns within the detector matrix:

$$L_{worst,case} = W_{det}(1.5 \times 128) = 35.2\ mm_{pdk} \tag{4.6}$$

Using Equation 4.5 the minimum number of storage elements required per signal would be:

$$k = \left\lfloor \frac{L_{worst,case}}{L_{wire,D8,max}} \right\rfloor = 5 \tag{4.7}$$

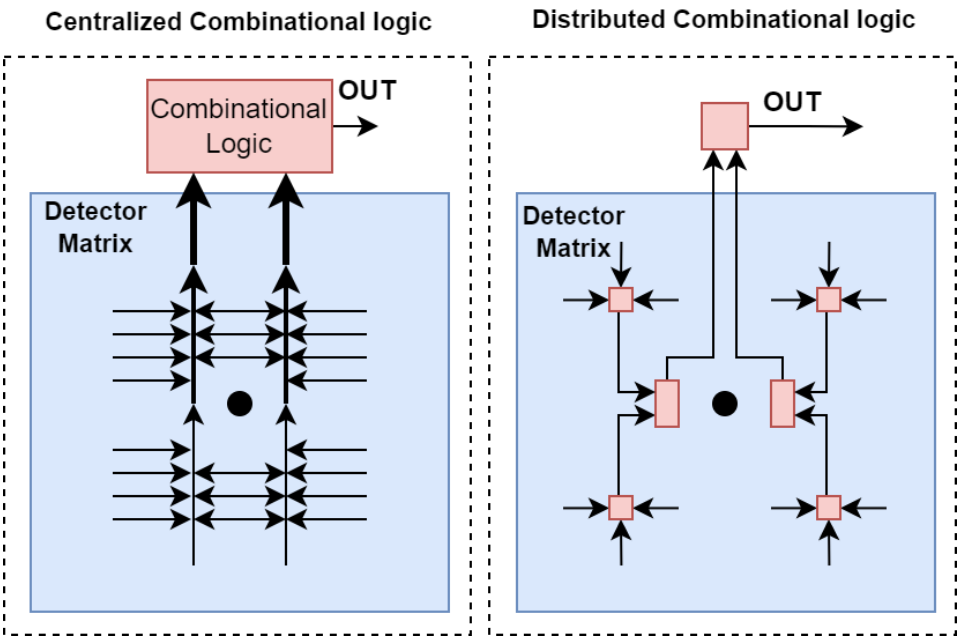


Figure 4.9: Implementation of the the pipelined adder tree using a centralized layout for the combinational logic (left) or a distributed layout for the combinational logic (right)

Therefore, every single of the 16384 detector outputs requires 5 memory elements in order to be routed to the centralized combinational logic block. Each of these memory elements increases the load of the clock. When designing for a low power application this large clock load is undesirable, therefore, using a centralized combinational logic approach is not suitable for this application.

Distributed Combinational logic
Locating the combinational logic throughout the detector matrix increases heat dissipation within the matrix and the design complexity, but reduces overall power consumption, as the number of signals that have to be routed are quickly reduced. There are two possible approaches, either adding signals as soon as possible or routing the signals to a central location within the pipeline segment and performing the addition there.

With instant addition, three same weight bits are added as soon as possible using FAs as illustrated in Figure 4.10. This has the advantage that the number of signals is always minimized, resulting in reduced power consumption due to signal propagation.

Alternatively, the addition can be centralized within the pipeline segment. The signals are routed to the combinational logic, and the outputs of the combinational logic are ready before the next clock edge so that they can be stored. This results in Just-In-Time (JIT) addition, as the resulting sum is available just before the next clock period.

Whether instant addition of JIT addition is preferred depends on the application. Instant addition reduces power consumption if many of the inputs are likely to switch. As we are dealing with a sparse matrix, the inputs are not very likely to switch. Therefore, instant addition would have no significant impact on the power consumption, while requiring more design effort. The choice is made to implement the combinational logic using a Just In Time distributed approach.

4.4. Double Data Rate Architecture

The dynamic power dissipation from the clock is given by:

$$P_D = CfV_{dd}^2 \tag{4.8}$$

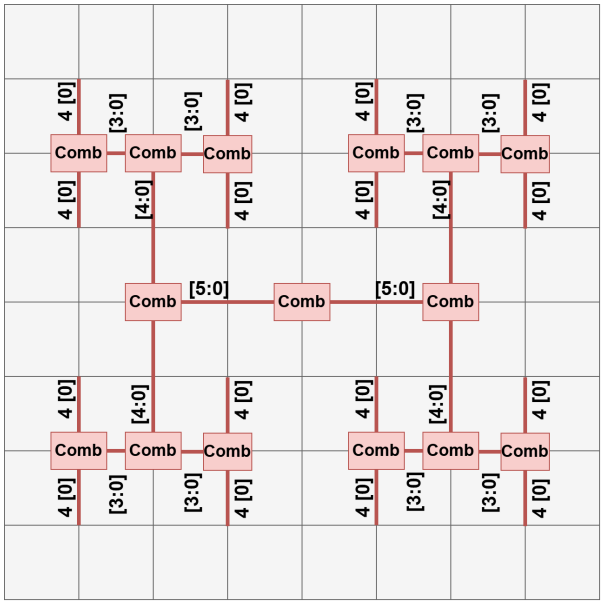


Figure 4.10: Illustration of distributed combinational logic using instant addition in a cluster of 8 x 8 detectors. The numbers between brackets indicate the number of bits of different weights within the bus, and the number in front indicate the number of instances of a bus.

where f is the clock frequency, V_{dd} is the supply voltage and the total capacitance C is the capacitive load seen by the clock. The capacitive load of the clock can be attributed to the capacitance of the pins of the clocking elements C_{pin} and the parasitic capacitance of the clock interconnect C_{wire}:

$$C = C_{wire} + C_{pin} \tag{4.9}$$

As explained in detail in AppendixC C_{pin} is much smaller than C_{wire}.Therefore, power consumption can be reduced by reducing the clock frequency from f_1 to f_2 if the following is satisfied:

$$\frac{C_{f2}}{C_{f1}} < \frac{f_1}{f_2} \tag{4.10}$$

A double data rate (DDR) architecture reduces the clock frequency by storing data at both the rising and falling clock edges. This would allow the global clock frequency to be reduced from 400 MHz to 200 MHz, while still meeting *Requirement 1*. Unfortunately, the PDK does not implement any DDR memory elements. Therefore, the dual edge-triggered D- flipflop (DETDFF) proposed by Afghahi and Yuan [49] is implemented using two latches and a multiplexer as shown in Figure 4.11. When the clock is high, **D** can propagate to **QH**, and **QL** is latched and connected to **Q** through the MUX. When the clock is low, **D** can propagate to **QL**, and **QH** is latched and connected to **Q** through the MUX. The DETDFF has the following timing constraints:

- $t_{setup} = 41\ ps$
- $t_{hold} = 37\ ps$

A single DETDFF increases the clock load by $C_{detdff} = 0.463fF$, which is a 4 fold increase compared to a DFF triggered by a single edge.

$$C_{detdff} = 4C_{dff} = 0.463\ fF \tag{4.11}$$

This load can be expressed as an equivalent length of the interconnect wire using Equation D.1:

$$L_{eq,wire} = \frac{C_{detdff}}{c_{wire,2-5}} \tag{4.12}$$

In order for the clock to reach all detectors, it has to at least cover a distance of

$$L_{clk} > n_{matrix} W_{det} \tag{4.13}$$

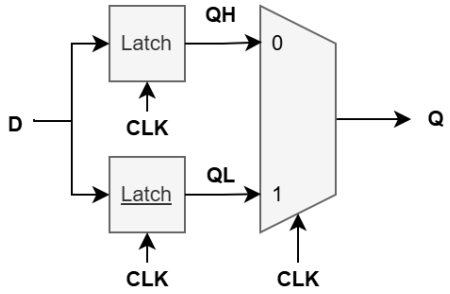


Figure 4.11: Dual Edge Triggered D-flipflop proposed by Afghahi and Yuan [49], implemented using two latches and a MUX. When the clock is high **D** can propagate to **QH**, and **QL** is latched and connected to **Q** through the MUX. When the clock is low **D** can propagate to **QL**, while **QH** is latched and connected to **Q** through the MUX.

Where n_{matrix} is the number of detectors in the detector matrix and W_{det} is the detector pitch as provided in Section 2.1. Therefore, Equation 4.10 is satisfied as long as:

$$N_{dff} < \frac{L_{clk}}{L_{eq,wire}} \tag{4.14}$$

which evaluates to:

$$N_{dff} < \frac{n_{matrix} W_{det} c_{wire,2-5}}{C_{detdff}} \tag{4.15}$$

The designed detector matrix contains a total of 16384 detectors at a pitch of 165 μm. Using equation 4.3 this equation evaluates to:

$$N_{dff} < 136822 \tag{4.16}$$

Therefore

4.5. Backend readout implementation

In section 4.1 the combinational circuits of the structure of the backend readout pipeline, and the combinational logic of the segments was designed. Using the clock distribution designed in section 4.2, the floor-planning and layout design choices made in section 4.3 and the DDR memory elements from section 4.4, the backend readout subsystem will be implemented.

The following subsections provide a detailed analysis of the timing for each pipeline segment using the setup and hold-slack (S_{su} and S_h, respectively). These give a good indication of whether the timing requirements are met at the data input of the memory elements.

- **Setup-Slack**: Indicates the time between the input of the DETDFF becoming valid and the clock edge at which these data are stored. Negative values indicate that the clock edge arrives at the DETDFF before the data input has met the setup time requirements. For the DETDFFs used in this design, the setup time is 50̃ ps.
- **Hold-Slack**: Indicates the time between the clock edge at which the data are stored and the input of the DFF becoming invalid. Negative values indicate that the data at the input becomes invalid before the hold time of the DETDFF is satisfied. For the DETDFFs used in this design, the hold time is 50̃ ps.

Using the timing analysis, the implementation of each pipeline segment will be finalized.

It is important to verify performance with the PVT variation in mind, therefore, they are performed in the fastest process corner (FF) and the slowest process corner (SS) to ensure each pipeline segment operates as expected regardless of PVT variation. To guaranty functionality throughout the PVT variation, the propagation delay of a single repeater is determined in the slowest process corner, with a supply voltage of 1.05 V at a temperature of 50 °C, and in the fastest process corner, with a supply voltage of 1.15 V at a temperature of -20 °C. To estimate the minimum and maximum path delay through the

Process corner	FF	SS
Supply Voltage	1.15	1.05
Temperature	-20 °C	50 °C
$t_{p,rep,m3,185\mu m}$	49 ps	90 ps
$t_{p,rep,m6,740\mu m}$	63 ps	132 ps
$t_{p,fa,max}$	49 ps	101 ps
$t_{p,buff}$	20 ps	40 ps
$t_{p,clk,185\mu m}$	22 ps	42 ps

Table 4.3: Timing characteristics of the main components within the readout pipeline

adder tree logic, the maximum delay through a single FA and the minimum delay through a buffer are also determined. The resulting propagation delays are shown in Table 4.3.

4.5.1. L1 - Critical adder tree implementation details

The setup-slack is calculated through:

$$S_{su} = T_{clk} - (t_{p,clk256\rightarrow clk\_frc} + t_{pmax,frc\rightarrow l1\_in} + t_{l1,com,max} + t_{su}) \tag{4.17}$$

where T_{clk} is the clock period, $t_{p,clk256\rightarrow clk\_frc}$ is the clock propagation delay to the FRC, $t_{pmax,frc\rightarrow l1\_in}$ is the maximum propagation delay from the FRC output to the input of the combinational logic of L1, $t_{l1,com,max}$ is the maximum delay in the logic path through L1 and t_{su} is the setup time of the DFF.

The hold-slack is calculated through:

$$S_h = t_{p,clk256\rightarrow clk\_frc} + t_{pmin,frc\rightarrow l1\_in} + t_{l1,com,min} - t_h \tag{4.18}$$

where $t_{pmin,frc\rightarrow l1\_in}$ is the minimum propagation delay from the FRC output to the input of the combinational logic of L1 and t_h is the hold time of the DFF.

To evaluate the timing requirements, the propagation delays are determined through simulations in both the slowest process corner with a supply voltage of 1.05 V at 50°C and the fastest process corner with a supply voltage of 1.15 V at -20°C. The maximum path delay of the L1 combinational logic arises from a signal entering the first stage and propagating through all 12 stages to the output bit with a weight of 8. The minimum path delay through the critical adder tree arises from a signal entering the first stage and propagating through 6 stages to the output bit with a weight of 1.

Table 4.4 shows the results of the delay simulation and the slack calculations. When CLK_{256} is used to trigger the DETDFFs at the output of L1, the setup-slack in the slowest corner is -878 ps. To meet the timing requirements, an additional buffer stage can be added before the combinational logic of L1. However, this significantly increases the clock load.

CLK_{256} is delayed by 1350 ps in the slowest corner. The delayed clock is referred to as CLK_{L1} Using CLK_{L1} to trigger the DETDFFs at the output of the combinational logic of L1 ensures that the timing requirements are met. The resulting pipeline segment has a margin of 413 ps. Therefore, we can conclude that the synchronous L1 pipeline segment will operate reliably, regardless of PVT variations and device mismatches. This is further illustrated in Figure 4.12. To ensure the output of L1 is synchronized to the clock tree an additional layer of DETDFFs is added which are triggered by CLK_{256}.

Process-Corner **Supply Voltage** **Temperature**	FF 1.15 V -20 °C	SS 1.05 V 50 °C
No Buffers between FRC_{out} and $L1_{out}$		
$t_{p,clk256 \rightarrow clk\_frc}$	412 ps	810 ps
$t_{pmax,frc \rightarrow l1}$	562 ps	1026 ps
$t_{pmin,frc \rightarrow l1}$	513 ps	947 ps
$t_{p,l1comb,max}$	662 ps	1493 ps
$t_{p,l1comb,min}$	189 ps	409 ps
$S_{su}(FRC_{out} \rightarrow L1_{out})[CLK_{256}]$	+814 ps	-878 ps
$S_h(FRC_{out} \rightarrow L1_{out})[CLK_{256}]$	+1113 ps	+2194 ps
Delaying CLK_{256} 1350 ps in SS corner		
$t_{p,clk256 \rightarrow clk,l1}$	700 ps	1362 ps
$S_{su}(FRC_{out} \rightarrow L1_{out})[CLK_{L1}]$	+1514 ps	+484 ps
$S_h(FRC_{out} \rightarrow L1_{out})[CLK_{L1}]$	+413 ps	+833 ps

Table 4.4: Overview of signal, clock and combinational delays associated with the synchronous pipeline segment L1

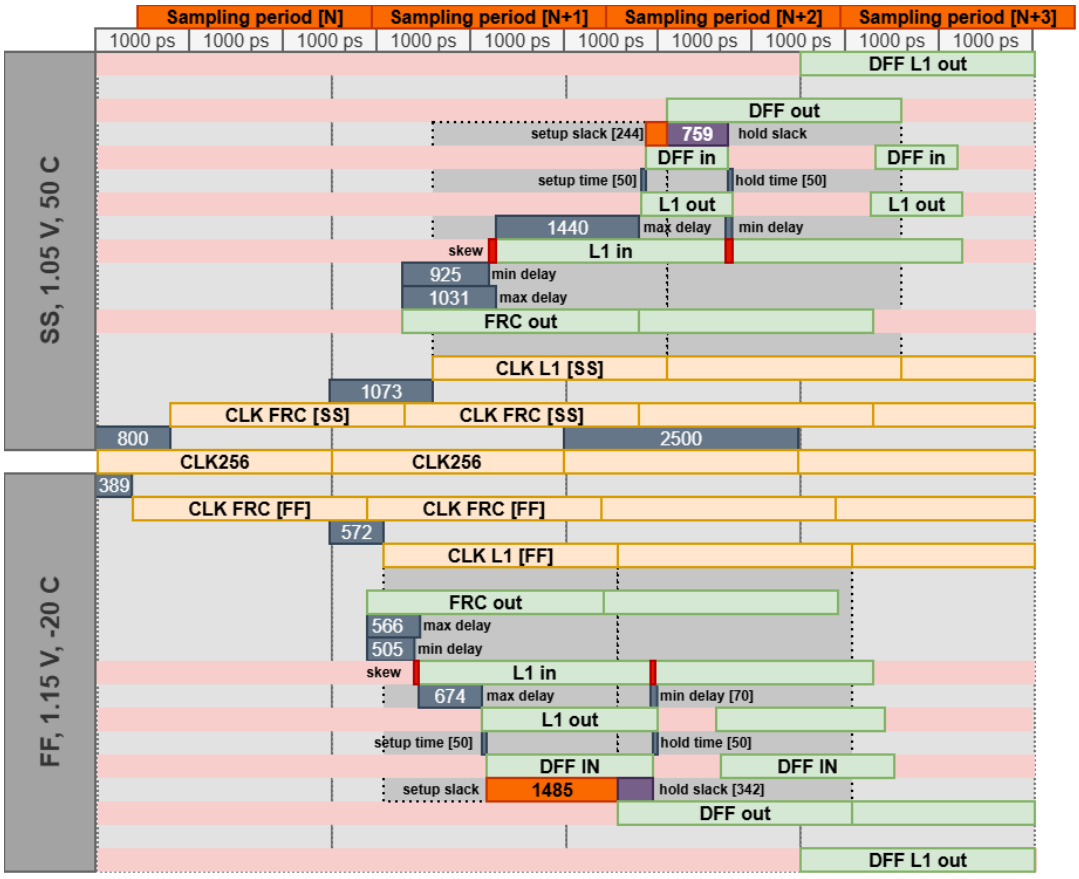


Figure 4.12: Timing diagram for synchronous critical adder tree L1 pipeline segment for the slowest corner (Top) and fastest corner (Bottom) using the timing values obtained through simulation.

4.5.2. L1 - Pulse counter implementation details

To ensure that the sampling windows of the FRC and the pulse counter overlap as much as possible, the clock signal must be delayed. Figure 4.13 shows the sampling window of the synchronous FRC, which is determined by CLK_{frc}. The propagation delay from CLK_{256} to CLK_{frc} has already been determined. The output of the asynchronous FRC has some propagation delay before reaching the input of the asynchronous L1 pipeline segment (the pulse counter). Therefore, the clock to the pulse counter must be delayed so that it matches the sampling window of the synchronous FRCs, which becomes:

$$t_{p,clk256\rightarrow clk\_pc} = t_{p,clk256\rightarrow clk\_frcc} + t_{p,\overline{frc\rightarrow l1\_in}} \tag{4.19}$$

The data flow within the pulse counter can be summarized as follows.

1. Incoming asynchronous FRC outputs are converted to pulses and merged onto a single bus to activate the shift register. The delay associated with an input being converted to a pulse, merged into a single signal, and propagating to the output of the shift register is denoted by $t_{p,sr}$.

2. The outputs of the shift register are stored using DETFFs triggered by CLK_{SR}, which is generated by the pulse counter controller. The delay from CLK_{PC} to CLK_{SR} is denoted by $t_{p,clk\_pc\rightarrow clk\_sr}$ which should be matched to $t_{p,sr}$.

3. The outputs of the DETDFFs are presented to the encoder and propagate to its output. The minimum and maximum logic path delay of the encoder is denoted by $t_{pmin,enc}$ and $t_{pmax,enc}$ respectively.

4. The encoder outputs are stored using DETDFFs triggered by CLK_{256}.

The setup-slack at the output of the pulse counter can be determined using:

$$S_{su} = T_{clk} - (t_{p,clk256\rightarrow clk\_frc} + t_{p,\overline{frc\rightarrow l1\_in}} + t_{p,clk\_pc\rightarrow clk\_sr} + t_{pmax,enc} + t_{su}) \tag{4.20}$$

The hold-slack at the output of the pulse counter can be determined using:

$$S_h = T_{clk} - (t_{p,clk256\rightarrow clk\_frc} + t_{p,\overline{frc\rightarrow l1\_in}} + t_{p,clk\_pc\rightarrow clk\_sr}) + t_{pmin,enc} - t_h \tag{4.21}$$

Table 4.5 shows the results of the delay simulation and the slack calculations for the asynchronous L1 pipeline segment. When using CLK_{256} to trigger the DETDFFs at the output of the encoder the setup-slack does not meet the requirements in the slowest corner. To fix this, the clock signal to the DETDFFs at the output is delayed by 600 ps in the slowest corner. An additional layer of DETDFFs are placed at the output and triggered by CLK_{256}. This ensures that the outputs of both the synchronous and asynchronous L1 pipeline segments are synchronized. This is further illustrated in figure 4.13.

Process-Corner **Supply Voltage** **Temperature**	FF 1.15 V -20 °C	SS 1.05 V 50 °C
CLK_{256} at output buffers		
$t_{p,clk256 \rightarrow clk\_frc}$	412 ps	810 ps
$t_{pmax,frc \rightarrow l1}$	562 ps	1026 ps
$t_{pmin,frc \rightarrow l1}$	537 ps	987 ps
$t_{pavg,frc \rightarrow l1}$	537 ps	987 ps
$t_{p,clk256 \rightarrow clk\_pc}$	933 ps	1770 ps
$t_{p,clk\_pc \rightarrow clk\_sr}$	478 ps	952 ps
$t_{p,pmax,enc}$	116 ps	272 ps
$\delta_{Tfrc-Tpc}$	26 ps	16 ps
$S_{su}(FRC_{out} \rightarrow ENC_{out})[CLK_{256}]$	+923 ps	-544 ps
$S_h(FRC_{out} \rightarrow ENC_{out})[CLK_{256}]$	+1371 ps	+2682 ps
Delaying CLK_{256} 600 ps in SS corner		
$t_{p,clk256 \rightarrow clk\_l1,pc}$	300 ps	619 ps
$S_{su}(FRC_{out} \rightarrow ENC_{out})[CLK_{L1,PC}]$	+1701 ps	+1028 ps
$S_h(FRC_{out} \rightarrow ENC_{out})[CLK_{L1,PC}]$	+594 ps	+1110 ps

Table 4.5: Overview of signal, clock and combinational delays associated with the asynchronous pipeline segment L1

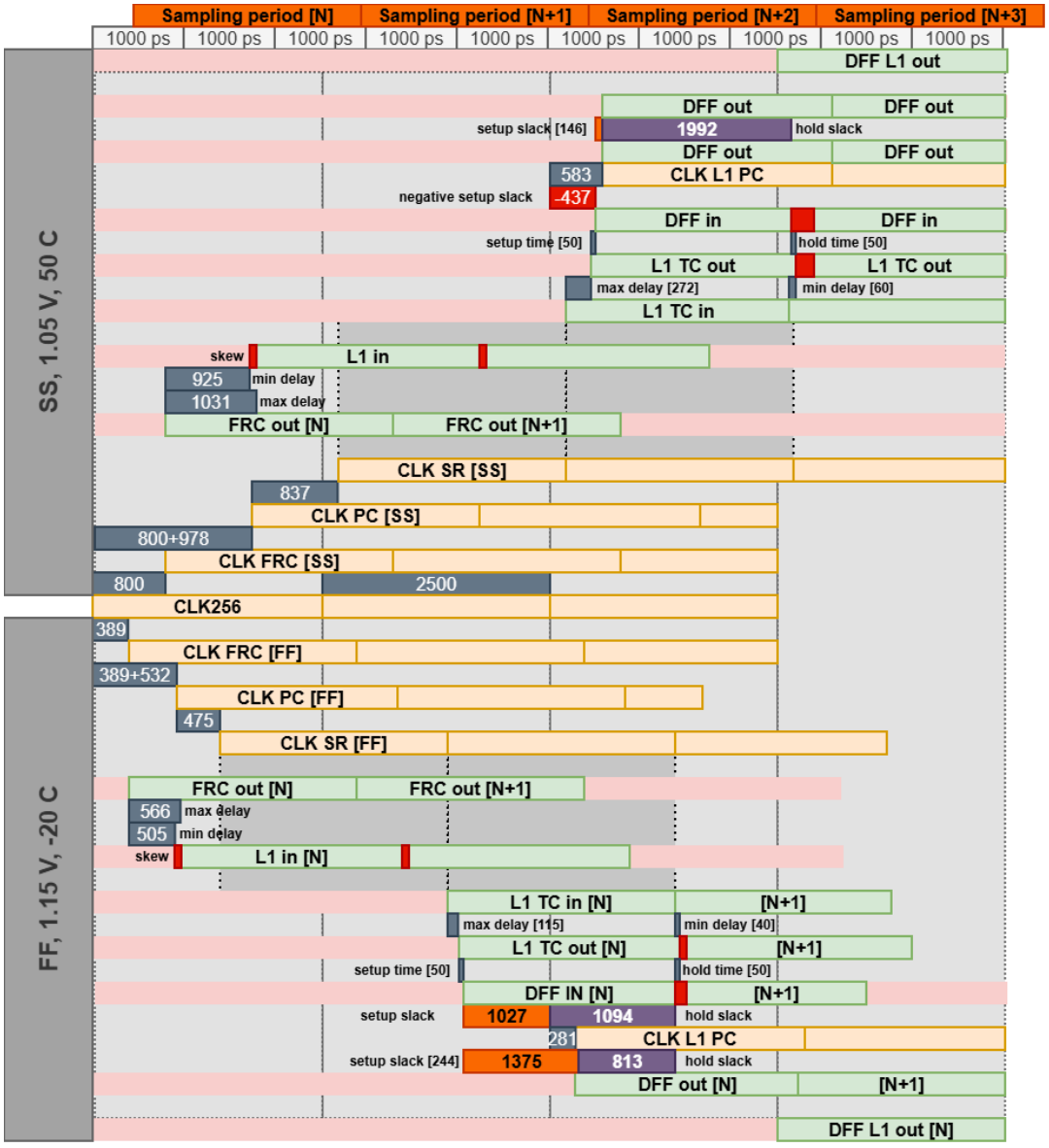


Figure 4.13: Timing diagram for asynchronous pulse counter L1 pipeline segment using the clock tapped from the clock tree entering the 16 x 16 cluster

4.5.3. L2 implementation details

The critical path delay of this pipeline segment logic arises from a signal with a weight of '1' entering the first layer and propagating all the way to the output with a weight of '16' after layer 6. The minimum path delay occurs when a signal with a weight of '1' enters the first layer and propagates through two layers to the output with a weight of '1'.

Table 4.6 shows the worst and best case propagation delays of the clock, data, and L2 combinational logic. When using the clock tapped from the H-Tree to trigger the DETDFFs at the output of the combinational logic, the setup slack is negative in the slowest corner. To solve this, the tapped clock signal is delayed by 600 ps in the slowest corner. As a result both the setup and hold slack's are positive in each corner, with a 300 ps margin.

After the DETDFFs that are triggered by the delayed clock, an additional layer of DETDFFs is added. These are triggered by the clock signal tapped from the H-Tree to ensure that the data in the pipeline are synchronized.

Variable		
Process-Corner	FF	SS
Supply Voltage	1.15 V	1.05 V
Temperature	-20 °C	50 °C
No Buffers between L1$_{out}$ and L2$_{in}$		
$t_{p,clk1024 \rightarrow clk256}$	368 ps	702 ps
$t_{p,l1 \rightarrow l2,in}$	795 ps	1433 ps
$t_{p,l2comb,max}$	334 ps	736 ps
$t_{p,l2comb,min}$	67 ps	137 ps
$S_{su}(L1_{out} \rightarrow L2_{comb,out})[CLK_{1024}]$	+954 ps	-421 ps
$S_h(L1_{out} \rightarrow L2_{comb,out})[CLK_{1024}]$	+1179 ps	+2221 ps
Delaying CLK$_{1024}$ 600 ps in SS corner		
$t_{p,clk1024 \rightarrow clk\_l2}$	363 ps	712 ps
$S_{su}(L1_{out} \rightarrow L2_{comb,out})[CLK_{L2}]$	+1318 ps	+291 ps
$S_h(L1_{out} \rightarrow L2_{comb,out})[CLK_{L2}]$	+816 ps	+1509 ps

Table 4.6: Overview of signal, clock and combinational delays associated with pipeline segment L2

4.5.4. L3 implementation details

The critical path delay of this pipeline segment logic arises from a signal with a weight of '1' entering the first layer and propagating all the way to the output with a weight of '16' after layer 7. The minimum path delay occurs when a signal with a weight of '1' enters the first layer and propagates through two layers to the output with a weight of '1'.

As signals at the output of L3 are routed through M6 using D24 inverters, the output of the D1 DETFF will not be able to drive the gates properly. An inverter chain is added to reduce the propagation delay. Table 4.7 shows the worst and best case propagation delays of the clock, data, and L3 combinational logic. When using the clock tapped from the H-Tree to trigger the DETFFs at the output of the combinational logic, the setup slack is negative in the slowest corner. To solve this, the tapped clock signal is delayed by 1600 ps in the slowest corner. As a result both the setup and hold slack's are positive in both corners, with a 470 ps margin on the hold time in the fastest corner.

After the DETDFFs that are triggered by the delayed clock, an additional layer of DETDFFs is added. These are triggered by the clock signal tapped from the H-Tree to ensure that the data in the pipeline are synchronized.

Variable		
Process-Corner	FF	SS
Supply Voltage	1.15 V	1.05 V
Temperature	-20 °C	50 °C
No Buffers between L2$_{out}$ and L3$_{in}$		
$t_{p,clk4096 \rightarrow clk1024}$	741 ps	1404 ps
$t_{p,l2 \rightarrow l3,in}$	584 ps	1153 ps
$t_{p,l3comb,max}$	374 ps	815 ps
$t_{p,l3comb,min}$	67 ps	137 ps
$S_{su}(L2_{out} \rightarrow L3_{comb,out})[CLK_{1024}]$	+751 ps	-922 ps
$S_h(L2_{out} \rightarrow L3_{comb,out})[CLK_{1024}]$	+1342 ps	+2643 ps
Delaying CLK$_{4096}$ 1600 ps in SS corner		
$t_{p,clk4096 \rightarrow clk\_l3}$	872 ps	1607 ps
$S_{su}(L2_{out} \rightarrow L3_{comb,out})[CLK_{L3}]$	+685 ps	+1623 ps
$S_h(L2_{out} \rightarrow L3_{comb,out})[CLK_{L3}]$	+1036 ps	+471 ps

Table 4.7: Overview of signal, clock and combinational delays associated with pipeline segment L3

4.5.5. L4 implementation details

The critical path delay of this pipeline segment logic arises from a signal with a weight of '1' entering the first layer and propagating all the way to the output with a weight of '16' after layer 7. The minimum path delay occurs when a signal with a weight of '1' enters the first layer and propagates through two layers to the output with a weight of '1'.

The signals are routed through M6 using D24 inverters as repeaters. Due to the large distance that the signals must traverse, buffers are added after every 8 repeaters. This amounts to a total of 5 buffers to get the signals to the L4 combinational logic. Table 4.8 shows the worst and best case propagation delays of the clock, data, and L4 combinational logic. As the buffer segments are sized equally the timing results of only one are shown

Variable		
Process-Corner	FF	SS
Supply Voltage	1.15 V	1.05 V
Temperature	-20 °C	50 °C
Timing for a single buffer segment (8 repeaters)		
$t_{p,clkbuff \rightarrow clk4096}$	351 ps	703 ps
$t_{p,l3 \rightarrow buff,in}$	599 ps	1183 ps
$S_{su}(L3_{out} \rightarrow Buff)[CLK_{buff}]$	+1500 ps	+565 ps
$S_h(L3_{out} \rightarrow Buff)[CLK_{buff}]$	900 ps	+1835 ps
L4 combinational timing		
$t_{p,l4comb,max}$	334 ps	758 ps
$t_{p,l4comb,min}$	67 ps	137 ps
$S_{su}(L4_{in} \rightarrow L4_{out})[CLK_{16384}]$	+2116 ps	+1692 ps
$S_h(L4_{in} \rightarrow L4_{out})[CLK_{16384}]$	+17 ps	+87 ps

Table 4.8: Overview of signal, clock and combinational delays associated with pipeline segment L4

4.5.6. Finalized backend readout design

The implementation details discussed in the previous subsection, along with the floor-planning and signal routing discussed in section 4.3 are combined to create the final backend readout subsystem. The complete design is shown in figure 4.14. A total of 4 pipeline segments containing combinational logic are used, along with 9 buffer segments required for signal routing. Therefore, the pipeline has a width of 13.

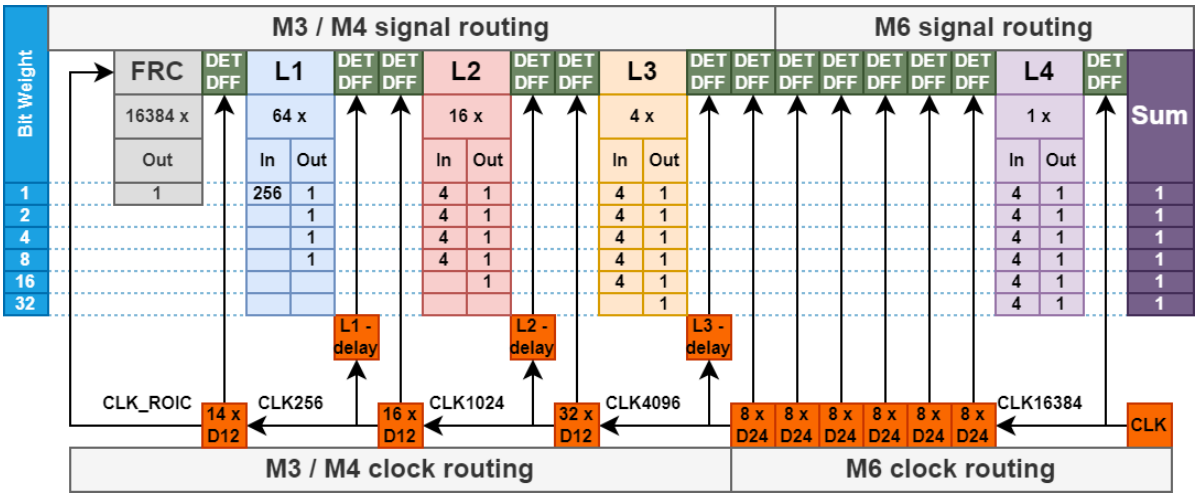


Figure 4.14: Implementation of the readout pipeline with the data-flow represented at the top. The number of inputs and outputs of different weight signals are shown under the pipeline segment. The clock distribution, and the clock signals used to trigger the DETDFFs are shown at the bottom.

5

Summary of simulation results and discussion

In chapter 3.1 the three proposed backend readout architectures, full adder tree, critical adder tree, and pulse counter, were compared against each other. This was done using the error results in figure 3.19 and the power consumption results in figure 3.20. In this chapter the results related to ASIC designed in the previous chapter will be presented. First, the model used for the FRC will be introduced. Next, the simulation results related to the system requirements presented in section 2.1.2 are presented. Following these results, the system requirements provided in section 2.1 will be discussed.

5.1. Modeling the FRC

To account for the analog signals, supplies, clock and reset signal to the FRC throughout the ASIC design, a dummy detector is created. This dummy detector has the dimensions of the detector and FRC as provided in Section 2.1. The incoming clock signal is buffered for further distribution inside the FRC. The analog references and supplies are decoupled to *VSS_AN* using 50 fF deep NWELL capacitors surrounded by a *VSS* guard-ring. A diagram of the dummy detector is provided in Figure 5.1.

In order for the dummy detector to provide a behavior similar to a synchronous detector, an input can be provided, which is stored with a DFF at the next clock edge and appears on the *EVNT* output until the next clock edge. To provide a similar behavior as that of an asynchronous detector, the DFF is removed, allowing the input to appear directly on the *EVNT* output. The current implementation of the FRC does not support this functionality. However, simply removing the DFF from the FRC produces an asynchronous output signal.

5.2. Results

In this section, the clock-distribution subsystem results will be presented, starting with the power consumptions and then moving on to the performance. Next, the power consumption of the backend readout ASIC will be presented followed by its performance. Finally, the results showing the noise the ASIC introduces on the analog supplies and reference of the FRC will be presented.

5.2.1. Clock Distribution Power consumption

The clock distribution is a very large subsystem and is fully responsible for ensuring *Requirement 10* is met, and partially responsible for ensuring *Requirement 5* is met. The clock distribution from a L1 pipeline segment to the detectors is determined through simulation of the layout. However, determining the power consumption for the full layout of the clock distribution would be impractical due to the simulation time that would be required. Instead, the post layout parasitics are extracted only for the core components:

- **HT_CLK:** Repeater section of the clock distribution that runs the length of a single detector, routed through M5.

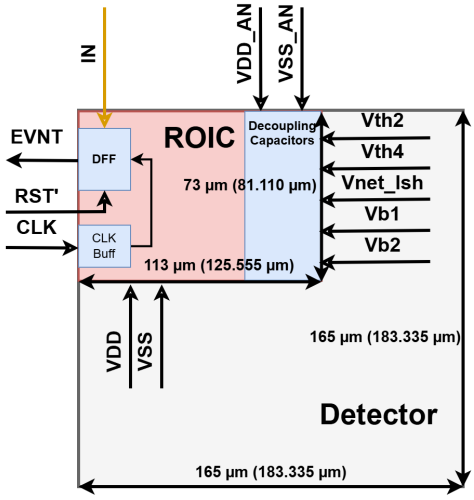


Figure 5.1: Layout of a dummy detector using the dimensions provided in Section 2.1 and the layout dimensions accounting for the shrinkage factor between brackets. The analog references, and supply, are decoupled to *VSS_AN*. Any input provided to the dummy detector is stored at the next clock edge, and then provided on the *EVNT* output during the next sample period.

Segment	**Segments in sensor**	**Memory per segment**	**Memory total**
L1-Pulse Counter	60	23	1380
L1-Critical Adder tree	4	264	1056
L2	16	10	160
L3	4	49	196
L4	1	6	6
Sensor			2798

Table 5.1: Total number of memory elements used within the sensor backend readout subsystem

- **HT_CLK_BRANCH:** Branch section of the clock distribution that branches the incoming clock signal into two separate clock paths routed through M5.
- **HT_CLK_M6:** Repeater section of the clock distribution that runs the length of a single detector, routed through M6.
- **HT_CLK_BRANCH_M6:** Branch section of the clock distribution that branches the incoming clock signal into two separate clock paths routed through M6.
- **DETDFF_D1:** Dual edge triggered d-flipflop with a drive strength of D1.
- **DFF_D1:** d-flipflop with a drive strength of D1

The clock distribution subsystem is implemented using the first four of these core components. The total number of memory elements within the backend readout subsystem presented in table 5.1 is determined from figure 4.14. The clock distribution from the global clock to the inputs of the L1 clusters is simulated at 200 MHz and 400 MHz. The resulting RMS power consumption is shown in table 5.2. Furthermore, the power consumption resulting from the clock driving a single DFF and DETDFF is also determined, and multiplied by the number of memory elements within the entire backend readout subsystem.

The clock distribution from L1 down to the individual detectors is simulated using the parasitics extracted from the layout of L1, as this part of the clock distribution depends on the readout architecture which is used. From these simulations the resulting power consumption was 58 mW.

Clock Frequency [MHz]	200	400
Component	**Power [W]**	
DFF	-	0.013
DETDFF	0.021	-
Global clock → L1	0.425	0.603
Total	0.446	0.616

Table 5.2: RMS power consumption of the entire clock distribution H-Tree, and the power usage for driving the normal and DDR memory elements.

5.2.2. Clock Distribution Performance

Determining the skew of the clock distribution through simulation of the entire subsystem is impractical. Due to the size of the subsystem, a single simulation would take multiple days. Instead, the skew is obtained for each individual section of the clock distribution along a single branch. These sections are as follows.

- The clock branch from the input of a group of 16 detectors to a single detector.
- The clock branch from the input of a group of 64 detectors to the input of a group of 16 detectors.
- The clock branch from the input of a group of 256 detectors to the input of a group of 64 detectors.
- The clock branch from the input of a group of 1024 detectors to the input of a group of 256 detectors.
- The clock branch from the input of a group of 4096 detectors to the input of a group of 1024 detectors.
- The clock branch from the input of a group of 8192 detectors to the input of a group of 4096 detectors.
- The clock branch from the global clock to the input of a group of 8192 detectors.

At the inputs of these sections, a clock signal is presented and its propagation delay is measured using Monte Carlo simulations with local device mismatch. The standard deviation of the results is directly related to the clock skew which can be expected due to local device mismatch. However, the resulting skew does not account for mismatches in signal routing. Table 5.3 provides the simulation results for the FF, TT, SS process corners. To obtain the standard deviation of the propagation delay of the complete clock distribution, the standard deviations of the individual sections are added together.

Within the FF corner, a standard deviation σ_{ff} of 11.3 ps is seen. The probability that the propagation delay ρ_{ff} is within the interval $[-50; 50]$ (i.e. a maximum skew of 100 ps as defined in *Requirement 10*) can be determined using the z-score.

$$P\left(\frac{-50}{\sigma_{ff}} \leq \rho_{ff} \leq \frac{50}{\sigma_{ff}}\right) = P\left(-4.42\sigma_{ff} \leq \rho_{ff} \leq 4.42\sigma_{ff}\right) \approx 100\% \tag{5.1}$$

Within the TT corner, a standard deviation σ_{tt} of 16.3 ps is seen. The probability that the propagation delay ρ_{tt} is within the interval $[-50; 50]$ is:

$$P\left(\frac{-50}{\sigma_{tt}} \leq \rho_{tt} \leq \frac{50}{\sigma_{tt}}\right) = P\left(-3.07\sigma_{tt} \leq \rho_{tt} \leq 3.07\sigma_{tt}\right) \approx 99.8\% \tag{5.2}$$

Within the SS corner, a standard deviation σ_{ss} of 28.55 ps is seen. The probability that the propagation delay ρ_{ss} is within the interval $[-50; 50]$ is:

$$P\left(\frac{-50}{\sigma_{ss}} \leq \rho_{ss} \leq \frac{50}{\sigma_{ss}}\right) = P\left(-1.75\sigma_{ss} \leq \rho_{ss} \leq 1.75\sigma_{ss}\right) \approx 91.8\% \tag{5.3}$$

Section		σ (ps)		
In	**Out**	**FF**	**TT**	**SS**
16	1	0,714	0,956	1,398
64	16	0,895	1,184	1,912
256	64	1,334	1,812	2,790
1024	256	1,395	2,327	5,216
4096	1024	2,201	3,090	6,729
8192	4096	1,748	2,571	2,794
Global Clock	8192	3,017	4,385	7,711
Total		**11,3**	**16,33**	**28,55**

Table 5.3: Standard deviation of the propagation delay through individual sections of the clock distribution obtained through Monte Carlo simulations with local device mismatch.

Symbol	Repeater	Distance [μm]	Power (mW)
P_{D8}	D8	165	0.0214
P_{D24}	D24	660	0.1129

Table 5.4: Power draw of a single repeater

5.2.3. ASIC Power consumption

Simulating the entire ASIC to measure the power consumption would be impractical, as the simulation would take extremely long to complete. Therefore, the power consumption of the individual pipeline segments will be determined, along with the power consumption of the signal propagation and the clock distribution.

Within the design, two different repeaters are used. That is, repeaters used for signals routed within metal layers 2-5 with a drive strength of D8, and repeaters used for signals routed within metal layer 6 with a drive strength of D24. Figure 4.14 provides an overview of the complete signal path from the detector to the final result. The power used by a single repeater to propagate one rising edge and one falling edge (i.e. a single event) is determined through simulation. The results are presented in table 5.4. The RMS power resulting from the signal propagation is determined using the activity α (i.e. how many signals are expected to turn on). Given the number of repeaters used within the signal path n, the rms power P_{sig} can then be estimated using:

$$P_{sig} = n\alpha P_{D8} \tag{5.4}$$

or

$$P_{sig} = n\alpha P_{D24} \tag{5.5}$$

Within each L1 pipeline segment, on average λ events will occur. Therefore, the activity α is equal to λ. The outputs of the L1-L4 pipeline segments represent a binary signal. Therefore, the activity is estimated through:

$$\alpha = \begin{cases} 1; & 0 < \lambda < 1 \\ 2; & 1 < \lambda < 2 \\ 3; & 2 < \lambda < 4 \\ 4; & 4 < \lambda < 8 \\ \dots \end{cases} \tag{5.6}$$

This can also be written as:

$$\alpha = \lceil log_2(\lambda + 1) \rceil \tag{5.7}$$

Table 5.5 provides an overview of the repeater types used, the values of n, and the values of α for the different pipeline segments.

The rms power of the combinational logic of each pipeline segments is determined by interpolating

Pipeline Segment	From	To	Repeater type	Amount (n)	Activity (α)
L1	Detector	L1-input	D8	14	λ_{L1}
L2	L1-output	L2-input	D8	16	$\sum_{i=0}^{4}\lceil log_2(\lambda_{L1,i}+1)\rceil$
L3	L2-output	L3-input	D24	8	$\sum_{i=0}^{4}\lceil log_2(\lambda_{L2,i}+1)\rceil$
L4	L3-output	L4-input	D24	48	$\sum_{i=0}^{4}\lceil log_2(\lambda_{L2,i}+1)\rceil$

Table 5.5: Overview of parameters for estimating signal propagation rms power within each pipeline segment.

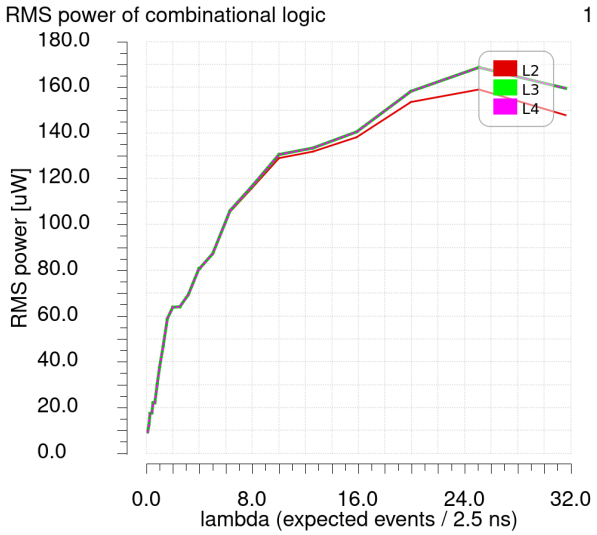


Figure 5.2: RMS power of the L2, L3 and L4 combinational logic based on the event expectation within the covered area.

the simulation results to the value of λ of the specific pipeline segment. The rms power for the combinational logic of the L1 pipeline segment was presented earlier in figure 3.20. The rms power for the combinational logic of pipeline segments L2-L4 is presented in figure 5.2.
The expected event rates λ from figure 4.3b are used to determine the rms power for the signal propagation and combinational logic of the different pipeline segments. Furthermore, for pipeline segment L1 the rms power for the clock distribution from L1 to the individual detectors is also taken into account for the critical adder tree architectures. This was found to be 58 mW in section 5.2.1.

The resulting rms power estimates for pipeline segments L1-L3 are shown in figures 5.3, 5.4a and 5.4b respectively. An overview of the total power consumption is provided in figure 5.5. When using only synchronous critical adder trees for the readout at this level, the power consumption is 4.3 W. With the proposed architecture of the L1 using pulse counters together with critical adder trees, the power consumption drops down to just 0.9 W.

5.2.4. Backend readout error rate

At the end of chapter 3 the error rates of the different backend readout architectures were presented in figure 3.19. In chapter 4. These results, along with the expected event rates in figures 4.3a and 4.3b, were used to design the backend readout subsystem for the ASIC. To determine the error rate of the entire backend readout system these same figures are used.

The critical adder trees within the L1-L4 pipeline segments were designed so that the error rate is approximately zero. Therefore, it can be assumed that the largest contributor to the error rate will be the pulse counters. Using the event expectations in figures 4.3b and 4.3a the error rates of the L1 pulse counter pipeline segments are determined by interpolating the results in figure 3.19. It must be noted that the simulation results only have data for 1000 sampling periods. Therefore, for lower values of λ the interpolated results may not be accurate. For that reason, the calculated error rates calculated using equation 3.44 have also been included.5.6. The total error rate of the system is determined by normalizing the error rates of each L1 pipeline segment according to their event expectation, and

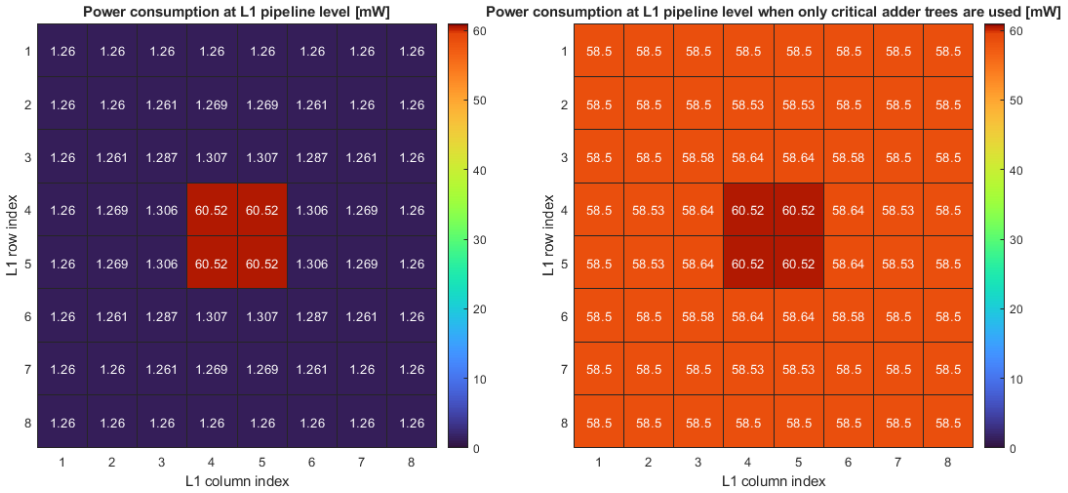
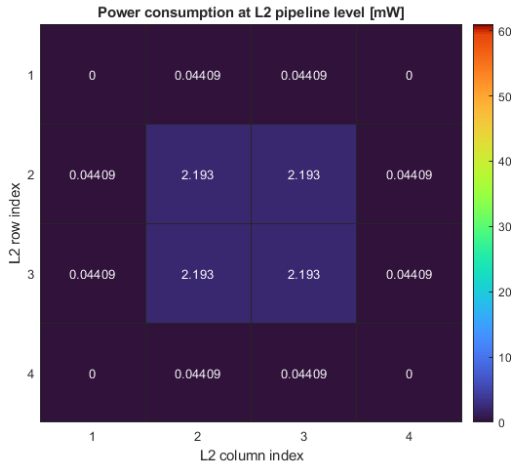
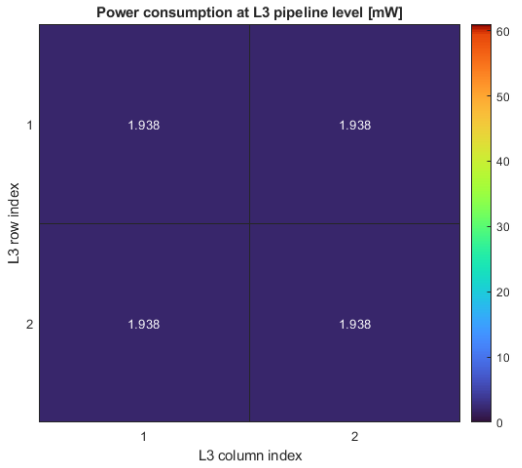


Figure 5.3: RMS power consumption estimation for the L1 pipeline segments using the proposed architecture with pulse counters and critical adder trees (left) and using only critical adder trees (right). The results include the rms power for the clock distribution within the pipeline segment, the signal propagation from the detector level up to the inputs of the L1 pipeline segments, and the combinational logic.



(a) RMS power estimation for the L2 pipeline segments. The results include the rms power for the signal propagation from the L1 outputs up to the inputs L2, and the combinational logic.



(b) RMS power estimation for the L3 pipeline segments. The results include the rms power for the signal propagation from the L2 outputs up to the inputs L3, and the combinational logic.

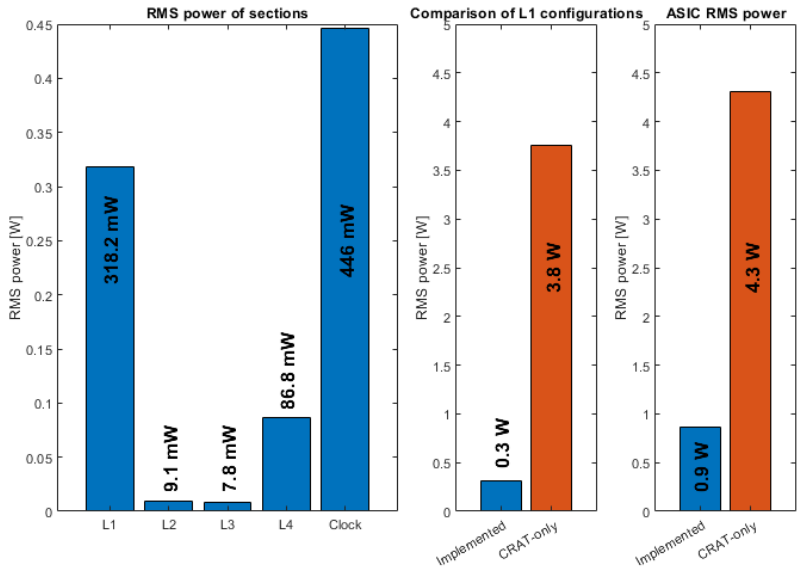


Figure 5.5: Power consumption of the ASIC separated into the different components (left), and a comparison of the power consumption at the L1 pipeline segment level using the proposed approach with asynchronous pulse counter and purely synchronous critical adder trees (right)

adding the normalized values together:

$$\varepsilon_{total} = \sum_{i=0}^{64} \frac{\varepsilon_i \lambda_i}{\sum_{j=0}^{64} \lambda_j} \tag{5.8}$$

Resulting in the following total error rates at 15 mm and 5 mm respectively:

$$\varepsilon_{15,calc} = 232 \tag{5.9}$$

$$\varepsilon_{5,calc} = 0.01 \tag{5.10}$$

5.2.5. Parasitic coupling of digital and analog signals

Within the backend readout subsystem the signals are routed inside the detector matrix. Furthermore, the combinational logic is also placed inside the detector matrix. In section 2.1.1 the FRC was introduced, showing the analog reference required for its operation. These analog supplies and references are also routed within the detector matrix. The noise introduced through parasitic capacitances between the digital and analog signals affects the performance of the FRCs.

In section 4.1 the layout of a detector matrix for pipeline segment L1 was designed, including the clock routing, FRCs, supplies and backend signal routing. The parasitics of the layout were extracted with probes placed at the decoupling capacitors of the analog supplies and references of the detectors in the center of the matrix. The L1 pipeline segments of the synchronous critical adder tree introduce the most noise on the analog signals, as the detector outputs switch simultaneously. The resulting noise of all 256 detector outputs switching simultaneously (the worst case scenario) is measured through simulation. The voltage variation observed at the probes was less than 1 nV.

5.3. Discussion

In this section, the requirements presented in section 2.1 will be evaluated and discussed.

5.3.1. Readout

Requirement 1 stated that the backend readout must count the total number of events for each sampling period of 2.5 ns. *Requirement 2* stated that this readout must be performed continuously. Figure 4.14

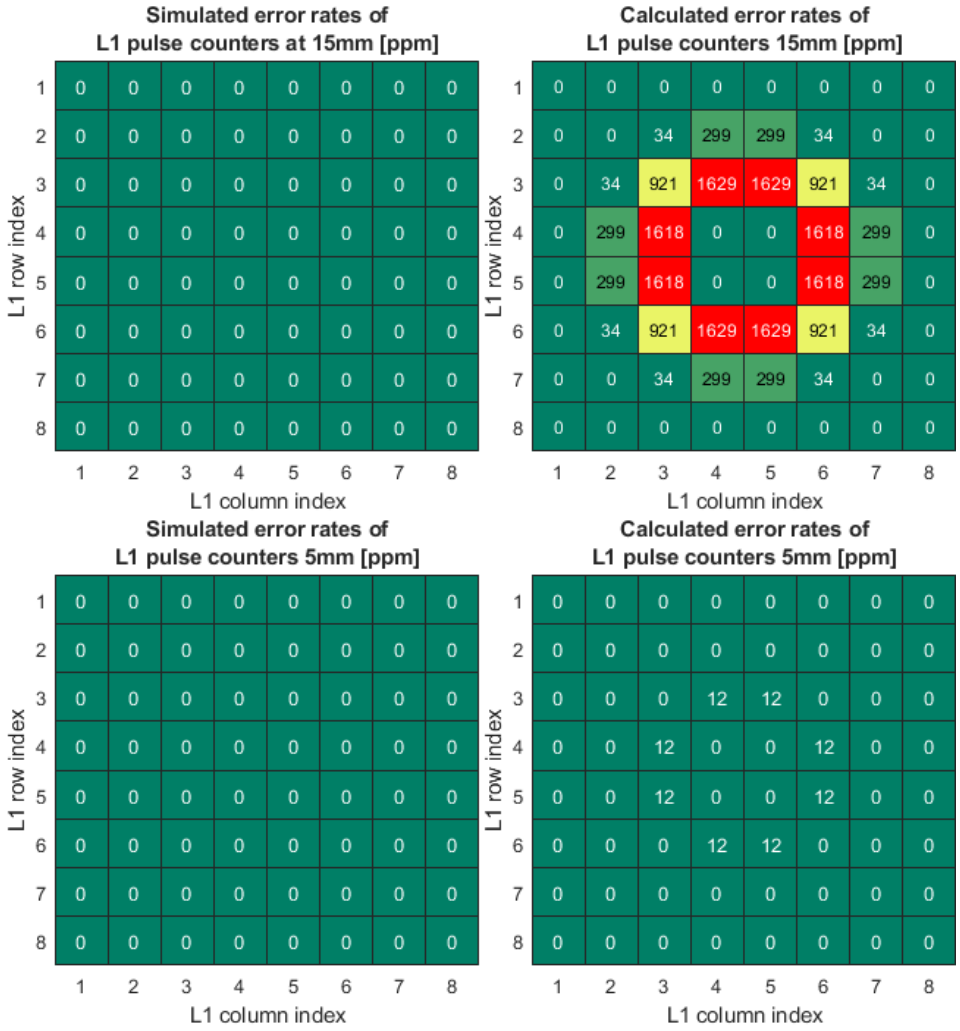


Figure 5.6: Error rates of the L1 pulse counter pipeline segments obtained through calculations (left) and interpolation of simulation data (right). The top row shows the error rates with the imaging surface at a distance of 15 mm of the sensor (zoomed out), and the bottom row shows the error rates with the imaging surface at a distance of 5 mm (zoomed in).

shows the backend readout architecture pipeline. As illustrated in chapter 2 a pipelined design reduces the critical path delay by distributing the operations over multiple clock cycles. The entire pipeline is divided into 13 sections. Therefore, the readout will be delayed by 13 sampling periods. However, the readout can be performed continuously for every single sampling period of 2.5 ns. Therefore, the proposed system meets both *Requirement 1* and *Requirement 2*.

5.3.2. Analog noise

Requirement 3 stated that the rms voltage of the noise in the analog references introduced by the backend readout must be less than 10 mV when measured at the FRC input. The implemented backend readout circuit has signals routed and combinational logic laid out within the detector matrix. The analog supplies and reference voltages required for the operation of the FRC are also routed within the detector matrix. For the operation of the FRC it is important that the analog supplies and references remain stable. To prevent parasitic capacitive coupling of the digital and analog signals, the analog signals have been routed inside a shield, and separated from the digital supplies as shown in figures 4.7 and 4.8. Finally, at each FRC the analog supplies and references are decoupled using a 50 fF capacitor.

The observed voltage variation from the simulation presented in section 5.2.5 was less than 1 nV, which illustrates the effectiveness of shielding the analog signals, in combination with the decoupling capacitors at each FRC. Therefore, it is safe to conclude that the system meets *Requirement 3*. However, the noise injected through the bulk is not taken into account by the simulation and can only be verified through physical measurements.

5.3.3. Error Rate

In order to guarantee good image quality *Requirement 4* stated that the error rate introduced by the backend readout must be less than 1000 ppm. In figure 5.6 the error rates of the backend readout circuit were presented for an imaging surface at a distance of 15 mm and 5 mm. When the error rate is based on the simulation results, none of the pulse counters are expected to produce errors. However, looking at the results calculated using equation 3.44, eight of the L1 pulse counters exceed the allowed error rate for the expected event rates with the imaging surface at a distance of 15 mm from the sensor from figure 4.3b. For the expected event rate with the imaging surface at a distance of 5 mm, all pulse counters fall within the allowable error rate.
Looking at the total error rates determined using equation 5.8, both imaging distances fall well within the 1000 ppm range with 232 and 0.01 ppm for 15 mm and 5 mm, respectively. Therefore, when looking at the entire backend readout subsystem *Requirement 4* has been met.

5.3.4. Power consumption

Requirement 5 stated that the digital supply of the ASIC must not draw more than 5 W. In chapter 4 the clock load of the DETDFF was also determined using the PDK data sheet, leading to inequality 4.10, and consequently the hypothesis that a DDR architecture would reduce the power consumption of the system. In section 5.2.1 the rms power of the clock distribution was determined for a normal clock architecture and DDR. The results presented in table 5.2 show that DDR memory elements with a clock frequency of 200 MHz resulted in a power consumption of 0.446 W, while using normal memory elements with a clock frequency of 400 MHz resulted in a power consumption of 0.616 W. Therefore, using a DDR based architecture reduces the power consumption by 27.6% and thereby proving that the hypothesis based on equation 4.10 is in fact correct.

In section 5.2.3 the power consumption of the backend readout, and clock distribution to the detector matrix were estimated, as simulating these entire systems would have been impractical. The results presented in figure 5.5 show that the power consumption of the backend readout system implemented with a DDR clocking architecture is estimated to be 0.9 W. This leaves 4.1 W of headroom for the remaining subsystems, namely the memory, configuration registers, and IO. Furthermore, the results illustrated how the asynchronous readout architecture was able to reduce the power consumption from 4.3 to 0.9 W

5.3.5. Clock performance

Requirement 10 stated that the global clock must be distributed such that the resulting clock skew is within 100 ps in the detector matrix. In section 5.2.2 the clock skew resulting from local device mismatches was determined using a Monte Carlo simulation. These results were used to estimate the yields within the FF, TT and SS process corners, which were 100% 99.8% and 91.8%, respectively. However, these results do not account for mismatches between the different branches of the clock tree within the layout. Therefore, when designing the final ASIC, special care must be taken to accurately match the parasitic capacitances of each clock tree branch.

5.3.6. Layout

Requirement 8 stated that the detector matrix must cover an area with a diameter of at least 16 mm. The proposed sensor matrix contains 128 by 128 detectors. As presented in section 2.1.1 each detector is 165 $\times$ 165 μm. Therefore, the proposed system has a sensor matrix covering an area with a diameter of 21.12 mm, and therefore meets *Requirement 8*. The main reason the sensor matrix was designed to be larger than required was so that the clock could be nicely distributed using an H-tree.

Requirement 7 stated that the entire ASIC must be implemented within an area of 25 mm $\times$ 25 mm (625 mm^2). The detector matrix and the largest part of the backend readout are contained within an area of 446 mm^2. The combinational logic of L4 occupies less than 0.05 mm^2, this leaves an area of roughly 179 mm^2 for the memory, the configurations registers and the IO.

6

Conclusion

The goal of this thesis was to design an ASIC that performs the backend readout of an event counting based electron microscope as proposed in US Patent 0123152A1 [4]. Such a sensor would allow the use of low energy electrons within the primary beam, leading to an increase in resolution. The proposed sensor consisted of a large matrix of detectors, each capable of detecting a single electron. Using the frontend readout circuit proposed by Disi, Zaki and Nihtianov [9] this could be done at a sample rate of 400 MHz. The backend readout of this detector matrix consisted of adding the outputs of all detectors together. A complete list of ASIC requirements was provided in section 2.1.2.

While researching readout architectures of existing imaging sensors in section 2.2 we found that there exists no prior work similar to this ASIC. Consequently, various readout architecture concepts were explored for this ASIC and evaluated based on the list of requirements. From the proposed readout architectures, the full adder tree, the critical adder tree, and the pulse counter were found to have the most potential. The first two were based on a Wallace tree, whereas the latter was based on asynchronously counting the events. These three readout architectures were designed and implemented for a small matrix containing 16×16 detectors.

The simulation results illustrated how the critical adder tree performed identically to the full adder tree, with a slightly lower power consumption. The pulse counter was based on asynchronously counting the events. Equation 3.43 was derived to estimate the error probability, which was shown to accurately predict the actual error probability. Furthermore, due to the asynchronous nature of the pulse counter, the clock did not have to be routed to each individual detector, reducing the power consumption by 58 mW. However, the pulse counter architecture performed reliably only when the expected event rate at its inputs was low.

The final sensor was designed as a matrix of 128×128 detectors. The complete backend readout subsystem was designed as a pipeline using DDR memory elements. Each pipeline segment computed the sum of the underlying pipeline segments. A total of four pipeline segments for the addition were required, and 9 buffer segments to route the signals. For the design of the first pipeline segment, a combination of critical adder trees and pulse counters were used based on the expected event rate of the sensor matrix within their area. This choice was made to reduce the power required for the clock distribution. These first pipeline segments produced the sum of 256 detectors. The second, third and fourth pipeline segments produced the sum of 1024, 4096 and 16384, respectively. The clock was distributed throughout the detector matrix using a balanced H-tree, and the analog reference voltages for the FRC were shielded and routed separately from the digital supplies and signals.

The simulation results illustrated how the use of asynchronous readout architectures can significantly reduce the power consumption of the system. Furthermore, the simulation results showed that the proposed system met all requirements provided in section 2.1.2. Figure 6.1 provides a clear overview of the subsystems that have been implemented by this work and the subsystems that need to be implemented in the future. A big limitation in the simulations was the time required to run them. Initially, the simulations were attempted on a complete sensor matrix, however, these simulations ran into memory

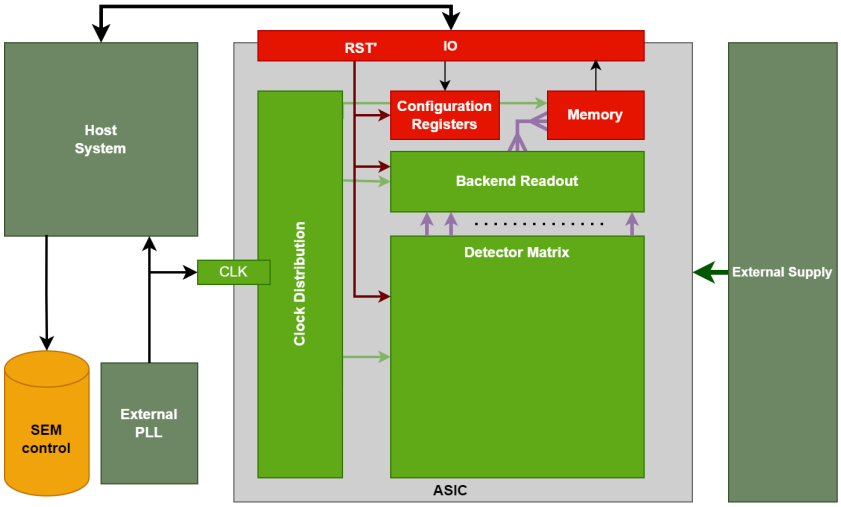


Figure 6.1: Overview of the subsystems implemented by this work (green), and the subsystems that have not been implemented (red)

issues resulting in failures. Additional limitations can be attributed to the available data related to the incidence patterns of electrons on a SEM sensor. Although this was solved by creating the model in appendix B, it required a lot of additional effort.

6.1. Future Work

The main focus of future work should be to implement the memory, configuration register and IO subsystems in figure 6.1 in order to obtain a functional ASIC. However, future work should also focus on improving the proposed system by updating the backend readout to support a segmented sensor, improving the pulse counter, and exploring alternative asynchronous readout architectures.

6.1.1. Segmented electron counting sensor

Section 2.2.3 briefly discussed segmented SEM sensors, and their ability to improve image quality through different illumination angles, as illustrated in figure 2.7. The system proposed in this thesis provides a single readout for the entire sensor. Future work should look at the possibility of creating a programmable segmented sensor from the detector array, allowing imaging from different illumination angles. This could be achieved by routing the results of the L1 pipeline segments directly to the final pipeline segment outside the detector matrix. The final pipeline segment can be configured in different operating modes by the host system and combine the outputs of the L1 pipeline segments based on the operating mode.

6.1.2. Pulse counter improvements

The performance of the pulse counter can be improved using a different technology. Smaller devices would allow for even shorter pulses, reducing the probability of an overlap error. Smaller devices would also reduce power consumption. Furthermore, by halving the number of inputs of the asynchronous readout, the number of levels within the merger is reduced by one. Firstly, this reduced the pulse width at the output of the merger and consequently the probability of an overlap error given by 3.40. Secondly, with half the number of inputs, the expected event rate λ is also reduced, leading to a further decrease in the probability of an overlap error.

6.1.3. Alternative asynchronous readout architectures

The approach chosen in this work is to convert rising edges into pulses and merge the pulses to a single signal using NOR and NAND gates. However, there are different approaches that could be explored. In section 2.2.4 the readout of multiple SPADs through a single TDC was discussed. Patinawa [30] proposed the use of an XOR tree with toggle flip flops at the input to merge multiple SPAD outputs into a single signal. Toggle flip flops are essentially a DFF with its inverted output connected to its input, and

the incoming signal connected to the CLK input, as a result its output is inverted for every rising edge at the input. At the output of the XOR tree the exact same behavior is seen, a signal that is inverted every time a rising edge is seen at the input. The exact same technique can be applied to the outputs of the detectors. This would no longer require the rising edges to be converted into pulses, saving a lot of energy. The modified shift register in the asynchronous readout architecture would have to be replaced by a DDR shift register. This could potentially lead to a significant decrease in the error rate from the asynchronous readout architecture, as events occurring close to each other could be detected more reliably, only being limited by setup and hold time of the memory elements in the DDR shift register.

A

Scanning Electron Microscopes

Scanning electron microscopes (SEM) use electrons to analyze a specimen. Using electrostatic lenses, apertures and electromagnetic coils, electrons are focused to a beam of the desired diameter on the specimen [1]. These focusing elements are also used to sequentially position the electron beam at discrete locations throughout the specimen, allowing surface information to be obtained in a scanning fashion.

Electrons from the electron beam, also known as primary electrons (PE), interact with the specimen in distinct ways and are scattered due to interactions with atoms within the specimen. There are two main ways in which this scattering occurs with the specimen, namely elastic and inelastic scattering [1]. These two interactions result in back-scattered electrons (BSE) and secondary electrons (SE), respectively.

A.1. Back-scattered electrons

Elastic scattering occurs due to the electrostatic interaction between atoms and an incoming electron, resulting in a deflection of the incoming electron [1]. This does not cause a noticeable change in the energy of the electron and therefore will maintain a large fraction of its original energy. If a PE has enough elastic interactions with the specimen atoms it can be emitted from the specimen as a BSE as illustrated in figure A.3. The energy of BSE's also depends strongly on the PE beam energy, as is illustrated in Figure A.1.

BES's usually originate from deeper within the specimen, and do not provide much information about the surface topology. The back-scattering coefficient η is defined by the ratio of back-scattered electrons N_{bse} to the number of electrons in the primary beam N_{pe}:

$$\eta = \frac{N_{bse}}{N_{pe}} \tag{A.1}$$

The back-scattering coefficient is directly dependent on the atomic number as illustrated in figure A.2. Therefore, BSEs are very well suited for analyzing the atomic composition of a specimen. [50][1]. Furthermore, as the angle θ between the specimen surface normal and the primary beam increases, the back-scattering coefficient also increases [51]. This can be approximated by:

$$\eta(\theta) \approx \eta_0(1 + \sin(\theta)) \tag{A.2}$$

The angular distribution of BSE's $\eta(\varphi)$ follows a cosine distribution as described by equation A.3 [3] centered around the reflection angle ϕ of the PE beam on the specimen surface.

$$\eta(\varphi) \propto \cos(\varphi + \phi) \tag{A.3}$$

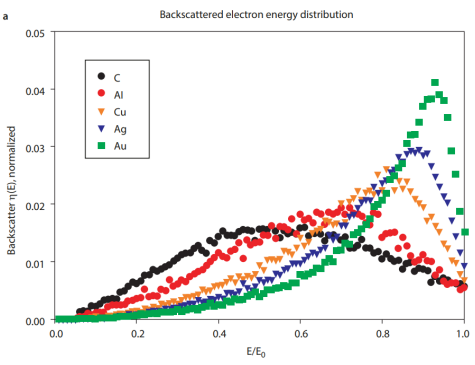


Figure A.1: Energy of BSE's relative to the PE beam energy [1]

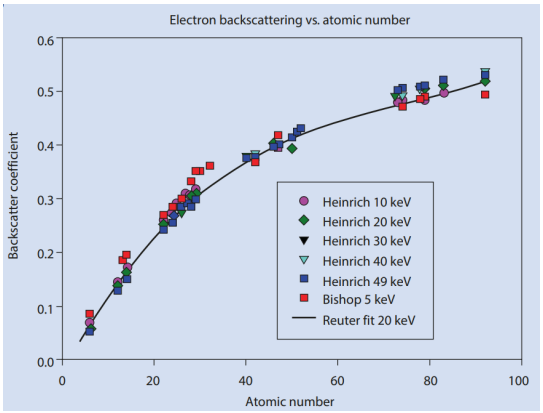


Figure A.2: Back-scattering coefficient vs atomic number [52] [53].

The reflection angle ϕ is given by equation A.4 where $\mathbf{\bar{r}_{PE}}$ is the beam unit vector and $\mathbf{\bar{n}}$ is the specimen surface normal.

$$\phi = 2\arccos\left(\mathbf{\bar{r}_{PE}} \cdot \mathbf{\bar{n}}\right) \tag{A.4}$$

These behaviors allow for contrasting mechanisms where differences in the relative number of back-scattered electrons can be used to map the surface topography of the specimen. Furthermore, in segmented detectors, the directionality of the back-scattered electrons described in equation A.3 can be used to further improve the mapping of the surface topology.

A.2. Secondary electrons

Inelastic scattering is characterized by a noticeable change in energy of the scattered electron. This can occur when PE's interact with bound or valence electrons within the specimens atoms, vacating the electron out of the atom's shell resulting int a SE [1]. The energy of these SE's is rather small compared to that of the PE's and BSE's. Figure A.5 shows the distribution of SE energy for various materials.

Due to these SE's electrons having low energy, there is a large chance they get reabsorbed by different atoms in the specimen. Therefore, they only escape from the specimen if they are close to the surface as illustrated in figure A.3. The exact escape depth for SEs depends on the specimen composition and PE beam energy but is less than 5 nm for conductors, and less than 50 nm for insulators [54]. Therefore, SEs mainly provide information about the surface of the specimen as illustrated in figure A.4. Furthermore, the emission coefficient for SE's increases as the PE beam energy decreases [55] as illustrated in figure A.6. The secondary electron emission coefficient δ is defined by the ratio of secondary electrons emitted from the specimen N_{se} to the number of electrons in the primary beam N_{pe}:

$$\delta = \frac{N_{se}}{N_{pe}} \tag{A.5}$$

The secondary electron coefficient is also related to the tilt of the specimen surface [1]. This can be approximated through:

$$\delta(\theta) \approx \frac{\delta_0}{\cos(\theta)} \tag{A.6}$$

Where the angle between the surface normal and the PE beam θ is given by:

$$\theta = \arccos\left(\mathbf{\bar{r}_{PE}} \cdot \mathbf{\bar{n}}\right) \tag{A.7}$$

where $\mathbf{\bar{r}_{PE}}$ is the beam unit vector and $\mathbf{\bar{n}}$ is the specimen surface normal.

Similar to BSE's, the angular distribution of SE's is proportional to a cosine distribution. However, the cosine distribution is centered at the surface normal of the specimen. The secondary electron

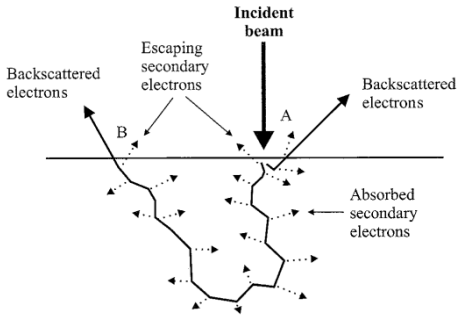


Figure A.3: PE path through specimen, illustrating reabsorbed and emitted SEs [50].

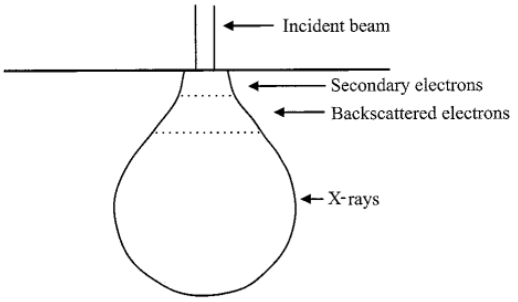


Figure A.4: SEM signals and the depth of which they contain most information[50].

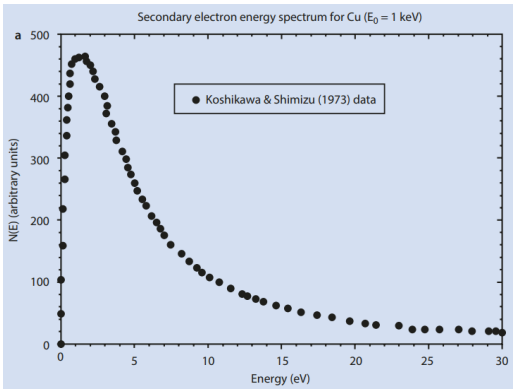


Figure A.5: Energy distribution of SE's for copper [2]

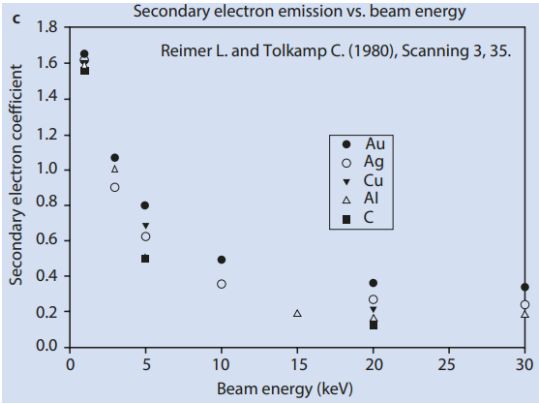


Figure A.6: SE coefficient for silicon and copper vs PE beam energy [56]

coefficient $\delta(\varphi)$ as a function of the angle to the PE beam φ is proportional to:

$$\delta(\varphi) \propto \cos(\varphi + \theta) \tag{A.8}$$

The most used signal within SEM's is that of the SE's. However, the SE's exit the specimen in random directions. In order to collect and detect enough an electric field is applied from the specimen to the detector. Due to the low energy of the SE's they are pulled toward the detector by the electric field, allowing most of the emitted SE's to be detected. [50]

A.2.1. SE signal behavior

The signal obtained from SE's directly depends on the surface topography of the specimen [11]. The most intuitive way to estimate the SE signal intensity is by observing the interaction volume of the PE's for different structures on the specimen surface. As illustrated in A.4 the interaction volume of the PE's is a teardrop shaped structure. The PE's will follow a random path within this teardrop structure due to the scattering. Therefore, whenever a part of this teardrop structure lies closer to the specimen surface, or even outside of the specimen surface, the probability of SEs or BSEs being scattered out of the specimen rather than deeper into it are increased [54]. Figure A.7 illustrates this for step-function structures, and sloping structures on the specimen surface. Figure A.8 also shows an estimation of the SE signal strength obtained from these structures.

These examples illustrate how the peak signal strength from SE's are most relevant when analyzing the surface profile of a specimen. As these hold most information regarding height variations on the surface or the specimen. Especially within the verification of semiconductors, as they contain many step-function like structures, like interconnect wires.

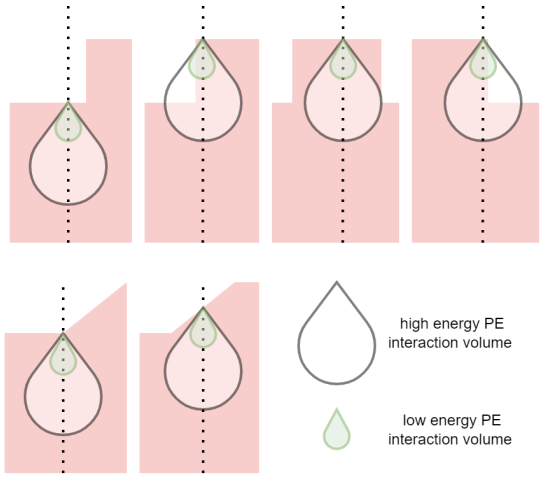


Figure A.7: Illustration of PE interaction volume depending on the surface profile of the specimen. The smaller the distance between the interaction volume and the specimen surface, the larger the probability that PE's and BSE's are scattered out of the specimen as opposed to deeper.

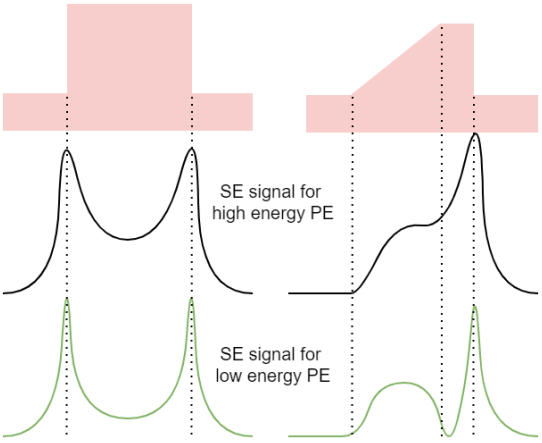


Figure A.8: Estimated SE signal for step-function structure (left) and a sloping structure followed by a step-function structure (right).

B

Model of electron incidence at sensor

In the following section, a model will be derived that can be used to approximate the distribution of incident electrons throughout the sensor matrix. Using this model, we will then analyze the results for the combined and individual distribution of BSE's, SE's.

B.1. Model Derivation

In order to create a model for the distribution of incident electrons across the entire sensor we must take a closer look at the angular distribution of SE's and BSE's. As previously mentioned in Section A Koshikawa and Shimizu [2] and Niedrig [3] analyzed the behavior of SE's and BSE's. They concluded that the angular distribution of the emission coefficient η_0, which relates the emitted particles from the sample N_e to the number of particles in the PE beam N_{PE} as given in equation B.1, is proportional to the cosine of the angle of emission. This proportionality is given by equation B.2 where φ is the angle relative to the primary beam $\hat{\mathbf{r}}_{PE}$ reflected by the specimen surface with the normal vector $\hat{\mathbf{n}}$ and the particle vector $\hat{\mathbf{r}}$ as given in equations B.3 and B.4. This relation was observed for both SE's and BSE's.

$$\eta_0 = \frac{N_e}{N_{PE}} \tag{B.1}$$

$$\eta \propto \cos\varphi \tag{B.2}$$

$$\hat{\mathbf{r}}_{PE'} = \hat{\mathbf{r}}_{PE} - 2(\hat{\mathbf{r}}_{PE} \cdot \hat{\mathbf{n}})\hat{\mathbf{n}} \tag{B.3}$$

$$\varphi = \cos^{-1}(\hat{\mathbf{r}}_{PE'} \cdot \hat{\mathbf{r}}) \tag{B.4}$$

A schematic representation of the system is shown in figure B.1. The objective lens focuses the primary beam to the desired location within the scan area. Depending on the radius of the field of view R_{fov} the maximum angle of incidence of the primary beam to the normal surface of the sample is given by equation B.5. As a result, the center of the cosine distribution in equation B.2 varies as the primary beam scans the sample surface. In typical SEM systems, the working distance d_{wd} is between 1 - 5 mm [1], while the field of view is typically two to three orders of magnitude smaller [1]. Therefore, it can be approximated that the angle of incidence of the primary beam will always be perpendicular to the optical axis, as shown in equation B.6.

$$\Phi = \tan^{-1}\frac{R_{fov}}{d_{wd}} \tag{B.5}$$

$$\hat{\mathbf{r}}_{PE} \approx \begin{bmatrix} 0 \\ 0 \\ -1 \end{bmatrix} \tag{B.6}$$

Electrons leaving the specimen have some momentum in the angular direction which is related to their kinetic energy K by equation B.7, where $\hat{\mathbf{r}}$ is the directional unit vector and m is the mass of the electron.

$$\vec{v}_0 = \hat{\mathbf{r}}\sqrt{\frac{K}{m}} \tag{B.7}$$

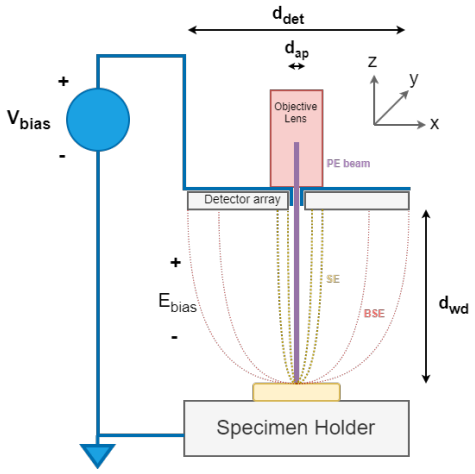


Figure B.1: Schematic of specimen (yellow) in a SEM

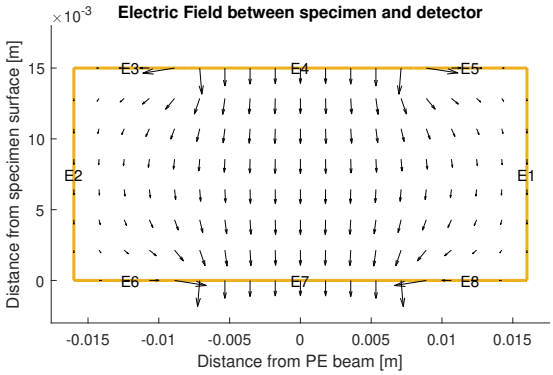


Figure B.2: Electric field for $V_{bias} = 6000V$, with the detector metal plate (edge E4) connected to the positive terminal, and the speciman holder (edge E7) connected to the negative terminal.

A bias voltage V_{bias} is applied to a thin metal sheet above the detector and the specimen holder is connected to the ground. This creates an electric field between these points defined by equation B.8.

$$\vec{E}(x,y,z) = \nabla V \tag{B.8}$$

Any charged particle within an electric field experiences a force as defined in Equation B.9. This force results in an acceleration of the particle as shown in equation B.11, and therefore in a change in velocity as shown in equation B.12.

$$\vec{F}(x,y,z) = q\vec{E}(x,y,z) \tag{B.9}$$

$$\vec{p} = x\hat{\mathbf{x}} + y\hat{\mathbf{y}} + z\hat{\mathbf{z}} \tag{B.10}$$

$$\vec{a}(x,y,z) = \frac{\vec{F}(x,y,z)q}{m_e} = \frac{d^2\vec{p}}{dt^2} \tag{B.11}$$

$$\vec{v}(x,y,z) = \int \vec{a}(x,y,z)dt = \frac{d\vec{p}}{dt} \tag{B.12}$$

These equations can be divided into cardinal directions and written as a system of differential equations $\mathbf{w}'(t) = \mathbf{A}(t)\mathbf{w}(t)$. Equation B.13 defines the column vector used to represent the system, equation B.14 defines the derivatives of the system, and equation B.15 defines the initial values of the system.

$$\mathbf{w} = [w_1, w_2, w_3, w_4, w_5, w_6]^{-1} = \left[x, \frac{dx}{dt}, y, \frac{dy}{dt}, z, \frac{dz}{dt}\right] \tag{B.13}$$

$$\mathbf{w}' = \left[w_2, \frac{\hat{\mathbf{x}}\vec{E}(w_1,w_3,w_5)q}{m}, w_3, \frac{\hat{\mathbf{y}}\vec{E}(w_1,w_3,w_5)q}{m}, w_5, \frac{\hat{\mathbf{z}}\vec{E}(w_1,w_3,w_5)q}{m}\right]^{-1} \tag{B.14}$$

$$\mathbf{w}(0) = \left[x_0, \hat{\mathbf{r}}\cdot\hat{\mathbf{x}}\sqrt{\frac{K}{m}}y_0, \hat{\mathbf{r}}\cdot\hat{\mathbf{y}}\sqrt{\frac{K}{m}}, z_0, \hat{\mathbf{r}}\cdot\hat{\mathbf{z}}\sqrt{\frac{K}{m}},\right]^{-1} \tag{B.15}$$

Under the assumption that the electric field strength E_z in the z-direction is uniform and zero in the x and y directions, and that the electrons do not lose any energy due to the vacuum, a general solution

can be formulated given by equation B.16

$$\mathbf{A}(t)\mathbf{w}(t) = \begin{bmatrix} t(\hat{\mathbf{r}}\cdot\hat{\mathbf{x}}\sqrt{\frac{K}{m}}) + x_0 \\ \hat{\mathbf{r}}\cdot\hat{\mathbf{x}}\sqrt{\frac{K}{m}} \\ t(\hat{\mathbf{r}}\cdot\hat{\mathbf{y}}\sqrt{\frac{K}{m}}) + y_0 \\ \hat{\mathbf{r}}\cdot\hat{\mathbf{y}}\sqrt{\frac{K}{m}} \\ \hat{\mathbf{r}}\cdot\hat{\mathbf{z}}\sqrt{\frac{K}{m}}\frac{\hat{\mathbf{z}}E_z q}{2m}t^2 + z_0 \\ \hat{\mathbf{r}}\cdot\hat{\mathbf{z}}\sqrt{\frac{K}{m}} + \frac{\hat{\mathbf{z}}E_z q}{m}t \end{bmatrix} \tag{B.16}$$

Using Matlab this system can also be solved numerically. The electric field is modeled using an electrostatic analysis model within Matlab. This allows for more accurate results for the electron landing location on the detector compared to a homogeneous electric field. For the electric field model the width of the specimen holder and detector are equal. The resulting electric field is shown in figure B.2 where line E7 is the specimen holder and line E4 is the metal sheet above the detector. The system of differential equations is then solved for a number of evenly spaced sample points over the escape angles. First the exit angle θ relative to the z-axis is divided into n equal samples for the interval $[0; \frac{\pi}{4}]$, therefore the angle of k'th sample is given by

$$\theta_k = \frac{k\pi}{4n} \tag{B.17}$$

And its scalar projection onto the xy plane is given by:

$$r_{xy} = \sin\theta_k \tag{B.18}$$

Next each of theses samples is rotated around the z-axis such that the arc length between adjacent points is consistent. The arc length between two points is given by $L = R\phi$ where in this case $R = r_{xy}$ and ϕ is the rotation around the z-axis, seen from the positive x-axis. The arc length between the k samples is $L_k = \frac{\pi}{4n}$ The number of sample points around the z-axis m_k, and their angles $\phi_{k,l}$ are given by:

$$m_k = \text{floor}\left[\frac{2\pi r_{xy}}{L_k}\right] = \text{floor}\left[\frac{2\pi^2\sin(\frac{k\pi}{4n})}{4n}\right] \tag{B.19}$$

$$\phi_{k,l} = \frac{2\pi l}{m_k} \tag{B.20}$$

Therefore the x and y and z components of the (k, l)'th sample point are given by:

$$\hat{\mathbf{r}}_{k,l}\hat{\mathbf{x}} = r_{xy}\cos\phi_{k,l} \tag{B.21}$$

$$\hat{\mathbf{r}}_{k,l}\hat{\mathbf{y}} = r_{xy}\sin\phi_{k,l} \tag{B.22}$$

$$\hat{\mathbf{r}}_{k,l}\hat{\mathbf{z}} = \sin\theta_k \tag{B.23}$$

The proportional intensity of each sample point for a certain specimen surface normal $\hat{\mathbf{n}}$ and the primary beam direction $\hat{\mathbf{r}}_{PE}$ be derived from equations B.2, B.3 and B.4 and is given by:

$$\eta_{k,l} = (\hat{\mathbf{r}}_{PE} - 2\,(\hat{\mathbf{r}}_{PE}\cdot\hat{\mathbf{n}})\,\hat{\mathbf{n}})\cdot\begin{bmatrix}\sin(\frac{k\pi}{4n})\cos(\frac{2\pi l}{m_k}) \\ \sin(\frac{k\pi}{4n})\sin(\frac{2\pi l}{m_k}) \\ \sin(\frac{k\pi}{4n})\end{bmatrix} \tag{B.24}$$

If the sample points were spaced unevenly the values of $\eta_{k,l}$ would have to be normalized to the solid angle occupied by the sample point. However, by choosing sample points that are equally spaced,

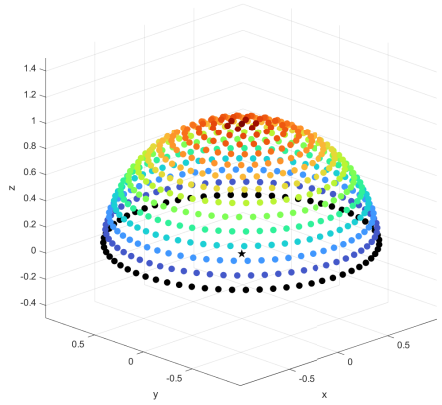


Figure B.3: 3-d view of sample angles originating from the origin

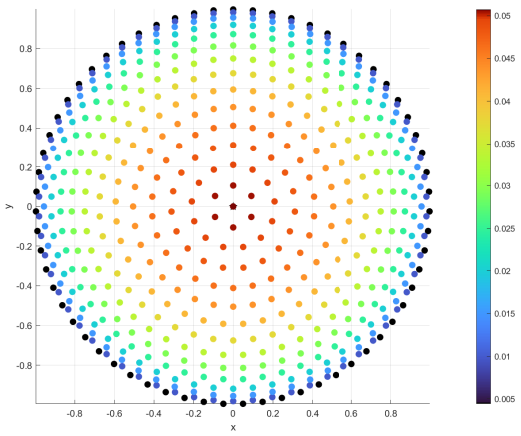


Figure B.4: top view of sample angles originating from the origin

the solid angles are all equal, and $\eta_{k,l}$ properly reflects the proportional intensity relative to the other sample points. The resulting sample points for $n = 15$, and $\hat{\mathbf{r}}_{PE'} = [0, 0, -1]'$ are shown in figure B.3 and B.4.

The model solves the differential equations for these sample points for the following input parameters:

- K_{PE}: Landing energy of primary electrons. The values of $\eta_{bse,0}$, K_{bse} and $\eta_{se,0}$ can be derived from this parameter using figures A.2, A.1 and A.6 respectively. The landing energy of PE's is less then then energy of the PE beam, as the electrons are slowed down by the electric field.
- K_{se}: Kinetic energy of secondary electrons exiting the sample. These values are automatically derived from figure A.5.
- V_{bias}: Voltage difference between the specimen holder and the metal plate above the detector.
- d_{wd}: Working distance.
- $\hat{\mathbf{n}}$: Surface normal.

The outputs of the model are three vectors containing the x- and y-coordinates of the incidence of each electron in the detector plane, and the corresponding value of $\eta_{k,l}$ of those electrons. From these results a contour plot can be created of η normalized to the area of a single detector ($A_{det} = 0.165 \times 0.165$ mm^2). However, the outputted vectors contain scattered data (shown in figure B.5) which has to be converted to gridded data. First the detector area is subdivided into bins, where each bin $\mathbf{B}$ has the height and width equal to the size of a detector. Next the $\eta_{k,l}$ of electrons whose incidence locations are within the bin are added, and the total value is assigned to that bin. The code that implements this can be found in Appendix G.2.5. This converts the scattered data into gridded data, as is illustrated in figure B.5.

Due to the limited sample points the gridded data is not smooth. Every sample point in the model represents some solid angle of emission at the specimen, and therefore should represent a certain area of incidence upon the detector. To estimate this area of incidence the binned data is convolved with a kernel $\mathbf{K}$ as shown in equation B.25. The kernel size is varied based on the average distance between sample points at the detector.

$$\mathbf{D}(i,j) = \mathbf{K} * \mathbf{B}(i,j) \tag{B.25}$$

This smooths out any high frequency components in the data over all the bins as shown in figure B.5. Furthermore, this method ensures that the sum of values in each bin stay equal to the sum of the original values (equation B.26, therefore the data is implicitly normalized to the surface area of a single

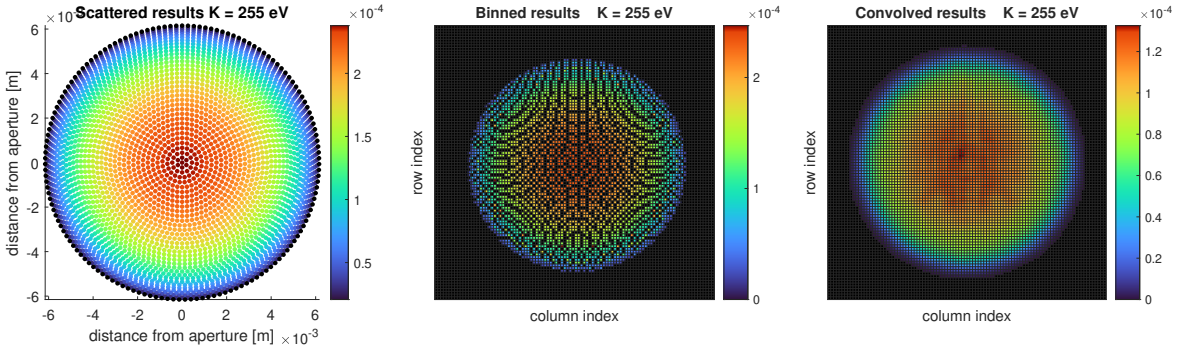


Figure B.5: Scattered output data of the model (left); Binned scatter data (center); Convolved binned data (right)

detector. The kernel is a circular smoothing kernel, where each non-zero element has the same value, and the sum of all elements is equal to one.

$$\sum_{k=1}^{n}\sum_{l=1}^{m_k}\eta_{k,l} = \sum_{i}\sum_{j}\mathbf{D}(i,j) \tag{B.26}$$

B.1.1. Analysis

To analyze the behavior of the distribution of incident electrons the following input parameters are used for the model:

- $K_{PE} = 300\ eV$
- $V_{bias} = 6\ kV$
- $d_{wd} = 15\ mm$

along with the sensor dimensions as presented in section 2.1. First the results for BSE's will be analyzed, followed by the results for SE's, and finally their combined results.

BSE model

The kinetic energy of BSE's depends on the landing energy of the primary electrons, as does the value of $\eta_{bse,0}$. Using figures A.1 and A.2 their values are found to be:

- $\eta_{bse,0} = 0.4$
- $K_{bse} = 0.85K_{PE} = 255eV$

BSE

The resulting distribution is shown in figure B.6. Using these same values the model is also applied to different angles of the surface specimen, where it is tilted 45° in each cardinal direction. The resulting distributions are shown in figure B.7.

SE model

The kinetic energy of SE's follows the distribution of Figure A.5. The value of $\eta_{se,0}$ is related to the energy of the PE beam as shown in Figure A.6. To obtain the distribution for incident SE's multiple energy levels are fed into the model, and the resulting distributions are combined based on the probability of these energy levels using equation B.27.

$$\mathbf{D}_{tot} = \sum \mathbf{D}_i P_i \tag{B.27}$$

The values used for the model are:

- $\eta_{se,0} = 1.6$

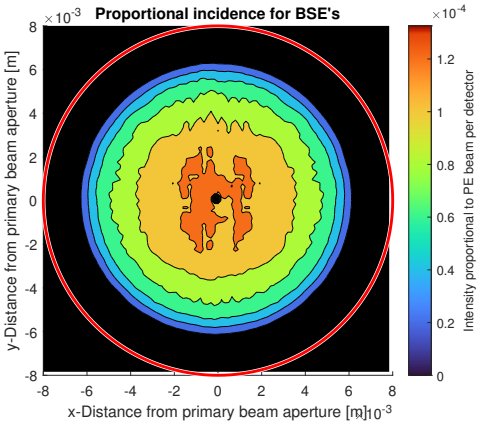


Figure B.6: Proportional incidence of BSE's relative to PE beam

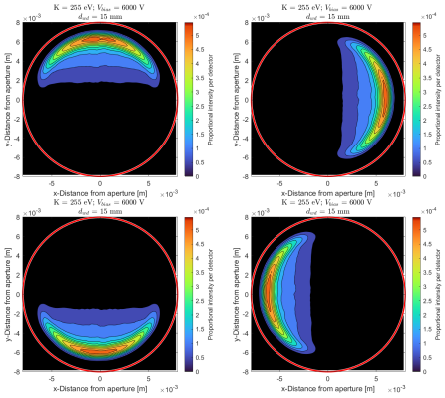


Figure B.7: Proportional incidence of BSE's relative to PE beam for 45° tilted sample surfaces

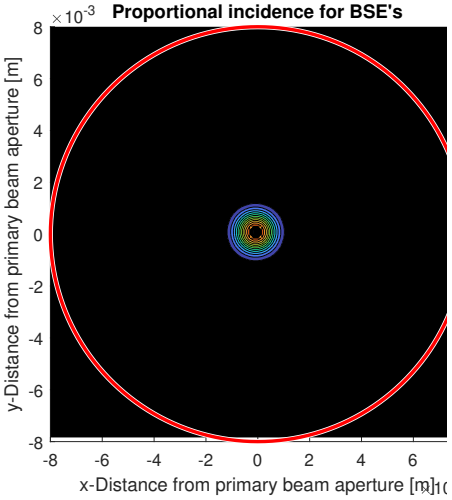


Figure B.8: Proportional incidence of SE's relative to PE beam

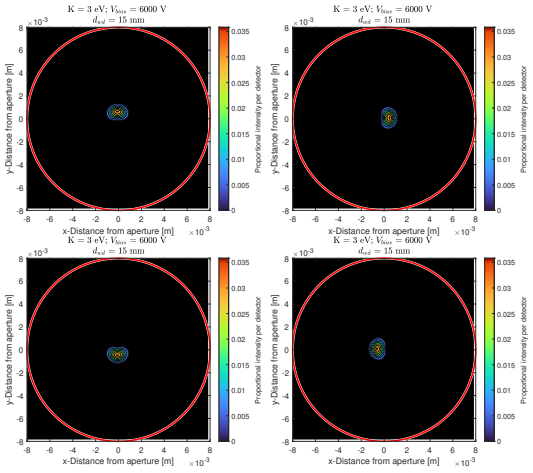


Figure B.9: Proportional incidence of SE's relative to PE beam for 45° tilted specimen surfaces

- $K_{bse} = [1.5;\ 2;\ 3;\ 4;\ 5;\ 7]\ eV$
- $P = [0.21;\ 0.24;\ 0.20;\ 0.15;\ 0.12;\ 0.08]$

The resulting distribution is shown in figure B.8. The results of a tilted sample surface are shown in Figure B.9. Looking at these figures it is clear that the SE's are incident over a very small area in the center of the detector. Even with a tilted sample, all SE's remain within this small area. As a result of the small incidence area, and the relatively high value of $\eta_{se,0}$ the proportional incidence of SE's is a couple orders of magnitude larger than that of the BSE's.

C

Layout

This Appendix covers techniques used for digital IC design and is intended to provide the reader with the background to understand the design choices made throughout this thesis. First, a technique for optimizing the delay of sequential logic is described. Following this, various aspects of on-chip interconnect are covered, including parasitics and signal delay.

C.1. Interconnect

A large part of the IC design is the optimization of the connections between components. Unfortunately, interconnection wires have many parasitic properties.

C.1.1. Resistive and Capacitive parasitics

The resistance of any metal wire depends on the resistivity ρ of the material and the relationship between the length of the wire and its cross section. This can be expressed as

$$R = \frac{\rho L}{A} \tag{C.1}$$

In an IC design, the material used within a metal layer is generally predefined along with the height of the wires. Therefore, the resistance of the wire is expressed using the sheet resistance $R_{\square}$ and the ratio between length and width as

$$R = R_{\square} \frac{L}{W} \tag{C.2}$$

with

$$R_{\square} = \frac{\rho}{H} \tag{C.3}$$

Similarly, the capacitance between two parallel metal plates depends on the ratio between the permittivity of the insulating material ϵ and the distance between the metal plates t, and scales proportionally to the area overlapping of these two plates. This can be defined as

$$C = \frac{\epsilon}{t} WL \tag{C.4}$$

Adjacent metal wires within the same layers form a parasitic parallel plate capacitor with one another. Metal wires on different layers also form parasitic parallel plate capacitors with each other. Furthermore, metal wires also form parasitic parallel plate capacitors with the substrate. Metal wires also have a certain resistance that limits current flow. These parasitic properties are summarized in Figure C.1.

For a wire of length L the parasitic resistances, along with the parasitic capacitance's, are distributed throughout its length, resulting in a distributed RC network. Resistance and capacitance can be expressed as the resistance and capacitance per unit length as

$$c = \frac{C}{L} \tag{C.5}$$

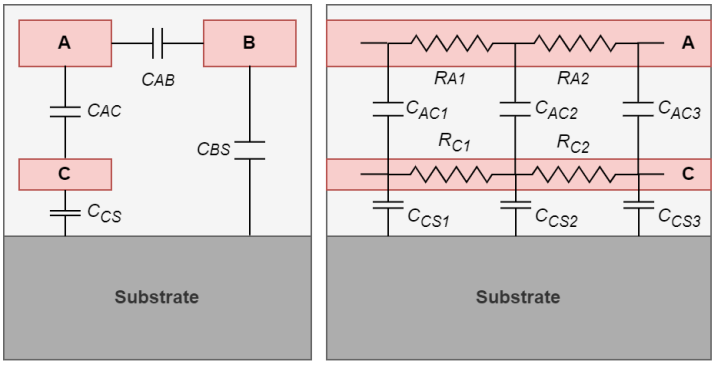


Figure C.1: Front and side view of parasitic capacitance and resistance of metal wires in layer one and two of an IC.

$$r = \frac{R}{L} \tag{C.6}$$

Using these expressions, the first order time constant of the wire can be expressed as [44]

$$\tau_D = \frac{rcL^2}{2} = \frac{RC}{2} \tag{C.7}$$

and the propagation delay of the wire can be expressed as

$$t_p = 0.38rcL^2 \tag{C.8}$$

From equations C.2, C.4 and C.7 the following conclusions can be drawn:

- The delay of a wire increases quadratically with its length.
- Although increasing the width of a wire decreases its parasitic resistance, it also increases the parasitic capacitance by the same amount or more depending on the surrounding geometry.

Repeaters (also known as buffers) or inverters can be used to reduce the propagation delay of sufficiently long wires [57][58]. Repeaters are basically two inverters connected in series, providing a non-inverted output at the cost of twice the gate delay. Therefore, inverters have the preference over buffers, but require an even number to ensure that the signal is not inverted at the destination. Given the delay of the inverter t_{inv} and the propagation delay of the unbuffered wire t_{pwire} the ideal number of inverters is given by

$$m_{opt} = \sqrt{\frac{0.38rcL^2}{t_{inv}}} = \sqrt{\frac{t_{pwire}}{t_{inv}}} \tag{C.9}$$

The inverters are to be placed equal distances apart and the propagation delay of the individual wire segments should be equal to the inverter delay. This reduces the propagation delay to

$$t_{p,buff} = 2m_{opt}t_{inv} = 2\sqrt{t_{pwire}t_{inv}} \tag{C.10}$$

Furthermore, the length of the wire for which the delay is equal to the delay of the inverter is obtained through

$$L_{crit} = \frac{L}{m_{opt}} = \sqrt{\frac{t_{inv}}{0.38rc}} \tag{C.11}$$

Furthermore, the propagation delay of a wire segment of critical length is independent of the routing layer. The critical length, however, does vary depending on the routing layer.

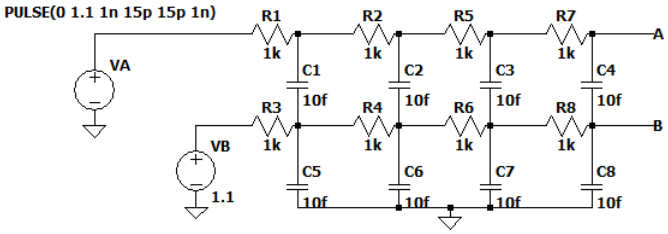


Figure C.2: Schematic of parasitics present in wire A and B from figure C.1

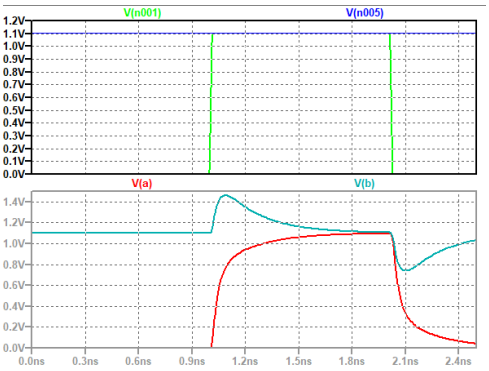


Figure C.3: Transient response for low to high data change on wire A

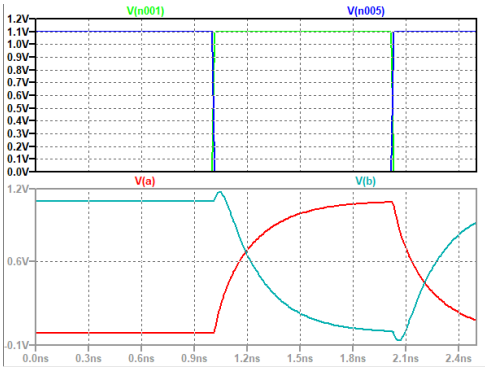


Figure C.4: Transient response for low to high data change on wire A, and a high to low data change on wire B

C.1.2. Crosstalk

Looking at figure C.1 it becomes clear that the wires are capacitively coupled to each other. Therefore, if the voltage on wire A changes, both wire B and C will see a transient voltage response due to the parasitic capacitance. A schematic model for this parasitic effect is shown in Figure C.2. The transient response for a low to high transition only on wire A is shown in figure C.3. The propagation delay for the signal on wire A is approximately 56 ps in this case. However, when a low-to-high transition occurs on wire A, and a high-to-low transition occurs on wire B, the transient response changes to that of figure C.4. In this case, the propagation delay for the signal on wire A increases to 150 ps. Therefore, it can be concluded that interwire capacitance of wires that carry data results in data-dependent propagation delays.

The phenomenon of two adjacent wires interfering with each other is known as crosstalk and can drastically degrade circuit performance. This can be attributed in large part to the data-dependent propagation delay. In the worst-case scenario, a single wire can receive crosstalk from four adjacent wires (above, below, left, and right). One way to deal with crosstalk would be to design the entire circuit for the worst-case propagation delay. However, this results in a circuit that is significantly slower. Alternatively, the circuit can be carefully laid out to minimize, or even mitigate, the effects of crosstalk through the following rules [59]:

1. Parallel wires should be spaced at least three times their width apart. This significantly reduces the parasitic capacitance between the wires.

2. Minimize the parallel run length of two wires on the same layer to prevent the capacitance between them from growing too large.

3. Alter the direction of routing between adjacent layers; this way, the metal layers can be routed independently from one another without the risk of high capacitance between wires above or below.

4. For signal busses, provide a shielding wire connected to VDD or GND between every signal wire. In this way, the interwire capacitance instead becomes a capacitance to either VDD or GND, and the data-dependent propagation delay is eliminated.

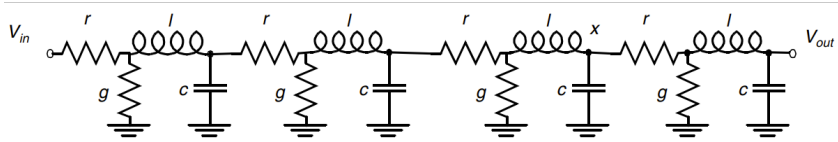


Figure C.5: Transmission line model of a wire[44]

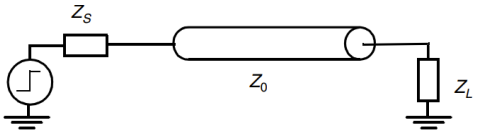


Figure C.6: Transmission line representation with a source impedance Z_s, wire impedance Z_0 and a load impedance Z_L [44]

C.1.3. Transmission line effects

As the speed of IC's increases, the parasitic inductance must also be taken into consideration. Furthermore, the physical limitations of an electromagnetic wave need to be taken into account. The transmission line model shown in figure C.5 is the most accurate approximation for this [44].

The propagation speed of an electromagnetic wave inside the wire is directly related to the inductance and capacitance per unit length:

$$v = \frac{1}{\sqrt{lc}} \tag{C.12}$$

Furthermore, the IC wire transmission line model is described by its characteristic impedance as:

$$Z = \sqrt{\frac{l}{c}} \tag{C.13}$$

Whenever the characteristic impedance changes suddenly, any wave propagating through the wire is partially reflected. This reflection is determined by the reflection coefficient as:

$$p = \frac{Z_1 - Z_0}{Z_1 + Z_0} \tag{C.14}$$

In an IC there is generally some logic gate, whose output is connected to a wire, which is then connected to the input of the next logic gate. The output of the logic gate and the input both have a characteristic impedance associated with them, the source and the load impedance, respectively. The wire that connects the two also has a certain impedance. Furthermore, the output of a logic gate is generally a step function. The model of such a circuit is shown in figure C.6. If the resistance of the wire is very small, it is referred to as a lossless transmission line.

The transient responses for three scenarios of a lossless transmission line with an open output ($Z_L = \infty$) are shown in figure C.7. In the first case, the source impedance is much larger than that of the wire. Therefore, once the EM wave arrives at the interface, a large part of the signal is reflected back. When the wave reaches the load, it is fully reflected. This process repeats multiple times before the output settles. In the second case, the source impedance is matched to that of the wire, resulting in no reflection at all until the wave reaches the output. In the last case, the source impedance is much smaller than that of the wire, resulting in a large part of the source signal propagating through the wire. At the output, this large signal is reflected and doubles the voltage at the output. This process repeats multiple times, resulting in a ringing behavior of the output.

When the resistance of the wire is taken into account, the behavior changes and the transmission line is called a lossy transmission line. Due to resistance, the wave propagating through the wire is attenuated and the step response presented at the input is relaxed. Furthermore, if the resistance of the wire is larger than five times the wire's impedance, the wire can be better approximated as a distributed RC line as described in the previous section. To determine when the transmission line effects should be taken into account, the following rules of thumb are used:

1. The time of flight is larger than the rise time of the input signal.

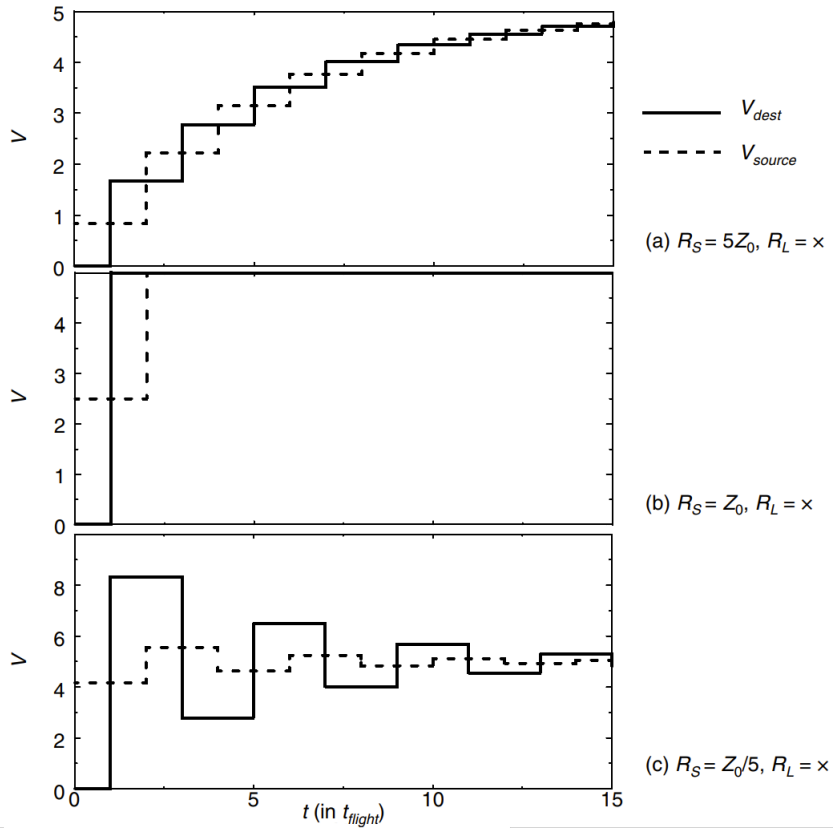


Figure C.7: Transient response of transmission line for different characteristic impedance's [44]

2. The resistance of the wire is smaller than five times its impedance.
3. If the wire's resistance is smaller than twice its impedance, it can be approximated as a lossless transmission line.

Rules 1 and 2 can be summarized as follows:

$$\frac{t_r(t_f)}{2.5}\frac{1}{\sqrt{lc}} < L < \frac{5}{r}\sqrt{\frac{l}{c}} \tag{C.15}$$

D

PDK characterization

The standard cell library (SCL) implements different drive strengths for each logic gate, including inverters. The drive strength is denoted by D[*strength*], where the strength indicates the number of parallel devices of minimum size used. Due to the large distances that signals will have to traverse, repeaters must be used to achieve acceptable propagation delays, as discussed in Appendix C. The critical wire length is determined using the following method:

1. Using Virtuoso, a layout is made in which an inverter of each available drive strengths drives a 1000 μm long wire on M2. To account for the worst-case scenario, the interconnect wires are shielded on both sides by a minimum-size wire connected to the VDD, at a minimum pitch distance of 0.07 μm.
2. The parasitics are extracted from the layout and simulated for an input with a rise time of 15 ps (which matches the rise/fall times at the output of an inverter chain).
3. From the post-layout simulation the propagation delay is extracted using Matlab.
4. From Virtuoso simulations the delay of the inverters with a fan-out of 1 (i.e., an inverter output connected to an inverter input with identical drive strength) is extracted using Matlab.
5. Using Equation C.11 the critical length is calculated.

Providing the following the results:

$$c_{wire,2-5} = 0.17348 \ \frac{fF}{\mu m_{pdk}} \tag{D.1}$$

$$L_{crit,2-5} = 234 \ \mu m_{pdk} \tag{D.2}$$

Where $c_{wire,2-5}$ is the capacitance per unit length for a minimum width wire on M2-M5 and $L_{crit,2-5}$ is the critical length for wires on M2-M5. Using the critical length, an additional simulation is performed in Virtuoso to extract the critical length delay and energy. These results are used to calculate the energy delay product and the energy per μm_{pdk} using the Matlab script from the Appendix G.3.1. The results are shown in Figure D.1, which illustrates that a repeater with drive strength **D8** is the most power efficient choice, while a repeater with driver strength **D24** minimizes the energy-delay product.

As M6 has a lower sheet resistance, the critical length will be longer. Due to the decrease in sheet resistance higher drive strengths will have a significantly larger impact on the propagation delay. For a **D24** inverter the following values were found:

- $L_{crit,6} = 740 \ \mu m$
- $t_{p,Lcrit,6} = 68 \ ps$
- $EpU_6 = 0.227 \ fJ$

As stated previously, M6 should be avoided when routing digital signals, as it requires more than twice as much energy compared to signals routed through M2-M5.

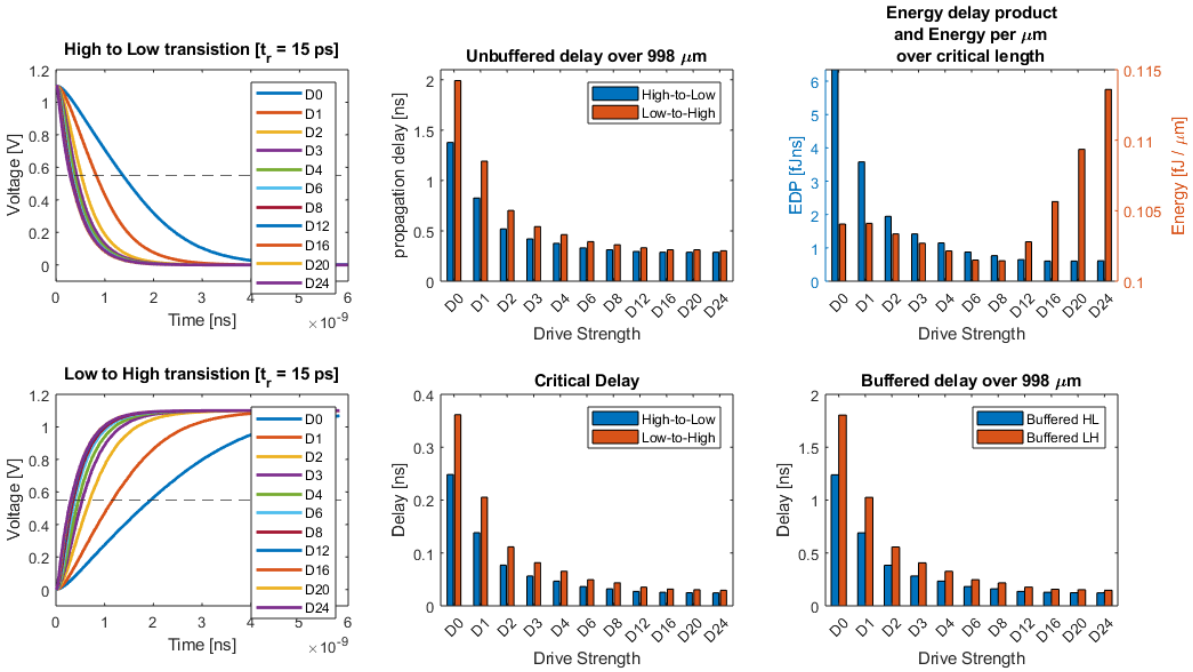


Figure D.1: Analysis of inverter as repeater for M2-M5 in the TSMC 40n PDK. A repeater with drive strength **D8** is shown to be the most power efficient, while a repeater with a drive strength of D24 provides the smallest EDP.

D.0.1. Combinational Logic Sizing

The pipeline segments primarily use full adders in their combinational logic. In Section D the properties of the inverter were determined when used as a repeater. In this subsection, the critical path delay for different drive strength FA connected to a DETDFF is determined using the PDK datasheet and the Matlab script in Appendix G.3.2. The results in Figure D.2 show that an FA with drive strength **D0** should be avoided to minimize the delay in the critical path. Furthermore, the difference in performance between **D1-D4** is rather small. As a consequence, only FA's with a drive strength of **D1** should be used to minimize power consumption.

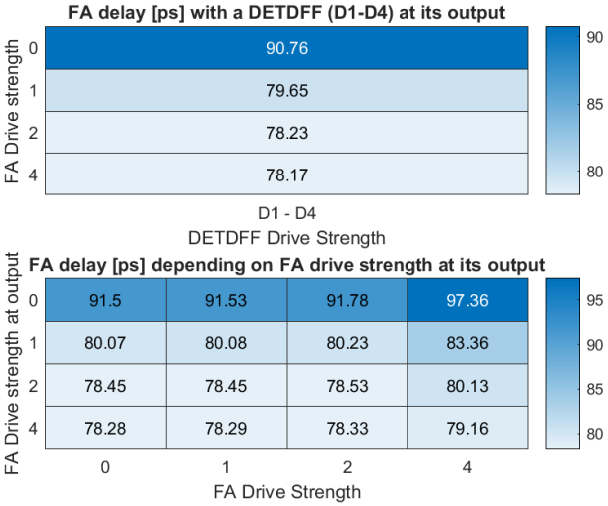


Figure D.2: Critical Path delay for FA connected to a DETDFF for different drive strengths (top) and Critical Path delay for FA connected to another FA for different drive strengths (bottom)

E

Detailed design of the backend readout pipeline

Cluster size (L1 clusters)	Repeaters	Logic path (layers)		Clock propagation delay (ps)		Data propagation delay (ps)		Maximum logic delay (ps)		Path delay (ps)		Arrival time relative to H-Tree clock (ps)		Δ Arrival time (ps)	Output Bits
		Min	Max	FF	SS	FF	SS	FF	SS	FF	SS	FF	SS		
L1															4
64	6	5	11	126	246	294	540	539	1111	833	1651	959	1897	938	
128	10	5	11	210	410	490	900	539	1111	1029	2011	1239	2421	1182	
256	14	6	12	294	574	686	1260	588	1212	1274	2472	1568	3046	1478	
512	22	7	13	462	902	1078	1980	637	1313	1715	3293	2177	4195	2018	
L2															5
2	8	1	5	168	328	392	720	245	505	637	1225	805	1553	748	
4	16	2	6	336	656	784	1440	294	606	1078	2046	1414	2702	1288	
8	32	3	8	672	1312	1568	2880	392	808	1960	3688	2632	5000	2368	
L3															6
2	16	1	6	336	656	784	1440	294	606	1078	2046	1414	2702	1288	
4	32	2	7	672	1312	1568	2880	343	707	1911	3587	2583	4899	2316	
Signal routing through M6															
2	4	1	6	336	656	252	528	294	606	546	1134	882	1790	908	
4	8	2	7	672	1312	504	1056	343	707	847	1763	1519	3075	1556	
8	16	3	8	1344	2624	1008	2112	392	808	1400	2920	2744	5544	2800	
L4															6
Signal routing through M6															
2	8	1	6	672	1312	504	1056	294	606	798	1662	1470	2974	1504	
4	16	2	7	1344	2624	1008	2112	343	707	1351	2819	2695	5443	2748	
4	8	0	0	672	1312	504	1056	0	0	504	1056	1176	2368	1192	
4	16	0	0	1344	2624	1008	2112	0	0	1008	2112	2352	4736	2384	

Table E.1: Timing estimates for different sizes of pipeline segments L1-L4 using the values from Table 4.3. The selected size of each pipeline segment is highlighted in green. Timing violations are indicated in red. For L4 the signals are buffered at the input of the combinational logic

F

Clock distribution techniques

The following sections discuss various techniques used to design the clock distribution in integrated circuits.

F.0.1. Clock Grid

Using the top metal layers, a clock grid is formed that is driven by multiple drivers along the boundary. This method ensures that the absolute delay to each clocking element is minimized. However, this comes at the cost of a larger power dissipation due to all the redundant interconnect, which only increases as the grid pitch becomes smaller. The main advantage of a clocking grid is the ability to easily change the layout and not have to adjust the clock routing, as it is accessible anywhere within the grid. The structure of a clocking grid is illustrated in Figure F.1.

F.0.2. Clock Tree

With a clock tree, the global clock is distributed from some point along branches to the leaf nodes. At the end of each branch, a buffer is placed to maintain steep edges on the clock. There are many variations of the tree topology, the most interesting of which are briefly presented.

Unconstrained tree

This clock distribution method is most commonly used in automated design flows. There are no limitations on the number of buffer stages that can be inserted; however, a cost function is used to minimize skew between the different leaf nodes of the network. This method is only suitable for use within small

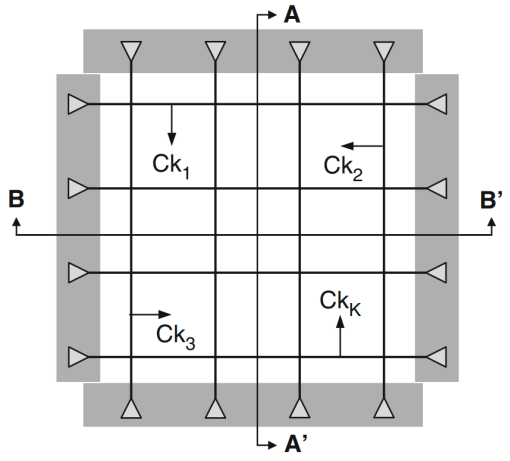


Figure F.1: Clock distribution using a clocking grid [60]

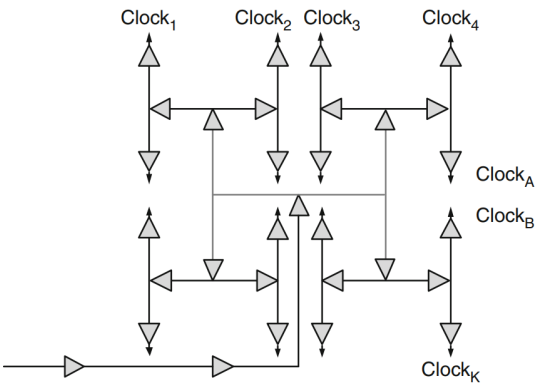


Figure F.2: Clock distribution using a balanced H-tree [60]

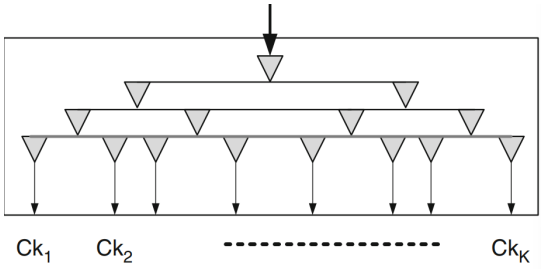


Figure F.3: Binary tree clock distribution with intermediate shorting [60]

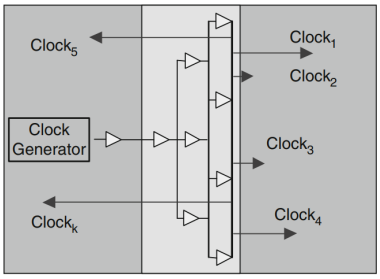


Figure F.4: Central spine clock distribution [60]

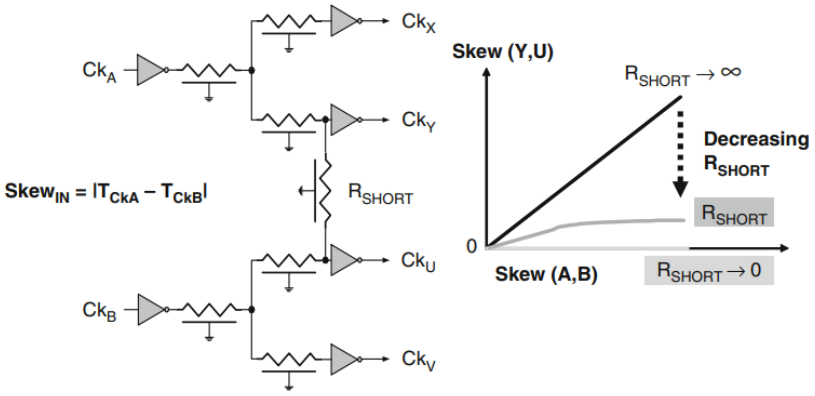


Figure F.5: Effect of shorting the clock signals from different buffer inputs in a binary tree [61]

functional blocks rather than in a large-scale design. [60]

Balanced tree

The balanced tree, also called an H-tree, distributes the global clock from a central point to the leaf nodes using balanced branches. The delays of these branches are matched to each other; therefore, the leaf nodes have minimal clock skew compared to each other. Figure F.2 shows a balanced H-tree clock network. Furthermore, in a tapered H-tree, the impedance of the T-junctions is matched to minimize reflections at higher clock rates. Balanced trees are ideal for distributing the clock in both horizontal and vertical dimensions simultaneously. [60] [44]

Binary tree

Binary trees can distribute the clock either in horizontal or vertical dimensions. The placement of the buffers is not as strict as in a balanced H-tree, allowing for more flexible integration within pre-existing circuits [61]. Generally, intermediate shortening is used to minimize the clock skew between different branches on the same level as shown in Figure F.3. The effectiveness of splicing branches together depends highly on the technology used and the resistance to interconnection seen between the shorted buffer inputs. The relation between the clock skew from the parent branches A and B to the child branches Y and U is shown in Figure F.5. The central spine is a specific implementation of a binary tree in which the clock is distributed over the entire chip in one direction as illustrated in figure F.4. The final branches are able to reach all parts of the IC; however, there will be a significant skew depending on the location of the leaf node.

F.0.3. Hybrid Clock distribution

Hybrid clock distribution typologies use a combination of the aforementioned typologies. In general, the higher-level clock distribution is implemented using a tree topology and the distribution at the final branches and leaf nodes is implemented using a grid [60] [44]. In this way, the skew between the

clocking elements is minimized, while the accessibility and flexibility of a clocking grid is still available. Furthermore, such a hybrid design allows for fine adjustment of the skew-to-power consumption relation by adjusting the grid pitch for each grid individually depending on the timing restrictions.

G

Matlab Scripts

G.1. Model

```
%% Solve
clear;

% 5 SE's with different energy at normal incidence
load("conf_se.mat");
SE = getModelResults(conf);
SE.conf = conf;
save("SE.mat", "SE");

% 5 SE's with 3 eV at different incidence
load("conf_se_d.mat");
SE_d = getModelResults(conf);
SE_d.conf = conf;
save("SE_d.mat", "SE_d");

% 5 BSE's with different incidence
load("conf_bse_d.mat");
BSE_d = getModelResults(conf);
BSE_d.conf = conf;
save("BSE_d.mat", "BSE_d");

% BSE with normal incidence
load("conf_bse.mat");
BSE = getModelResults(conf);
BSE.conf = conf;
save("BSE.mat", "BSE");

%% Combine SE and BSE
load("conf.mat");
load("BSE.mat");
load("SE.mat");
load("BSE_d.mat");
load("SE_d.mat");
load("customColormap.mat");
cmap = CustomColormap;


Total.NORM = BSE.RES + SE.RES;
```

```
eta_tot = sum(Total.NORM, "all");
Total.mult = (conf.E_tot / eta_tot);
Total.CLST = (BSE.CLST + SE.CLST) * Total.mult;
[Total.e_pc, Total.E_pc] = getPcError(Total.CLST,16,10);
[Total.e_crat, Total.E_crat] = getCratError(Total.CLST, 4,10);
Total.conf = SE.conf;

% Accounting for surfaces at an angle
Total.BSETilt = 0.5*BSE_d.RES + 0.5*BSE.RES;
Total.SETilt = 0.5*SE_d.RES + 0.5*SE.RES;
Total.RES = Total.BSETilt + Total.SETilt;
Total.CLST_tilt = ...
    0.5*(BSE.CLST + SE.CLST + BSE_d.CLST + SE_d.CLST) * Total.mult;
Total.grid = BSE.grid;


%% Shown FE model
load("conf1.mat");
[R,g] = modelEfield(conf.V_bias, conf.H, conf.W, conf.W);

figure(6);
hold on;
p = pdegplot(g,EdgeLabels="on");
p.LineWidth = 2;
p.Color = "#EDB120";
xgrid = -1*conf.W:(2*conf.W/18):conf.W;
ygrid = 0:(conf.H/7):conf.H;
[Xgrid, Ygrid] = meshgrid(xgrid, ygrid);
Egrid = R.interpolateElectricField(Xgrid, Ygrid);
p = quiver(Xgrid, Ygrid, reshape(Egrid.Ex,size(Xgrid)), ...
    reshape(Egrid.Ey,size(Ygrid)));
p.Alignment = "tail";
p.Color = "black";
xlabel("Distance from PE beam [m]")
ylabel("Distance from specimen surface [m]")
title("Electric Field between specimen and detector")
xlim([-1.1*conf.W 1.1*conf.W]);
ylim([-0.2*conf.H 1.2*conf.H]);



%% Plotting detector heatmap

f=figure(4);
f.Position = [100 100 600 400];
h=imagesc(Total.RES);
cmap(1,:) = [0,0,0];
colormap(cmap);
a=colorbar;
ylabel(a, "Intensity proportional to PE beam per detector");
title("Intensity per detector proportional to PE beam");
xlabel("Detector column index");
ylabel("Detector row index");
%h.XDisplayLabels = repmat({''}, size(h.XDisplayData));
%h.YDisplayLabels = repmat({''}, size(h.YDisplayData));
daspect([1 1 1]);
```

```
pbaspect([1 1 1]);
set(gca, 'ColorScale', 'log');

f=figure(5);
f.Position = [100 100 800 700];
%h=imagesc(Total.CLST);
heatmap(Total.CLST_tilt);
colormap(cmap);
%colorbar;
%ylabel(a, "Intensity proportional to PE beam per cluster");
title("\lambda per cluster for \tau_{\lambda}=2.5 ns");
xlabel("Cluster column index");
ylabel("Cluster row index");

%% Error graph CRAT
bcrat = 1:6;
for i = 1:6
    E_crat(i,1) = getCratError(Total.CLST_tilt,i,10);
end
for i = 1:6
    E_crat(i,2) = getCratError(Total.CLST_tilt,i,2);
end

f = figure(7);
f.Position = [100 100 1200 400];
subplot(1,2,1);
semilogy(bcrat, [E_crat(:,1) E_crat(:,2)], 'LineWidth', 4, Marker='o');
ylim([1E-10, 1E1]);
hold on;
yline(1E3/1E6, LineWidth=3, Label="Error Limit");
hold off
title("CRAT Error Probability depending on width" );
xlabel("Implementation width [bits]");
ylabel("Error probability");
legend("All clusters", "Clusters with lambda < 2");

% Error graph PC
bpc = 1:20;
for i = 1:20
    E_pc(i) = getPcError(Total.CLST_tilt,i,10);
end
for i = 1:20
    E_pc(i,2) = getPcError(Total.CLST_tilt,i,2);
end

subplot(1,2,2);
f = figure(7);
semilogy(bpc, [E_pc(:,1) E_pc(:,2)], 'LineWidth',4, Marker='o');
yline(1E3/1E6, LineWidth=3,  Label="Error Limit");
ylim([1E-10, 1E1]);
xlim([1 20]);
title("Pulse Counter Error Probability" + " depending on shift register
    size" );
xlabel("Shift register size");
ylabel("Error probability");
legend("All clusters", "Clusters with lambda < 2");
```

```
150
151 %% Error cluster
152 f=figure(6);
153 f.Position = [100 100 1600 700];
154 subplot(1,2,1);
155 heatmap(Total.E_pc);
156 colormap(cmap);
157 title("Error Probability for pulse counter" + "b=7, e_{tot} = " + Total.
       e_pc );
158 xlabel("Cluster column index");
159 ylabel("Cluster row index");
160
161 subplot(1,2,2);
162 heatmap(Total.E_crat);
163 colormap(cmap);
164 title("Error Probability for CRAT" + "b=4, e_{tot} = " + Total.e_crat );
165 xlabel("Cluster column index");
166 ylabel("Cluster row index");
167
168 %% Plotting exit angles
169 f = figure(5);
170 f.Position = [100 100 900 700];
171
172 ev = 1.602e-19;          % Electron Volt
173 me= 9.10938*10^-31;      % Electron mass
174 mag = sqrt(2.*conf.K(1).*ev./me);
175
176 % Specimen surface normal
177 quiver3(0,0,0,2*IV.n_hat(1,1),2*IV.n_hat(2,1),2*IV.n_hat(3,1), ...
178     Color="#0072BD",LineWidth=5, MaxHeadSize=1.5);
179
180 % PE beam
181 quiver3(0,0,2,0,0,-2, LineWidth=5, Color="red", MaxHeadSize=1.5);
182
183 % PE beam reflected
184 quiver3(0,0,0, 2*IV.r_refl(1,1), 2*IV.r_refl(2,1), 2*IV.r_refl(3,1),...
185     LineWidth=5, Color="#D95319", MaxHeadSize=1.5);
186
187 ax = scatter3(IV.v_X(:,1)./mag, ...
188     IV.v_Y(:,1)./mag, ...
189     IV.v_Z(:,1)./mag, ...
190     50, IV.eta(:,1), "filled");
191
192 legend("Surface normal", "PE beam", "Reflected PE beam", "Sample points")
193
194 colormap(cmap);
195 xlabel("x");
196 ylabel("y");
197 zlabel("z");
198 a =colorbar;
199 ylabel(a, "Proportional intensity");
200 pbaspect([1 1 1]);
201 daspect([1 1 1]);
202 hold on;
203
204 %% Plotting incidence locations
```

```
index = 1;       % Select which energy levels are plotted

f = figure(6);
f.Position = [100 100 900 700];
hold on

viscircles([0, 0], 0.5*conf.W)
scatter(DET.iX(:,index), DET.iY(:,index), 50, DET.iEta(:,index), "filled")
    ;

title("Incident electrons at detector" +  ...
    "K = " + conf.K(index) + " eV; V_{bias} = " + conf.V_bias);
xlabel("x-Distance from primary beam aperture [m]");
ylabel("y-Distance from primary beam aperture [m]");
zlabel("Proportionality factor  \eta]")
colormap(cmap);

xlim([-0.5*conf.W 0.5*conf.W]);
ylim([-0.5*conf.W 0.5*conf.W]);
clim([0 max(DET.iEta(:,index), [], "all")])
a =colorbar;
ylabel(a, "Proportional Intensity");
pbaspect([1 1 1]);
daspect([1 1 1]);

%% Individual Contour plots
load("customColormap");
cmap = CustomColormap;
cmap(1,:) = [0,0,0];

f = figure(7);
f.Position = [100 100 1400 850];
hold on
DET = SE_d;
conf = DET.conf;
m = size(DET.eta,3);

tiledlayout(2, ceil(m/2), "TileSpacing","tight");

for i = 1:m
    nexttile;

    contourf(DET.grid.X, DET.grid.Y, DET.eta(:,:,i));

    viscircles([0, 0], 0.5*conf.W);
    title(["K = " + conf.K(i) + " eV; " + ...
        "$V_{bias}$ = " + conf.V_bias + " V ", ...
        "$d_{wd}$ = " + conf.H*1000 + " mm"], Interpreter="latex");
    xlabel("x-Distance from aperture [m]");
    ylabel("y-Distance from aperture [m]");
    zlabel("Proportional intensity]")
    colormap(cmap);

    xlim([-0.5*conf.W 0.5*conf.W]);
    ylim([-0.5*conf.W 0.5*conf.W]);
    a =colorbar;
```

```
260     clim([0 max(DET.eta, [],"all")]);
261     ylabel(a, "Proportional intensity per detector");
262     pbaspect([1 1 1]);
263     daspect([1 1 1]);
264 end
265 hold off;
266 
267 %% Selective Contour plot
268 load("customColormap");
269 cmap = CustomColormap;
270 cmap(1,:) = [0,0,0];
271 f = figure(8);
272 f.Position = [100 100 600 400];
273 hold on
274 DET = SE;
275 conf = DET.conf;
276 
277 contourf(DET.grid.X, DET.grid.Y, DET.RES);
278 
279 viscircles([0, 0], 0.5*conf.W);
280 %title("$$\frac{\eta}{detector}$ " + "relative to PE beam", Interpreter="
    latex");
281 title("Proportional incidence for BSE's");
282 xlabel("x-Distance from primary beam aperture [m]");
283 ylabel("y-Distance from primary beam aperture [m]");
284 zlabel("Proportional intensity]")
285 colormap(cmap);
286 
287 xlim([-0.5*conf.W 0.5*conf.W]);
288 ylim([-0.5*conf.W 0.5*conf.W]);
289 a =colorbar;
290 clim([0 max(DET.RES, [],"all")]);
291 ylabel(a, "Intensity proportional to PE beam per detector");
292 pbaspect([1 1 1]);
293 daspect([1 1 1]);
294 
295 hold off;
296 
297 %% Total Contour plot
298 load("customColormap");
299 cmap = CustomColormap;
300 cmap(1,:) = [0,0,0];
301 f = figure(8);
302 f.Position = [100 100 600 400];
303 hold on
304 DET = Total;
305 plot = Total.NORM;
306 conf = DET.conf;
307 
308 contourf(DET.grid.X, DET.grid.Y, plot);
309 
310 viscircles([0, 0], 0.5*conf.W);
311 %title("$$\frac{\eta}{detector}$ " + "relative to PE beam", Interpreter="
    latex");
312 title("Proportional incidence for BSE's");
313 xlabel("x-Distance from primary beam aperture [m]");
```

```
ylabel(”y-Distance from primary beam aperture [m]”);
zlabel(”Proportional intensity]”)
colormap(cmap);

xlim([-0.5*conf.W 0.5*conf.W]);
ylim([-0.5*conf.W 0.5*conf.W]);
a =colorbar;
clim([0 max(plot, [],”all”)]);
set(gca, ”ColorScale”, ”log”);
ylabel(a, ”Intensity proportional to PE beam per detector”);
pbaspect([1 1 1]);
daspect([1 1 1]);

hold off;

%% Binned plot example
index = 1;       % Select which energy levels are plotted
DET = BSE;
conf = DET.conf;
f = figure(8);
t = tiledlayout(1,3,”TileSpacing”,”tight”,”Padding”,”none”);
f.Position = [100 100 1100 320];

nexttile;
scatter(DET.iX(:,1), DET.iY(:,1), 10, DET.iEta(:,1), ”filled”);
title(”Scattered results ” +  ...
    ”K = ” + conf.K(index) + ” eV”);
xlabel(”distance from aperture [m]”);
ylabel(”distance from aperture [m]”);
pbaspect([1 1 1]);
daspect([1 1 1]);
colorbar;

nexttile;
h = heatmap(DET.eta_binned(:,:,1));
title(”Binned results    ” +  ...
    ”K = ” + conf.K(index) + ” eV”);
xlabel(”column index”);
ylabel(”row index”);
xlim([28 78]);
ylim([28 78]);
h.XDisplayLabels = repmat({''}, size(h.XDisplayData));
h.YDisplayLabels = repmat({''}, size(h.YDisplayData));
colormap(cmap);

nexttile;
h = heatmap(DET.eta(:,:,1));
title(”Convolved results    ” +  ...
    ”K = ” + conf.K(index) + ” eV”);
xlabel(”column index”);
ylabel(”row index”);
xlim([25 75]);
ylim([25 75]);
h.XDisplayLabels = repmat({''}, size(h.XDisplayData));
h.YDisplayLabels = repmat({''}, size(h.YDisplayData));
colormap(cmap);
```

G.1.1. Configuration

```
%% ASML / HMI data
clear
conf.E_tot = 10;                % Total events peer 2.5 ns
conf.E_center = 6*(2.5/100);      % Event rate per 2.5 ns at center

conf.H = 15E-3;                % Working Distance
conf.V_bias = 6000;            % Bias voltage for electric field
conf.W = 20.48E-3;               % Width of detector
conf.E_pe = 1000;              % Primary electron landing energy (eV)

conf.Wcg = 5E-4;               % Center gap diameter
conf.Ld = 1.6E-4;              % Detector pitch
conf.d = ceil(conf.W/conf.Ld);% Diameter in detectors
conf.dclust = 16;              % Diameter of clusters in detectors

conf.n = 30;                   % Sample points over exit angle
conf.m = 128;                  % Sample points across detector
conf.frac = 0.1;
conf.o_se = 200;               % Sample points for SE path
conf.a_max = 85;               % Maximum escape angle

save(”conf.mat”, ”conf”);


%% conf 1 : 5 SE's
clear
load conf.mat;
conf.K_se = 5;                 % Kinetic energy of SE's
conf.K_bse = 0.85*conf.E_pe;       % Kinetic energy of BSE's
conf.eta_se = 1.6;
conf.eta_bse = 0.40;

conf.K = [1.5 2 3 4 5 7];          % Energy levels
conf.P = [420 470 400 300 240 150];      % Particles of this energy level
conf.eta =  [conf.eta_se, ...
    conf.eta_se, ...
    conf.eta_se, ...
    conf.eta_se, ...
    conf.eta_se, ...
    conf.eta_se];
conf.loc_PE = [ ...                        % PE beam scanning location
    0, 0;
    0, 0;
    0, 0;
    0, 0;
    0, 0;
    0, 0;
    ];
conf.norm = [...
    0, 0, 1; ...
    0, 0, 1; ...
    0, 0, 1; ...
    0, 0, 1; ...
    0, 0, 1; ...
    0, 0, 1 ...
```

```
56      ];
57
58  conf.linestyle = ["-", "-", "-","-","-", "-"];
59  conf.plot = [1 0 0 0 0 0];
60
61
62  save("conf_se.mat", "conf");
63
64  %% conf 2: 3 eV SE's for 5 surface normals
65  clear
66  load conf.mat;
67  conf.K_se = 5;                % Kinetic energy of SE's
68  conf.K_bse = 0.85*conf.E_pe;      % Kinetic energy of BSE's
69  conf.eta_se = 1.6;
70  conf.eta_bse = 0.45;
71
72  conf.K = [3 3 3 3];
73  conf.P = [400 400 400 400];
74  conf.eta =  [conf.eta_se conf.eta_se conf.eta_se conf.eta_se];
75  conf.loc_PE = [ ...                       % PE beam scanning location
76                  0, 0;
77                  0, 0;
78                  0, 0;
79                  0, 0];
80  conf.norm = [...
81      0, 1, 1; ...
82      1, 0, 1; ...
83      0, -1, 1; ...
84      -1, 0, 1; ...
85      ];
86
87  conf.labels = [...
88      "north"...
89      "east"...
90      "south"...
91      "west"];
92
93  conf.linestyle = ["-", "-","-","-","-", ];
94  conf.plot = [1 0 0 0 0 0];
95
96  conf.N_pe = (conf.E_tot)/(conf.eta_se+conf.eta_bse);
97  save("conf_se_d.mat", "conf");
98
99  %% conf 3: 5 BSE's in kardinal directions
100 clear
101 load conf.mat;
102 conf.K_se = 5;                % Kinetic energy of SE's (eV)
103 conf.K_bse = 0.85*conf.E_pe; % Kinetic energy of BSE's (eV)
104 conf.eta_se = 1.6;
105 conf.eta_bse = 0.40;
106
107 conf.K = [conf.K_bse conf.K_bse conf.K_bse conf.K_bse];
108 conf.P = [400 400 400 400];
109 conf.eta =  [conf.eta_bse conf.eta_bse ...
110     conf.eta_bse conf.eta_bse conf.eta_bse];
111 conf.loc_PE = [ ...                       % PE beam scanning location
```

```
                0, 0;
                0, 0;
                0, 0;
                0, 0];
conf.norm = [...
    0, 1, 1; ...
    1, 0, 1; ...
    0, -1, 1; ...
    -1, 0, 1; ...
    ];

conf.labels = [...
    "normal incidence"...
    "north"...
    "east"...
    "south"...
    "west"];

conf.linestyle = ["-", "-","-","-","-"];
conf.plot = [1 0 0 0 0];

conf.N_pe = (conf.E_tot)/(conf.eta_se+conf.eta_bse);
save("conf_bse_d.mat", "conf");

%% conf 4: BSE nomal
clear
load conf.mat;
conf.K_se = 2;                % Kinetic energy of SE's (eV)
conf.K_bse = 0.85*conf.E_pe; % Kinetic energy of BSE's (eV)
conf.eta_se = 1.6;
conf.eta_bse = 0.40;

conf.K = [conf.K_bse];
conf.P = [400];
conf.eta =  [conf.eta_bse];
conf.loc_PE = [ ...                        % PE beam scanning location
                0, 0];
conf.norm = [ ...
              0, 0, 1];

conf.linestyle = ["-"];
conf.plot = [1];

conf.N_pe = (conf.E_tot)/(conf.eta_se+conf.eta_bse);
save("conf_bse.mat", "conf");
```

G.2. SEM model

```
function DET = getModelResults(conf)
    IV = getElectronData(conf.n, conf.K, conf.eta,...
        -conf.a_max, conf.a_max, conf);
    [R,g] = modelEfield(conf.V_bias, conf.H, conf.W, conf.W);

    %% Solve
    f = figure(1);
```

```
    DET = getElectronPath(R, IV, f, conf);
    DET.IV = IV;

    %% Bin and convolve data
    k = size(IV.X, 2); % Number of energy levels

    % Create discrete array that reflects individual detectors
    % Locations of detector centers
    [DET.Mask, DET.grid.X, DET.grid.Y] = ...
        getDetectorMask(conf.W, conf.Ld, conf.Wcg, false);

    % Grid to which the data will be interpolated
    DET.grid.A = (conf.W/conf.m)^2;
    DET.RES = zeros(size(DET.grid.X(:,:,1)));
    Ptot = sum(conf.P, "all");

     % Convert scattered data to meshgrid data
    for j = 1:k
        this = DET.iN(j);       % Only include nonzero values
        [tmp, DET.A(j), DET.eta_binned(:,:,j)] = ...
            normConvIncedence(DET.iX(1:this,j), DET.iY(1:this,j), ...
            DET.iEta(1:this,j), DET.grid, conf);
        DET.eta(:,:,j) = max(tmp, 0);
        DET.RES = DET.RES + DET.eta(:,:,j) .* conf.P(j)/Ptot;
        clear("tmp");
    end

    DET.RES = DET.RES .* DET.Mask;
    DET.CLST = getClusterProb(DET.RES, conf.dclust);
end
```

G.2.1. Initial electron Data

```
function [SE] = getElectronData(n, K, eta, amin, amax, conf)
    % n : sample over angle
    % K : energy in eV [K1 K2 K3]
    % eta: eta of each energy level [e1 e2 e3]

    ev = 1.602e-19;          % Electron Volt
    me= 9.10938*10^-31;      % Electron mass

    dangle = amax / n;    % Angle step Phi
    danglepi = dangle/180 *pi;
    angles = linspace(0, amax, n)';
    anglespi = angles ./180 * pi;

    r = sind(angles);
    darcpi = danglepi;
    SE.rotsamples = floor((2*pi*r)./(darcpi));

    m = size(K,2);

    SE.X = zeros([sum(SE.rotsamples, "all")+1 m]);        % x coordinate
       of particle
    SE.Z = zeros([sum(SE.rotsamples, "all")+1 m]);        % y coordinate
       of particle
```

```
22      SE.Y = zeros([sum(SE.rotsamples, "all")+1 m]);          % z coordinate
            of particle
23      SE.eta = zeros([sum(SE.rotsamples, "all")+1 m]);        % eta values of
            each particle
24
25      SE.v_X = zeros([sum(SE.rotsamples, "all")+1 m]);          % x coordinate
             of particle
26      SE.v_Z = zeros([sum(SE.rotsamples, "all")+1 m]);          % y coordinate
             of particle
27      SE.v_Y = zeros([sum(SE.rotsamples, "all")+1 m]);
28
29      mag = sqrt(2.*K.*ev./me);
30      yx = mag .* sind(angles);
31
32
33      % Loop over rotsamples
34      previ = 1;
35      for i = 1:n
36          % Angle of velocity vector w.r.t. positive z-direction
37          drot = pi / SE.rotsamples(i);
38          if SE.rotsamples(i) == 0
39              SE.rotsamples(i) = 1;
40              rot = 0;
41          else
42              rot = linspace(0, (360 - 360/SE.rotsamples(i)), SE.rotsamples(
                    i))';
43          end
44
45          SE.v_X(previ:previ+SE.rotsamples(i)-1, :) = sind(rot) .* yx(i,:);
46          SE.v_Y(previ:previ+SE.rotsamples(i)-1, :) = cosd(rot) .* yx(i,:);
47          SE.v_Z(previ:previ+SE.rotsamples(i)-1, :) = zeros([previ-previ+SE.
                rotsamples(i), m]) + mag .* cosd(angles(i,:));
48
49          previ = previ+SE.rotsamples(i);
50          clear("v")
51      end
52
53      % https://math.stackexchange.com/questions/180418/calculate-rotation-
            matrix-to-align-vector-a-to-vector-b-in-3d
54      for j = 1:m
55
56          % Convert surface normal vector to unit vector
57          n_hat = conf.norm(j, :) / norm(conf.norm(j,:)); % local-z
58          SE.n_hat(:,j) = n_hat;
59
60          % Determine reflected PE beam
61          r = [0 0 -1]';
62          % Reflected PE beam for which sample points are defined
63          r_refl_0 = [0 0 1]';
64          r_refl = r - 2*(dot(r, n_hat')*n_hat');
65          SE.r_refl(:,j) = r_refl;
66
67          u = cross(r_refl_0, n_hat');
68          U = [0      -u(3)   u(2);
69               u(3)    0      -u(1);
70               -u(2)   u(1)   0];
```

```
        a = dot(r_refl, n_hat');

        I = [1 0 0;
             0 1 0;
             0 0 1];

        % Rotation matrix from [0 0 1] to n_hat
        R = I + U + U^2.*(1)/(1+a);
        SE.R(:, :, j) = R;

        % Rotate sample points to be centered on specimen normal
        A = [SE.v_X(:,j)';
             SE.v_Y(:,j)';
             SE.v_Z(:,j)'];

        At = R*A;

        SE.v_X(:,j) = At(1,:)';
        SE.v_Y(:,j) = At(2,:)';
        SE.v_Z(:,j) = At(3,:)';

        SE.eta(:, j) = ...
               max( eta(j) * ...
                    dot( r_refl'+zeros(size(SE.v_X,1),1), ...
                        ([SE.v_X(:,j), SE.v_Y(:,j), SE.v_Z(:,j)])./mag(j),
                            2) ...
                    ,0);
        SE.eta(:,j) = SE.eta(:,j) .* (eta(j) / sum(SE.eta(:,j), "all"));
    end

    SE.n_tot = previ-1;
end
```

G.2.2. Electric Field Model

```
function [R, g] = modelEfield(V, WD, W, WB)

    r = 0.5*W;
    rb = 0.5*WB;

    % General working distance is between 25 and 5 mm
    % non homogenous
    R = [2,8,   -1*W,  -r,  r,  1*W,    1*W, rb,    -rb,    -1*W,...
                1*WD, WD,  WD, 1*WD,   0,   0,     0,      0,   ]';
    g = decsg(R,'R',('R')');
    % homogenous

    % figure(1)
    % pdegplot(g,EdgeLabels="on")
    % xlim([-1.1*W 1.1*W])
    % ylim([-0.1*WD 1.1*WD])

    model = femodel(AnalysisType="electrostatic", Geometry=g);
    model.VacuumPermittivity = 8.8541878128E-12;
    model.MaterialProperties = materialProperties(RelativePermittivity=1);

```

```
    % Comment the following line for non ground at sample
    model.EdgeBC(7) = edgeBC(Voltage=0);
    model.EdgeBC(4) = edgeBC(Voltage=V);
    model.FaceLoad = faceLoad(ChargeDensity=5E-9);
    model = generateMesh(model);
    R = solve(model);

    % hold on;
    % p = pdeplot(R.Mesh,FlowData=[R.ElectricField.x R.ElectricField.y]);
    % p.Alignment = "tail";
    % axis equal
end
```

G.2.3. Electron path

```
function DET = getElectronPath(R, IV, h, conf,tspan)
    % E: Electric Field model
    %
    % IV is a struct:
    %   IV.X: starting X
    %   IV.Y: starting Y
    %   IV.Z: starting Z
    %   IV.v_X: starting velocity in X
    %   IV.v_Y: starting velocity in Y
    %   IV.v_Z: starting velocity in Z
    %
    % h: Handle to figure to plot on

    Ld = conf.Ld;
    W = conf.W;

    m = size(IV.X, 2); % Number of energy levels
    n = size(IV.X, 1); % Number of angle samples

    options_ode = odeset( ...
        'Events',@(t,w) odeEvents(t,w,conf.H));

    [t, w, te, we, ie]= ode45( @(t,w) ...
            odefun(t, w, R, conf.H, conf.W), [0 1e-7], ...
            [IV.X(IV.n_tot,1) ; IV.v_X(IV.n_tot,1); ...
            IV.Y(IV.n_tot,1) ; IV.v_Y(IV.n_tot,1); ...
            IV.Z(IV.n_tot,1) ; IV.v_Z(IV.n_tot,1)], ...
            options_ode);

    if ~exist("tspan", "var")
        tspan = [0 1.4*te];
    end

    map = colormap('turbo');
    se2map = 1+round(IV.eta ./ max(IV.eta, [], "all")*255);

    DET.X = zeros([n m]);
    DET.Y = zeros([n m]);

    figure(h);
    hold on;
```

```
42
43      for j = 1:m
44          this = 1;
45          for i = 1:n
46              [t, w, te, we, ie]= ode45( @(t,w) ...
47                  odefun(t, w, R, conf.H, conf.W), tspan, ...
48                  [IV.X(i,j) ; IV.v_X(i,j); ...
49                  IV.Y(i,j) ; IV.v_Y(i,j); ...
50                  IV.Z(i,j) ; IV.v_Z(i,j)], ...
51                  options_ode);
52
53              if(size(te,1) > 0)
54                  DET.iX(this,j) = interp1(t, w(:,1), te);
55                  DET.iY(this,j) = interp1(t, w(:,3), te);
56                  DET.iEta(this,j) = IV.eta(i,j);
57                  this = this+1;
58              end
59
60              % If there is a figure handle plot everything
61              if conf.plot(j) == 1
62                  color = map(se2map(i), :);
63                  plot3(w(:,1)',w(:,3)', w(:,5)', "Color", color, "LineStyle
                        " , conf.linestyle(1,j));
64              end
65          end
66
67          DET.iN(j) = this-1;
68      end
69
70      % Draw detector
71      hold on
72      dt.x = [-0.5*conf.W -0.5*conf.W 0.5*conf.W 0.5*conf.W];
73      dt.y = [0.5*conf.W  -0.5*conf.W -0.5*conf.W 0.5*conf.W];
74      dt.z = [0.9*conf.H 0.9*conf.H 0.9*conf.H 0.9*conf.H];
75      fill3(dt.x, dt.y, dt.z, 'black');
76
77      % Draw specimen
78      sp.x = [-0.2*conf.W -0.2*conf.W 0.2*conf.W 0.2*conf.W];
79      sp.y = [0.2*conf.W -0.2*conf.W -0.2*conf.W 0.2*conf.W];
80      sp.z = [0 0 0 0];
81      fill3(sp.x, sp.y, sp.z, [237 177 32]./256);
82      quiver3(0,0,0, ...
83          0.2*conf.H*conf.norm(1,1), ...
84          0.2*conf.H*conf.norm(1,2), ...
85          0.2*conf.H*conf.norm(1,3))
86
87      xlim([-0.75*conf.W 0.75*conf.W])
88      ylim([-0.75*conf.W 0.75*conf.W])
89      xlabel("x-Distance from Primary Beam [m]")
90      zlabel("Distance from Specimen [m]")
91      ylabel("y-Distance from Primary Beam [m]")
92      a =colorbar;
93      ylabel(a, "Intensity Proportional to Primary Beam [N]")
94      clim([0 max(IV.eta, [], "all")]);
95      title("Modelled behavior of PE (solid) and BSE (dashed); " + ...
96            "V_{bias} = " + conf.V_bias + "V ;");
```

```
97            %"E_{PE} = " + (E_pe/ev) + ...
98            %" eV; E_{SE} = " + (K_se/ev) + " eV; E_{BSE} = " + (K_bse/ev) +
                  " eV");
99        hold off
100   end
101
102   function [position, isterminal, direction] = odeEvents(t, w, H)
103       position = w(5) - H;
104       isterminal = 1;
105       direction = 0;
106   end
```

ODE Solver

```
1   function dw = odefun(t,w, R, H, W)
2       % w = [x, x', y, y', z, z']'
3       % dw = [x', x'', y', y'', z', z'']'
4
5       me= 9.10938*10^-31;      % Electron mass
6       q= -1.60276*10^-19;      % Electron charge
7
8       % Instead of defining 3d E-field a 2d field is used
9       % It is assumed this field is rotationally symmetrical around z-axis
10      xy = [w(1), w(3)];
11      r = norm(xy);
12      if r == 0
13          n_xy = [0 0];
14      else
15          n_xy = xy./r;
16      end
17
18      % Ensure electric field values can be interpolated
19      if(w(5) > H)
20          z = H;
21      elseif (w(5) < 0)
22          z = 0;
23      else
24          z = w(5);
25      end
26
27      if (r > W)
28          r = W;
29      end
30
31      E_z = interpolateElectricField(R, r, z).y;
32      E_xy = interpolateElectricField(R, r, z).x;
33
34      E_x = E_xy * n_xy(1);
35      E_y = E_xy * n_xy(2);
36
37      dw = [w(2); E_x*q / me; w(4); E_y*q / me; w(6); E_z*q / me];
38  end
39
40  % function dz = odefun(t,z, R, H, vx, vy)
41  %     me= 9.10938*10^-31;      % Electron mass
42  %     q= -1.60276*10^-19;      % Electron charge
```

```
%
%     x = t*vx;
%     y = t*vy;
%     r = sqrt(x^2+y^2);
%
%     if(z(1) < H)
%         E_z = interpolateElectricField(R, r, z(1)).y;
%     else
%         E_z = interpolateElectricField(R, r, H).y;
%     end
%
%     dz = [z(2); E_z*q / me];
% end
```

G.2.4. Sensor Matrix

```
function [M, X, Y] = getDetectorMask(width, pitch, gap_width, circle)
    %% PE beam gap
    d = ceil(width/pitch);
    imageSizeX = d;
    imageSizeY = d;
    [columnsInImage, rowsInImage] = meshgrid(1:imageSizeX, 1:imageSizeY);
    centerX = 0.5+imageSizeX / 2;
    centerY = 0.5+imageSizeY / 2;
    radius = gap_width/(2*pitch);
    circlePixels = (rowsInImage - centerY).^2 ...
        + (columnsInImage - centerX).^2 >= radius.^2;

    %% Sensor Matrix
    if circle
        centerX = 0.5+imageSizeX / 2;
        centerY = 0.5+imageSizeY / 2;
        radius = d/2;
        Det = (rowsInImage - centerY).^2 ...
            + (columnsInImage - centerX).^2 >= radius.^2;
        for i = 1:numel(Det)
            if(Det(i) == 0)
                Det(i) = 1;
            else
                Det(i) = 0;
            end
        end
    else
        Det = ones(size(rowsInImage));
    end
    % circlePixels is a 2D "logical" array. so AND it with center gap
    M = Det & circlePixels;

    % Locations of detector centers
    XX = -0.5*width:pitch:0.5*width-pitch;
    YY = 0.5*width:-pitch:-0.5*width+pitch;
    [X, Y] = meshgrid(XX, YY);
end
```

G.2.5. Smooth Results

```
function [RES,A, binned] = normConvIncedence(x, y, q, grid, conf)

    mi = min([x; y]);
    ma = max([x; y]);
    dl = conf.Ld;           % distance between bins equal to detecor size
    d = (ma-mi)/(2*conf.n); % average distance between two particles
    A = dl^2';              % Area of each bin

    % The size of the kernel is defined so that at least two particles are
    % enclosed at all times
    k = max(floor(2*(floor(4*(d/dl))/2))+1, 5);

    % Extend borders so that there is enough zero padding for convolution
    mi = min(grid.X,[],"all") - 30*dl;
    ma = max(grid.Y,[],"all") + 30*dl;

     % Interpolate results to predefined x values but limit this grid to
     % the size of the detector with some margin
    [X, Y] = meshgrid((mi: dl: ma), (ma:-dl:mi));
    binned = zeros(size(X));

    % Binning to process incidence within same region
    % row
    for i = 1:size(X,1)-1
        % column
        for j = 1:size(X,2)-1

            % Get index of particles within bin range
            ix_g = find(x > X(i,j));
            ix_l = find(x <= X(i,j+1));
            iy_g = find(y < Y(i,j));
            iy_l = find(y >= Y(i+1,j));

            % Get the intersection of the upper and lower x,y indexes
            if(size(ix_g,1)>0 && size(ix_l,1) && size(iy_g,1)>0 && size(
                iy_l,1))
                ix = intersect(ix_g, ix_l);
                iy = intersect(iy_g, iy_l);
                i_all = intersect(ix, iy);

                if(size(i_all,1) ~= 0 && size(i_all,2) ~= 0)
                    tot = 0;

                    % Loop over incident electrons and sum their eta
                    for l = 1:size(i_all,1)
                        tot = tot + q(i_all(l));
                    end
                    clear("i_all");
                    binned(i,j) = tot;
                end
            end
        end
    end

    % Now perform a convolution to smooth out the incdence values
```

```
    % The convolution matrix size depends on the value of m,
    % The larger m is, the larger the convolution matrix must be
    % Can be improved by creating a circular kernel instead of box
    imageSizeX = k;
    imageSizeY = k;
    [columnsInImage, rowsInImage] = meshgrid(1:imageSizeX, 1:imageSizeY);

    % Next create the circle in the image.
    centerX = 0.5 + imageSizeX / 2; % Wherever you want.
    centerY = 0.5 + imageSizeY / 2;
    radius = ceil(k/2);
    circlePixels = (rowsInImage - centerY).^2 ...
        + (columnsInImage - centerX).^2 <= radius.^2;

    s = sum(circlePixels,"all");
    K = (1/(s)) .* circlePixels;

    % This makes it so the values are the average proportional incidence
    % per k*dl x k*dl square
    smoothed1 = conv2(binned, K, "same");

    % The result usually still contains a lot of high frequency components
    % so it is convolved again, but with a smaller kernel
    imageSizeX = 5;
    imageSizeY = 5;
    [columnsInImage, rowsInImage] = meshgrid(1:imageSizeX, 1:imageSizeY);

    % Next create the circle in the image.
    centerX = 0.5 + imageSizeX / 2; % Wherever you want.
    centerY = 0.5 + imageSizeY / 2;
    radius = ceil(5/2);
    circlePixels = (rowsInImage - centerY).^2 ...
        + (columnsInImage - centerX).^2 <= radius.^2;

    s = sum(circlePixels,"all");
    K = (1/(s)) .* circlePixels;
    smoothed2 = conv2(smoothed1, K, "same");

    % Now interpolate the result to the predefined grid
    RES = interp2(X, Y, smoothed2, grid.X, grid.Y, "linear");
end
```

G.2.6. Clustering

```
function s = sumSubset(X, tl, br)
% sum the subset of 2d array X contained within the rectangle with top
   left
% corner at tl and bottom right corner at br.
% tl = [row column]
% br = [row column]
    if tl(1) >= br(1) || tl(2) >= br(2)
        error("Invalid arguments tl or br");
    end

    s = 0;

```

```
    % row loop
    for i = tl(1):br(1)
        % column loop
        for j = tl(2):br(2)
            s = s + X(i, j);
        end
    end
end
```

```
function C = getClusterProb(Pd, c_size)
    % vector containing cluster sizes from outside to inside
    r = ceil(size(Pd,1)/c_size);
    c = ceil(size(Pd, 2)/c_size);
    C = zeros([r c]);

    % row loop
    for i = 1:r
        % column loop
        for j = 1:c
            tl = [(1 + c_size * (i-1)) (1 + c_size * (j-1))];
            bl = [(c_size * i) (c_size * j)];

            C(i, j) = sumSubset(Pd, tl, bl);
        end
    end
end
```

G.2.7. Poisson binomial distribution

```
function E = getTotalErrorProb(e)
    n = numel(e);
    e1 = zeros(n);
    for i = 1:size(e,1)
        for j = 1:size(e,2)
            e1(i+size(e,1)*(j-1))=e(i,j);
        end
    end

    % Implementation of FFT algorithm proposed by Biscarri et. al.
    M = 2^nextpow2(2*n);
    omega = exp(-2i * pi / M);
    A = ones(1,M);

    for i = 1:n
        A = A .* (1-e1(i)+e1(i)*omega.^(0:M-1));
    end

    ipdf = (ifft(A));
    pdf = abs(real(ipdf(1:n+1)));
    if pdf(1) > 1
        E=0;
    else
        E = 1-pdf(1);
    end
end
```

```
function [E, e] = getCratError(lambda, b, lim)
    % lambda:= Matrix containing lambda values of clusters
    % b     := width of CRAT
    lambda(lambda>lim) = 0;

    e = zeros(size(lambda));
    for i = 1:size(lambda,1)
        for j = 1:size(lambda,2)
            of=0;
            for n = 0:2^b
                of = of + (lambda(i,j))^n / factorial(n);
            end
            e(i,j) = 1 - exp(-lambda(i,j))*of;
        end
    end

    E = getTotalErrorProb(e);
end
```

```
function [E, e, e_of, e_ol] = getPcError(lambda, b, del, t)
    % lambda:= Matrix containing lambda values of clusters
    % b     := positions in PC shift register
    % del   := Delay of register cell
    % t     := overlap threshold
    T = 2.5E-9;
    tau = del+t;

    e = zeros(size(lambda));
    e_of = zeros(size(lambda));
    e_ol = zeros(size(lambda));

    for i = 1:size(lambda,1) % loop over cluster rows
        for j = 1:size(lambda,2) % loop over cluster columns

            rclst = lambda(i,j)/T;

            eps_0 = 0;
            for n = 0:1
                eps_0 = eps_0 + ((rclst*t)^n /(factorial(n)));
            end
            eps_0 = 1-exp(-rclst*t)*eps_0;

            eps_1 = 0;
            for n = 0:2
                eps_1 = eps_1 + (((rclst*tau)^n)/(factorial(n)));
            end
            eps_1 = (1-exp(-rclst*tau)*eps_1);
            eps_1 = eps_1 * (exp(-rclst*t)*rclst*t);
            e_ol(i,j) = eps_0+eps_1;

            of = 0;
            for n = 0:b
                of = of + lambda(i,j)^n / factorial(n);
            end
            e_of(i,j) = 1-exp(-lambda(i,j))*of;
```

```
37
38              e(i,j) = e_of(i,j)+e_ol(i,j);
39          end
40      end
41
42      E = getTotalErrorProb(e);
43  end
```

G.3. PDK Characterization

G.3.1. Inverter characteristics as repeater

```
1  % Author: Alex Smit
2
3  %% Import and prepare Data
4  clear;
5
6  L=998;
7  C=1.73135e-13;
8  Rsheet = 0.210;
9  R=3928.80;
10
11
12 load(”inv200u.mat”);
13 indx_hl = [find(inv200u{:,1}>100E-12,1) find(inv200u{:,1}>1E-9,1)];
14 indx_lh = [find(inv200u{:,1}>1E-9,1) find(inv200u{:,1}>1.9E-9,1)];
15 crit.hl = inv200u{indx_hl(1):indx_hl(2),2:12};
16 crit.t_hl = inv200u{indx_hl(1):indx_hl(2),1}-0.1075E-9;
17 crit.lh = inv200u{indx_lh(1):indx_lh(2),2:12};
18 crit.t_lh = inv200u{indx_lh(1):indx_lh(2),1}-1.0075E-9;
19 load(”inv1000u.mat”);
20 data.leg = inv1000u.Properties.VariableNames(2:12);
21 data.bar = categorical(inv1000u.Properties.VariableNames(2:12), 'Ordinal',
       true);
22 data.bar = categorical(data.bar, data.bar, 'Ordinal', true);
23 indx_hl = [find(inv1000u{:,1}>6E-9,1) find(inv1000u{:,1}>11.95E-9,1)];
24 indx_lh = [find(inv1000u{:,1}>12E-9,1) find(inv1000u{:,1}>17.78E-9,1)];
25 data.hl = inv1000u{indx_hl(1):indx_hl(2),2:12};
26 data.t_hl = inv1000u{indx_hl(1):indx_hl(2),1}-6.0075E-9;
27 data.lh = inv1000u{indx_lh(1):indx_lh(2),2:12};
28 data.t_lh = inv1000u{indx_lh(1):indx_lh(2),1}-12.0075E-9;
29
30 for i = 1:11
31     data.tp_hl(1,i) = data.t_hl((find(data.hl(:, i)<0.55,1)));
32     data.tp_lh(1,i) = data.t_lh((find(data.lh(:, i)>0.55,1)));
33     crit.tp_hl(1,i) = crit.t_hl((find(crit.hl(:, i)<0.55,1)));
34     crit.tp_lh(1,i) = crit.t_lh((find(crit.lh(:, i)>0.55,1)));
35 end
36
37 %% Calculations from data
38 load(”inv.mat”);     % Structure containing data extracted from simulations
39
40 tp1 = 14.24E-12;     % delay for fanout of 1 extracted from simulations
41 tpwire = 0.38*R*C;   % Distributed RC delay of interconnect wire
42 mopt = sqrt(tpwire./tp1);
43 Lcrit = L./mopt;
```

```
44
45 tpbuff.hl = L./Lcrit .* inv.tcrit_hl;           % delay with buffering
46 tpbuff.lh = L./Lcrit .* inv.tcrit_lh;           % delay with buffering
47 EDP = inv.Epn.*(inv.tcrit_hl+inv.tcrit_lh)*0.5; % Energy delay product
48 EpU = inv.Epn./200;                             % Energy per um
49
50
51 %% Plot Results
52 figure('Position',[100 100 1200 600]);
53 hold on;
54
55 % High-to-Low simulation results
56 subplot(2,3,1)
57 plot(data.t_hl, data.hl, 'LineWidth',2);
58 yline(0.55, '--');
59 legend(data.leg);
60 ylim([-0.1 1.2]);
61 xlabel("Time [ns]");
62 ylabel("Voltage [V]");
63 title("High to Low transistion [t_r = 15 ps]");
64
65 % Low-to-High simulation results
66 subplot(2,3,4)
67 plot(data.t_lh, data.lh, 'LineWidth',2);
68 yline(0.55, '--');
69 legend(data.leg);
70 ylim([-0.1 1.2]);
71 xlabel("Time [ns]");
72 ylabel("Voltage [V]");
73 title("Low to High transistion [t_r = 15 ps]");
74
75 % Propagation delay from simulation results
76 subplot(2,3,2)
77 bar(data.bar,[data.tp_hl' data.tp_lh']*1E9);
78 ylim([0 2.1]);
79 ylabel("propagation delay [ns]");
80 xlabel("Drive Strength");
81 title("Unbuffered delay over " + L + " \mum");
82 legend({"High-to-Low" "Low-to-High"});
83
84 % Propagation delay from simulation results, for calculated critical
      length
85 subplot(2,3,5)
86 bar(data.bar, [crit.tp_hl' crit.tp_lh']*1e9);
87 ylim([0 0.4]);
88 ylabel("Delay [ns]");
89 xlabel("Drive Strength");
90 title(["Critical Delay"]);
91 legend({"High-to-Low" "Low-to-High"});
92
93 % Energy-Delay-Product and Energy per Um
94 subplot(2,3,3)
95 yyaxis left
96 bar(categorical("D"+string(inv.D),string(data.bar), 'Ordinal', true),[EDP
      0*EDP]*1E24);
97 ylabel("EDP [fJns]");
```

```
xlabel(”Drive Strength”);
title([”Energy delay product”; ”over critical length”]);
yyaxis right
bar(categorical(”D”+string(inv.D),string(data.bar), 'Ordinal', true),[0*
    EpU EpU]*1E15);
ylim([0.1 0.115]);
ylabel(”Energy [fJ / \mum]”);

% Calculated propagation delay for buffered wires
subplot(2,3,6)
bar(categorical(”D”+string(inv.D),string(data.bar), 'Ordinal', true), [
    tpbuff.hl tpbuff.lh]*1E9, 1);
legend({”Buffered HL” ”Buffered LH”});
ylabel(”Delay [ns]”);
xlabel(”Drive Strength”);
title(”Buffered delay over ” + L + ” \mum”);
```

G.3.2. FA characteristics

```
%% FA Worst Case
clear;
% TSMC N40 PDK datasheet values (ff)
load(”fa.mat”);
load(”../General Layout/inv.mat”);

% Calc delay with inverter at output
tp_fa_inv = FA.ti*ones(size(inv.Cg')) + FA.F*(inv.Cg');
% Calc delay with other FA at output
tp_fa_fa = FA.ti*ones(size(FA.D')) + FA.F*(FA.Cg');
% Calc delay with DDR latch at the output
tp_fa_l = FA.ti+ FA.F*2*0.0005725;

figure (1)
% subplot(3,1,1)
% heatmap(inv.D, FA.D, tp_fa_inv*1e3);
% xlabel(”Inverter Drive Strength”);
% ylabel(”FA Drive strength”);
% title(”FA delay [ps] depending on inverter drive strength at its output
    ”);

subplot(2,1,1)
heatmap(”D1 - D4”, FA.D, tp_fa_l*1e3);
xlabel(”DETDFF Drive Strength”);
ylabel(”FA Drive strength”);
title(”FA delay [ps] with a DETDFF (D1-D4) at its output”);

subplot(2,1,2)
heatmap(FA.D, FA.D, tp_fa_fa*1e3);
xlabel(”FA Drive Strength”);
ylabel(”FA Drive strength at output”);
title(”FA delay [ps] depending on FA drive strength at its output”);
```

G.3.3. Power results

```
clear;
```

```
cmin = 0;
cmax = 61;

mm15 = [
    0, 0,       0,      0,      0,      0,      0,      0;
    0, 0,       0.003,  0.027,  0.027,  0.003,  0,      0;
    0, 0.003,   0.083,  0.146,  0.146,  0.083,  0.003,  0;
    0, 0.027,   0.145,  2.08,   2.08,   0.145,  0.027,  0;
    0, 0.027,   0.145,  2.08,   2.08,   0.145,  0.027,  0;
    0, 0.003,   0.083,  0.146,  0.146,  0.083,  0.003,  0;
    0, 0,       0.003,  0.027,  0.027,  0.003,  0,      0;
    0, 0,       0,      0,      0,      0,      0,      0;
];

mm5 = [
    0, 0,       0,      0,      0,      0,      0,      0;
    0, 0,       0,      0,      0,      0,      0,      0;
    0, 0,       0,      0.001,  0.001,  0,      0,      0;
    0, 0,       0.001,  2.5,    2.5,    0.001,  0,      0;
    0, 0,       0.001,  2.5,    2.5,    0.001,  0,      0;
    0, 0,       0,      0.001,  0.001,  0,      0,      0;
    0, 0,       0,      0,      0,      0,      0,      0;
    0, 0,       0,      0,      0,      0,      0,      0;
];


%% L1
l1_mm15 = [
    pc(0),      pc(0),     pc(0),      pc(0),      pc(0),      pc(0),
            pc(0),    pc(0);
    pc(0),      pc(0), pc(0.003),  pc(0.027),  pc(0.027),  pc(0.003),
            pc(0),    pc(0);
    pc(0),  pc(0.003), pc(0.083),  pc(0.146),  pc(0.146),  pc(0.083),  pc
        (0.003),    pc(0);
    pc(0),  pc(0.027), pc(0.145), crat(2.08), crat(2.08),  pc(0.145),  pc
        (0.027),    pc(0);
    pc(0),  pc(0.027), pc(0.145), crat(2.08), crat(2.08),  pc(0.145),  pc
        (0.027),    pc(0);
    pc(0),  pc(0.003), pc(0.083),  pc(0.146),  pc(0.146),  pc(0.083),  pc
        (0.003),    pc(0);
    pc(0),      pc(0), pc(0.003),  pc(0.027),  pc(0.027),  pc(0.003),
            pc(0),    pc(0);
    pc(0),      pc(0),     pc(0),      pc(0),      pc(0),      pc(0),
            pc(0),    pc(0);
];

load("customColormap.mat");
cmap = CustomColormap;

f=figure(1);
tiledlayout(1,2,"TileSpacing","tight","Padding","none");
f.Position = [100 100 1100 500];

nexttile;
h = heatmap(l1_mm15);
h.ColorLimits = [cmin cmax];
```

```
colormap(cmap)
title('Power consumption at L1 pipeline level [mW]');
xlabel("L1 column index");
ylabel("L1 row index");

nexttile;
h = heatmap(crat(mm15));
h.ColorLimits = [cmin cmax];
colormap(cmap)
title('Power consumption at L1 pipeline level when only critical adder trees are used [mW]');
xlabel("L1 column index");
ylabel("L1 row index");

% [W]
L1 = sum(l1_mm15,"all")*1e-3;
L1_crat = sum(crat(mm15), "all")*1e-3;


%% L2
% l2 lambdas
for i = 1:4
    for j = 1:4
        ii = 2*(i-1)+1;
        jj = 2*(j-1)+1;
        l2_mm15(i,j) = mm15(ii, jj)+mm15(ii+1, jj)+mm15(ii, jj+1)+mm15(ii+1, jj+1);
        l2_mm15_power(i,j) = (  l2_signals(mm15(ii, jj))+ ...
                                l2_signals(mm15(ii+1, jj))+ ...
                                l2_signals(mm15(ii, jj+1))+ ...
                                l2_signals(mm15(ii+1, jj+1)));
    end
end

% power from simulations
load("l2_power.mat")
l2_mm15_power = l2_mm15_power + interp1(l2power.RMS_X, l2power.RMS_Y, l2_mm15);

load("customColormap.mat");
cmap = CustomColormap;

f=figure(2);
f.Position = [100 100 600 500];
h = heatmap(l2_mm15_power*1000);
h.ColorLimits = [cmin cmax];
colormap(cmap)
title('Power consumption at L2 pipeline level [mW]');
xlabel("L2 column index");
ylabel("L2 row index");

% [W]
L2 = sum(l2_mm15_power,"all");


%% L3
```

```
% l3 lambdas
for i = 1:2
    for j = 1:2
        ii = 2*(i-1)+1;
        jj = 2*(j-1)+1;
        l3_mm15(i,j) =  l2_mm15(ii, jj)+ ...
                        l2_mm15(ii+1, jj)+ ...
                        l2_mm15(ii, jj+1)+ ...
                        l2_mm15(ii+1, jj+1);
        l3_mm15_power(i,j) = (  l3_signals(l2_mm15(ii, jj))+ ...
                                l3_signals(l2_mm15(ii+1, jj))+ ...
                                l3_signals(l2_mm15(ii, jj+1))+ ...
                                l3_signals(l2_mm15(ii+1, jj+1)));
    end
end

% power from simulations
load("l3_l4_power.mat")
l3_mm15_power = l3_mm15_power + interp1(L3L4power.RMS_X, L3L4power.RMS_Y,
    l3_mm15);

load("customColormap.mat");
cmap = CustomColormap;

f=figure(3);
f.Position = [100 100 600 500];
h = heatmap(l3_mm15_power*1000);
h.ColorLimits = [cmin cmax];
colormap(cmap)
title('Power consumption at L3 pipeline level [mW]');
xlabel("L3 column index");
ylabel("L3 row index");

L3 = sum(l3_mm15_power,"all");

%% L4
repeaters = 6*8;
d24_power = 0.1129; %mW
l4_sig_power = repeaters*d24_power*1e-3;
l4_mm15 = sum(l3_mm15, "all");
l4_mm15_power = interp1(L3L4power.RMS_X, L3L4power.RMS_Y, l4_mm15);
L4 = l4_mm15_power + 4*4*l4_sig_power;


%% Total
labels = ["L1", "L2", "L3", "L4", "Clock"];
val = [L1, L2, L3, L4, 0.446];

f=figure(4);
t = tiledlayout(1,4);
%t.ColumnWidth = {'1x', '0.5x', '1x'};
f.Position = [100 100 800 500];

nexttile([1 2]);
b = bar(labels, val);
title('RMS power of sections');
```

```
158 ylabel(”RMS power [W]”);
159 % Get bar centers
160 xt = b.XEndPoints;
161 yt = [0.25 0.05 0.05 0.13 0.25];
162 % Write values inside the bars
163 text(xt, yt, string(round(val*1000,1))+” mW”, ...
164     'HorizontalAlignment','center', ...
165     'VerticalAlignment','middle', ...
166     'Color','black', ...
167     'FontWeight','bold', ...
168     'Rotation', 90, ...
169     'FontSize', 12);
170
171 nexttile;
172 b=bar([”Implemented” ”CRAT-only”], [L1, L1_crat]);
173 title('Comparison of L1 configurations');
174 ylabel(”RMS power [W]”);
175 ax = gca;
176 ax.YLimMode = 'manual';
177 ax.YLim = [0 5];
178 % Get bar centers
179 xt = b.XEndPoints;
180 yt = [0.7 2.5];
181 % Write values inside the bars
182 text(xt, yt, string(round([L1, L1_crat],1))+” W”, ...
183     'HorizontalAlignment','center', ...
184     'VerticalAlignment','middle', ...
185     'Color','black', ...
186     'FontWeight','bold', ...
187     'Rotation', 90, ...
188     'FontSize', 12);
189 b.CData = [
190     0 0.4470 0.7410;
191     0.8500 0.3250 0.0980;
192 ];
193 b.FaceColor = 'flat';
194
195 nexttile;
196 ASIC = L1 + L2 + L3 + L4 + 0.446;
197 ASIC2 = L1_crat + L2 + L3 + L4 + 0.446;
198 b=bar([”Implemented”, ”CRAT-only”], [ASIC, ASIC2]);
199 title('ASIC RMS power');
200 ylabel(”RMS power [W]”);
201 ax = gca;
202 ax.YLimMode = 'manual';
203 ax.YLim = [0 5];
204 % Get bar centers
205 xt = b.XEndPoints;
206 yt = [0.45 2.5];
207 % Write values inside the bars
208 text(xt, yt, string(round([ASIC, ASIC2],1))+” W”, ...
209     'HorizontalAlignment','center', ...
210     'VerticalAlignment','middle', ...
211     'Color','black', ...
212     'FontWeight','bold', ...
213     'Rotation', 90, ...
```

```
    'FontSize', 12);
b.CData = [
    0 0.4470 0.7410;
    0.8500 0.3250 0.0980;
];
b.FaceColor = 'flat';


%% Functions
% pc power = 0.05 mW / 2.5 lambda
% x0 = 1.26
% pc pwr = 1.26+0.02*lambda
function p = pc(lambda)
    p = 1.26+0.02*lambda + l1_signals(lambda);
end

% crat power = 0.5+0.67*lambda + clk_pwr
function p = crat(lambda)
    clk_pwr = 58;
    p = 0.5+0.67*lambda+clk_pwr + l1_signals(lambda);
end

function p = l1_signals(lambda)
    d8_power = 0.0214; % mW
    repeaters = 14;
    p = d8_power*repeaters*lambda;
end

function p = l2_signals(lambda)
    d8_power = 0.0214;
    repeaters = 16;
    l2_sig = repeaters*d8_power; % W
    p = log2(lambda+1);
    % if(lambda < 1)
    %     p = lambda*l2_sig;
    % elseif((1<lambda)&&(lambda<2))
    %     p = 2*l2_sig;
    % else
    %     p = 3*l2_sig;
    % end
    % convert to W
    p = p*1e-3;
end

function p = l3_signals(lambda)
    d24_power = 0.1129; %mW
    repeaters = 8;
    l3_sig_power = d24_power*repeaters;
    p = log2(lambda+1);
    % if(lambda<1)
    %     p = lambda*l3_sig_power;
    % elseif((1<lambda)&&(lambda<2))
    %     p = 2*l3_sig_power;
    % elseif((2<lambda)&&(lambda<4))
    %     p = 3*l3_sig_power;
    % elseif((4<lambda)&&(lambda<8))
```

```
    %     p = 4*l3_sig_power;
    % else
    %     p = 5*l3_sig_power;
    % end

    % convert to W
    p = p*1e-3;
end
```

G.3.4. Error rate results

```
clear
mm15 = [
    0, 0,       0,      0,      0,      0,      0,      0;
    0, 0,       0.003,  0.027,  0.027,  0.003,  0,      0;
    0, 0.003,   0.083,  0.146,  0.146,  0.083,  0.003,  0;
    0, 0.027,   0.145,  0,      0,      0.145,  0.027,  0;
    0, 0.027,   0.145,  0,      0,      0.145,  0.027,  0;
    0, 0.003,   0.083,  0.146,  0.146,  0.083,  0.003,  0;
    0, 0,       0.003,  0.027,  0.027,  0.003,  0,      0;
    0, 0,       0,      0,      0,      0,      0,      0;
];
mm15_full = [
    0, 0,       0,      0,      0,      0,      0,      0;
    0, 0,       0.003,  0.027,  0.027,  0.003,  0,      0;
    0, 0.003,   0.083,  0.146,  0.146,  0.083,  0.003,  0;
    0, 0.027,   0.145,  2,      2,      0.145,  0.027,  0;
    0, 0.027,   0.145,  2,      2,      0.145,  0.027,  0;
    0, 0.003,   0.083,  0.146,  0.146,  0.083,  0.003,  0;
    0, 0,       0.003,  0.027,  0.027,  0.003,  0,      0;
    0, 0,       0,      0,      0,      0,      0,      0;
];
mm5 = [
    0, 0,       0,      0,      0,      0,      0,      0;
    0, 0,       0,      0,      0,      0,      0,      0;
    0, 0,       0,      0.001,  0.001,  0,      0,      0;
    0, 0,       0.001,  0,      0,    0.001,    0,      0;
    0, 0,       0.001,  0,      0,    0.001,    0,      0;
    0, 0,       0,      0.001,  0.001,  0,      0,      0;
    0, 0,       0,      0,      0,      0,      0,      0;
    0, 0,       0,      0,      0,      0,      0,      0;
];
mm5_full = [
    0, 0,       0,      0,      0,      0,      0,      0;
    0, 0,       0,      0,      0,      0,      0,      0;
    0, 0,       0,      0.001,  0.001,  0,      0,      0;
    0, 0,       0.001,  2.5,    2.5,    0.001,    0,      0;
    0, 0,       0.001,  2.5,    2.5,    0.001,    0,      0;
    0, 0,       0,      0.001,  0.001,  0,      0,      0;
    0, 0,       0,      0,      0,      0,      0,      0;
    0, 0,       0,      0,      0,      0,      0,      0;
];


low = 0;
high = 10000;
```

```
th = 1000;
N = 256;   % total number of colors
cmap = zeros(N,3);

% Fraction of colormap below threshold
f = (th - low) / (high - low);
k = round(f * N);

% Lower part: summer colormap
cmap_low = summer(k);

% Upper part: red
cmap_high = repmat([1 0 0], N-k, 1);

% Combine
cmap = [cmap_low; cmap_high];



load("pcib_ppm.mat");
load("ppm_calc.mat");
load("lambda_calc.mat");
pc_ppm_15_calc = ceil(interp1(lambda, ppm, mm15));
pc_ppm_15 = ceil(interp1(pcipppm.ppmpcib_X, pcipppm.ppmpcib_Y, mm15, "
    pchip"));
pc_ppm_5_calc = ceil(interp1(lambda, ppm, mm5));
pc_ppm_5 = ceil(interp1(pcipppm.ppmpcib_X, pcipppm.ppmpcib_Y, mm5, "pchip
    "));

T_15 = sum( (pc_ppm_15_calc.*mm15_full)./sum(mm15_full, "all"), "all");
T_5 = sum( (pc_ppm_5_calc.*mm5_full)./sum(mm5_full, "all"), "all");



%load("customColormap.mat");
%cmap = CustomColormap;

f=figure(3);
t = tiledlayout(2,2,"TileSpacing","tight","Padding","none");
f.Position = [100 100 650 700];

nexttile;
h = heatmap(pc_ppm_15);
h.ColorLimits = [low high];
h.ColorbarVisible="off";
colormap(cmap)
title({'Simulated error rates of' 'L1 pulse counters at 15mm [ppm]'});
xlabel("L1 column index");
ylabel("L1 row index");

nexttile;
h = heatmap(pc_ppm_15_calc);
h.ColorLimits = [low high];
h.ColorbarVisible="off";
colormap(cmap)
title({'Calculated error rates of' 'L1 pulse counters 15mm [ppm]'});
```

```
100 xlabel(”L1 column index”);
101 ylabel(”L1 row index”);
102
103 nexttile;
104 h = heatmap(pc_ppm_5);
105 h.ColorLimits = [low high];
106 h.ColorbarVisible=”off”;
107 colormap(cmap)
108 title({'Simulated error rates of' 'L1 pulse counters 5mm [ppm]'});
109 xlabel(”L1 column index”);
110 ylabel(”L1 row index”);
111
112 nexttile;
113 h = heatmap(pc_ppm_5_calc);
114 h.ColorLimits = [low high];
115 h.ColorbarVisible=”off”;
116 colormap(cmap)
117 title({'Calculated error rates of' 'L1 pulse counters 5mm [ppm]'});
118 xlabel(”L1 column index”);
119 ylabel(”L1 row index”);
```

H

SKILL Scripts

H.1. Full Adder Tree generation

```
; Call example: createFatSch(list(128 64 2))
; Creates FAT with input of 128, 64, 2 for
; the weights 0-2 respectively
;
; M -> number of inputs
procedure( createFatSch(M)
  let( (cv fa i j l w sum signalTable faTable haTable outputs netId
      inputPinLocation
    outputTerm inputTerm inputBus net maxComp)
    cv = geGetEditCellView()
    inputPinLocation = 0:0
    gridStep = 0.125

    ;Clear cellview
    dbDeleteAllNet(cv)
    foreach(inst cv~>instances
      if(inst then dbDeleteObject(inst))
    )
    foreach(inst setof(def cv~>busNetDefs t)
      if(inst then dbDeleteObject(inst))
    )
    foreach(inst setof(term cv~>busNeTerms t)
      if(inst then dbDeleteObject(inst))
    )

    ;[Layer][Weight]
    ;Determine the structure of the full adder tree
    signalTable = makeTable('signalTable 0)
    faTable = makeTable('faTable 0)
    haTable = makeTable('haTable 0)

    ;Insert input into signalTable and create pins
    outputs = 0;
    for(i 0 length(M)-1
      signalTable[list(0 i)] = nth(i M)
      outputs = outputs + nth(i M)*2^i
      inputPin = schCreatePin(
        cv
        dbOpenCellViewByType( "basic" "ipin" "symbol" "" 'r )
        sprintf(nil "in%d<0:%d>" i (nth(i M)-1))
        "input"
```

```
41          nil
42          car(inputPinLocation):(cadr(inputPinLocation)-gridStep*i)
43          "R0"
44        )
45        for(j 0 nth(i M)-1
46          dbCreateNet(cv sprintf(nil "in%d<%d>" i j))
47        )
48      )
49
50      ;Create output pins
51      outputs = fix(log(outputs)/log(2))
52      schCreatePin(
53        cv
54        dbOpenCellViewByType( "basic" "opin" "symbol" "" 'r )
55        sprintf(nil "out<0:%d>" (outputs-1))
56        "output"
57        nil
58        (car(inputPinLocation)+2*gridStep):cadr(inputPinLocation)
59        "R0"
60      )
61      printf(strcat("FAT will have " sprintf(nil "%d" outputs) " outputs"))
62
63      ;FAT algorithm from report
64      declare(maxComp[outputs])
65      for(i 0 outputs-1
66        maxComp[i]=0
67      )
68      for(w 0 outputs-1 ;Loop over weigths
69        l = 0;
70        schCreateNoteLabel(cv (l*2*gridStep):((6*outputs+2)*gridStep+(-2*w*gridStep)
              ) sprintf(nil "%d" signalTable[list(l w)]) "upperLeft" "R0" "euroStyle"
              0.0625 "normalLabel");
71        sum = 0
72        while((signalTable[list(l w)] != 1 || sum > 2)
73          ;First check for HA
74          sum = 0
75          for(j 0 w
76            sum = sum + signalTable[list(l j)]
77          )
78
79          if(((signalTable[list(l w)]==2) && (sum <= 3) ) then
80            haTable[list(l w)] = 1
81          )
82
83          ;Number of FA
84          faTable[list(l w)] = fix(signalTable[list(l w)]/3)
85
86          ;Same weight outputs
87          signalTable[list(l+1 w)] = signalTable[list(l+1 w)] + signalTable[list(l w
                )]-2* faTable[list(l w)]-haTable[list(l w)]
88          ;Next weight outputs
89          signalTable[list(l+1 w+1)] =  faTable[list(l w)]+haTable[list(l w)]
90          maxComp[w] = max(maxComp[w] signalTable[list(l+1 w)])
91
92          ;Create Schematic notes
93          schCreateNoteLabel(cv ((l+1)*2*gridStep):((6*outputs+2)*gridStep+(-2*w*
                gridStep)) sprintf(nil "%d" signalTable[list(l+1 w)]) "upperLeft" "R0"
                "euroStyle" 0.0625 "normalLabel");
94          schCreateNoteLabel(cv (l*2*gridStep):((4*outputs+1)*gridStep+(-2*w*
                gridStep)) sprintf(nil "%d" faTable[list(l w)]) "upperLeft" "R0" "
                euroStyle" 0.0625 "normalLabel");
```

```
95          schCreateNoteLabel(cv (l*2*gridStep):((2*outputs)*gridStep+(-2*w*gridStep)
                ) sprintf(nil "%d" haTable[list(l w)]) "upperLeft" "R0" "euroStyle"
                0.0625 "normalLabel");
96
97          l=l+1
98          sum = 0
99          for(j 0 w
100           sum = sum + signalTable[list(l j)]
101         )
102       )
103     )
104
105     ;Create Output nets
106     let((inNets sNet cNet sigIndx lStep wStep instStep sigStep x xStart y yStart
            yCaption dx xx aliasIndx thruStep)
107     instStep = 4*gridStep
108     sigStep = 2*gridStep
109     thruStep = 2*gridStep
110     lStep = 11.5*gridStep
111     xStart = 0
112     yStart = -6*gridStep
113     yCaption = 3*gridStep
114     dx = 3.5*gridStep
115
116     declare(wStep[outputs+1])
117     wStep[0] = yStart
118     for(i 1 outputs-1
119       wStep[i] = wStep[i-1] - maxComp[i-1]*instStep-yCaption
120     )
121
122     for(w 0 outputs-1 ;Loop over weights
123       l=0
124       x = xStart
125
126
127       while((signalTable[list(l w)] != 1 || signalTable[list(l+1 w)] !=0);Loop
              over levels
128         y = wStep[w]
129         xx = x-dx
130         if(((haTable[list(l w)] > 0) || (faTable[list(l w)] > 0)) then
131           ;Crea(te Caption
132           schCreateNoteLabel(cv x:(y+gridStep) sprintf(nil "L%d_b%d" l w) "
                  lowerLeft" "R0" "euroStyle" 1.5*gridStep "normalLabel")
133           ;Create FA Component bounding box
134           schCreateNoteShape(cv "rectangle" "solid" list(xx:(y- maxComp[w]*
                  instStep) (xx+lStep):y) 0.2)
135         )
136         if(w==0 then sigIndx = 0 else sigIndx = (faTable[list(l w-1)]+haTable[list
              (l w-1)]))
137
138         if((haTable[list(l w)]>0) then
139           if(l == 0 then
140             inNets = list(
141               strcat("in" sprintf(nil "%d" w) "<0>")
142               strcat("in" sprintf(nil "%d" w) "<1>"))
143           else
144             inNets = list(
145               sprintf(nil "L%d_b%d<0>" (l-1) w)
146               sprintf(nil "L%d_b%d<1>" (l-1) w))
147           )
148
```

```
149           ;Same weight net is allways an output
150           sNet = sprintf(nil "out<%d>" w)
151           dbCreateNet(cv sNet)
152
153           ;Check if this is last next weight signal
154           sum = 0
155           for(j 0 w+1
156             sum = sum + signalTable[list(l+1 j)]
157           )
158
159           if(((signalTable[list(l+1 w+1)] == 1) && (sum <= 2)) then
160             cNet = sprintf(nil "out<%d>" w+1)
161             dbCreateNet(cv sNet)
162           else
163             ;Next weight nets always start at 0
164             cNet = sprintf(nil "L%d_b%d<0>" l (w+1))
165             dbCreateNet(cv cNet)
166           )
167
168           ;Create HA
169           createHa(cv inNets sNet cNet x y)
170           y = y-instStep
171         )
172
173         for(j 0 faTable[list(l w)]-1 ;Loop over FA's to be added in current w/l
174           if(l == 0 then
175             inNets = list(
176               sprintf(nil "in%d<%d>" w (3*j))
177               sprintf(nil "in%d<%d>" w (3*j+1))
178               sprintf(nil "in%d<%d>" w (3*j+2)))
179           else
180             inNets = list(
181               sprintf(nil "L%d_b%d<%d>" (l-1) w (3*j))
182               sprintf(nil "L%d_b%d<%d>" (l-1) w (3*j+1))
183               sprintf(nil "L%d_b%d<%d>" (l-1) w (3*j+2)))
184           )
185
186           ;Check if this is last signal
187           sum = 0
188           for(j 0 w
189             sum = sum + signalTable[list(l+1 j)]
190           )
191
192           if(((signalTable[list(l+1 w)] == 1) && (sum <= 2)) then
193             sNet = sprintf(nil "out<%d>" w)
194             dbCreateNet(cv sNet)
195           else
196             ;Same weight net starting index depends on FAs and HAs in previous
                  stage
197             sNet = sprintf(nil "L%d_b%d<%d>" l w (sigIndx+j))
198             dbCreateNet(cv sNet)
199           )
200
201           if(((signalTable[list(l+1 w+1)] == 1) && ((sum+signalTable[list(l w+1)])
                <= 2)) then
202             sNet = sprintf(nil "out<%d>" w+1)
203           else
204             ;Next weight nets always start at 0
205             cNet = sprintf(nil "L%d_b%d<%d>" l (w+1) j)
206             dbCreateNet(cv cNet)
```

```
          )

          ;Create FA
          createFa(cv inNets sNet cNet x y)
          y = y-instStep
          aliasIndx = sigIndx+j
        )

        ;Create thru wires
        let( (cnt inIndx outIndx inNet outNet)
          if(w==0 then
            outIndx = faTable[list(l w)]+
                haTable[list(l w)]
          else
            outIndx = faTable[list(l w-1)]+
                haTable[list(l w-1)] +
                faTable[list(l w)]+
                haTable[list(l w)]
          )
          inIndx = 3*faTable[list(l w)]+2*haTable[list(l w)]
          cnt = signalTable[list(l w)] - 3*faTable[list(l w)]-2*haTable[list(l w)]
          for( j 0 cnt-1
            if(l == 0 then
              inNet = sprintf(nil "in%d<%d>" w (inIndx+j))
            else
              inNet = sprintf(nil "L%d_b%d<%d>" (l-1) w (inIndx+j))
            )
            outNet = sprintf(nil "L%d_b%d<%d>" l w (outIndx+j))
            dbCreateNet(cv outNet)

            createThru(cv inNet outNet x y)
            y = y - thruStep
          )
        )

        l = l+1
        x = x+lStep
      )
    ))
  )
)
```

H.2. Critical Adder Tree generation

```
; Call example: createCratSch(list(128 64 2) 4)
; Creates CRAT with input of 128, 64, 2 for
; the weights 0-2 respectively, while limiting
; the number of output bits to 4
;
; M -> number of inputs
procedure( createCratSch(M b)
  let( (cv fa i j l w sum signalTable faTable haTable outputs netId
      inputPinLocation
    outputTerm inputTerm inputBus net maxComp or2Table or4Table
    ty)
    cv = geGetEditCellView()
    inputPinLocation = 0:0
    gridStep = 0.125

    ;Clear cellview
```

```
16      dbDeleteAllNet(cv)
17      foreach(inst cv~>instances
18        if(inst then dbDeleteObject(inst))
19      )
20      foreach(inst setof(def cv~>busNetDefs t)
21        if(inst then dbDeleteObject(inst))
22      )
23      foreach(inst setof(term cv~>busNeTerms t)
24        if(inst then dbDeleteObject(inst))
25      )
26
27      ;[Layer][Weight]
28      ;Tables to store the structure of the CRAT
29      signalTable = makeTable('signalTable 0)
30      faTable = makeTable('faTable 0)
31      haTable = makeTable('haTable 0)
32      or2Table = makeTable('or2Table 0)
33      or4Table = makeTable('or4Table 0)
34
35      ;Insert input into signalTable and create pins
36      outputs = b;
37      for(i 0 length(M)-1
38        signalTable[list(0 i)] = nth(i M)
39        inputPin = schCreatePin(
40          cv
41          dbOpenCellViewByType( "basic" "ipin" "symbol" "" 'r )
42          sprintf(nil "in%d<0:%d>" i (nth(i M)-1))
43          "input"
44          nil
45          car(inputPinLocation):(cadr(inputPinLocation)-gridStep*i)
46          "R0"
47        )
48        for(j 0 nth(i M)-1
49          dbCreateNet(cv sprintf(nil "in%d<%d>" i j))
50        )
51      )
52
53      ;Create captions for tables
54      ty = (3*(outputs+1)+6)*gridStep
55      schCreateNoteLabel(cv 0:(ty+2*gridStep) "Signal Table" "upperLeft" "R0" "
            euroStyle" 0.0625 "normalLabel");
56      schCreateNoteLabel(cv 0:(ty-gridStep*(outputs+1-1)) "FA Table" "upperLeft" "R0
            " "euroStyle" 0.0625 "normalLabel");
57      schCreateNoteLabel(cv 0:(ty-gridStep*(2*(outputs+1)-1)) "HA Table" "upperLeft"
             "R0" "euroStyle" 0.0625 "normalLabel");
58      schCreateNoteLabel(cv 0:(ty-gridStep*(3*(outputs+1)-1)) "OR4 Table" "upperLeft
            " "R0" "euroStyle" 0.0625 "normalLabel");
59      schCreateNoteLabel(cv 0:(ty-gridStep*(3*(outputs+1)+2)) "OR2 Table" "upperLeft
            " "R0" "euroStyle" 0.0625 "normalLabel");
60
61      ;Create output pins
62      schCreatePin(
63        cv
64        dbOpenCellViewByType( "basic" "opin" "symbol" "" 'r )
65        sprintf(nil "out<0:%d>" (outputs-1))
66        "output"
67        nil
68        (car(inputPinLocation)+2*gridStep):cadr(inputPinLocation)
69        "R0"
70      )
71      printf(strcat("CRAT will have " sprintf(nil "%d" outputs) " outputs"))
```

```
72
73      ;FAT algorithm from report
74      declare(maxComp[outputs])
75      for(i 0 outputs-1
76        maxComp[i]=0
77      )
78      for(w 0 (outputs-1) ;Loop over weigths, last weigth will only be OR gates
79        l = 0;
80        schCreateNoteLabel(cv (l*2*gridStep):(ty+gridStep+(-1*w*gridStep)) sprintf(
              nil "%d" signalTable[list(l w)]) "upperLeft" "R0" "euroStyle" 0.0625 "
              normalLabel");
81        sum = 0
82        while((signalTable[list(l w)] != 1 || sum > 2)
83          ;Determine the sum of signals within current column
84          sum = 0
85          for(j 0 w
86            sum = sum + signalTable[list(l j)]
87          )
88
89          ;Only add OR gates in the last row
90          gnd_fix = 0
91          if(w == (outputs-1) then
92            ;Only create OR gates
93            or4Table[list(l 0)] = fix(signalTable[list(l outputs-1)]/4)
94
95            ;Only create OR2 gate in last column
96            if(((sum <= 3) && (signalTable[list(l w)] == 2)) then
97              or2Table[list(l 0)] = 1
98            )
99            if((sum <= 4 && (signalTable[list(l w)] == 3)) then
100             ;Use an OR4 gate with one input tied to ground.
101             or4Table[list(l 0)] = 1;
102             gnd_fix = -1;Dirty fix
103           )
104           signalTable[list(l+1 outputs-1)] = signalTable[list(l+1 outputs-1)]+
                  signalTable[list(l outputs-1)]-(3*or4Table[list(l 0)]+or2Table[list(l
                   0)]+gnd_fix)
105           maxComp[w] = max(maxComp[w] signalTable[list(l+1 w)])
106
107           schCreateNoteLabel(cv ((l+1)*2*gridStep):(ty+gridStep+(-1*(outputs-1)*
                  gridStep)) sprintf(nil "%d" signalTable[list(l+1 (outputs-1))]) "
                  upperLeft" "R0" "euroStyle" 0.0625 "normalLabel");
108           schCreateNoteLabel(cv (l*2*gridStep):(ty-gridStep*(3*(outputs+1)))
                  sprintf(nil "%d" or4Table[list(l 0)]) "upperLeft" "R0" "euroStyle"
                  0.0625 "normalLabel");
109           schCreateNoteLabel(cv (l*2*gridStep):(ty-gridStep*(3*(outputs+1)+3))
                  sprintf(nil "%d" or2Table[list(l 0)]) "upperLeft" "R0" "euroStyle"
                  0.0625 "normalLabel");
110         else
111           ;First check for HA
112           if(((signalTable[list(l w)]==2) && (sum <= 3) ) then
113             haTable[list(l w)] = 1
114           )
115
116           ;Number of FA
117           faTable[list(l w)] = fix(signalTable[list(l w)]/3)
118
119           ;Same weight outputs
120           signalTable[list(l+1 w)] = signalTable[list(l+1 w)] + signalTable[list(l
                  w)]-2* faTable[list(l w)]-haTable[list(l w)]
121           ;Next weight outputs
```

```
122           signalTable[list(l+1 w+1)] =  faTable[list(l w)]+haTable[list(l w)]
123           maxComp[w] = max(maxComp[w] signalTable[list(l+1 w)])
124
125           ;Create Schematic notes
126           schCreateNoteLabel(cv ((l+1)*2*gridStep):(ty+gridStep+(-1*w*gridStep))
                  sprintf(nil "%d" signalTable[list(l+1 w)]) "upperLeft" "R0" "
                  euroStyle" 0.0625 "normalLabel");
127           schCreateNoteLabel(cv (l*2*gridStep):((ty-gridStep*(1*(outputs+1)))+(-1*
                  w*gridStep)) sprintf(nil "%d" faTable[list(l w)]) "upperLeft" "R0" "
                  euroStyle" 0.0625 "normalLabel");
128           schCreateNoteLabel(cv (l*2*gridStep):((ty-gridStep*(2*(outputs+1)))+(-1*
                  w*gridStep)) sprintf(nil "%d" haTable[list(l w)]) "upperLeft" "R0" "
                  euroStyle" 0.0625 "normalLabel");
129           )
130
131         l=l+1
132         sum = 0
133         for(j 0 w
134           sum = sum + signalTable[list(l j)]
135         )
136       )
137     )
138
139     ;Create Output nets
140     let((inNets sNet cNet sigIndx lStep wStep instStep sigStep x xStart y yStart
            yCaption dx xx sumOutIndx inIndx thruStep
141     or2Step or4Step)
142     or2Step = 3*gridStep
143     or4Step = 5*gridStep
144     instStep = 4*gridStep
145     sigStep = 2*gridStep
146     thruStep = 2*gridStep
147     lStep = 11.5*gridStep
148     xStart = 0
149     yStart = -6*gridStep
150     yCaption = 3*gridStep
151     dx = 3.5*gridStep
152
153     declare(wStep[outputs+1])
154     wStep[0] = yStart
155     for(i 1 outputs-1
156       wStep[i] = wStep[i-1] - maxComp[i-1]*instStep-yCaption
157     )
158
159     for(w 0 outputs-1 ;Loop over weights
160       l=0
161       x = xStart
162
163
164       while((signalTable[list(l w)] != 1 || signalTable[list(l+1 w)] !=0);Loop
              over levels
165         y = wStep[w]
166         xx = x-dx
167         if(((haTable[list(l w)] > 0) || (faTable[list(l w)] > 0)) then
168           ;Create Caption
169           schCreateNoteLabel(cv x:(y+gridStep) sprintf(nil "L%d_b%d" l w) "
                  lowerLeft" "R0" "euroStyle" 1.5*gridStep "normalLabel")
170           ;Create FA Component bounding box
171           schCreateNoteShape(cv "rectangle" "solid" list(xx:(y- maxComp[w]*
                  instStep) (xx+lStep):y) 0.2)
172         )
```

```
173
174         ;Determine starting index for same weigth outputs
175         if(w==0 then sigIndx = 0 else sigIndx = (faTable[list(l w-1)]+haTable[list
                (l w-1)]))
176         sumOutIndx = sigIndx
177
178         ;Check if this is the final weight
179         if(w == (outputs-1) then
180           inIndx = 0
181           if(((or2Table[list(l 0)] > 0) || (or4Table[list(l 0)] > 0)) then
182             ;Create Caption
183             schCreateNoteLabel(cv x:(y+gridStep) sprintf(nil "L%d_b%d" l w) "
                    lowerLeft" "R0" "euroStyle" 1.5*gridStep "normalLabel")
184             ;Create FA Component bounding box
185             schCreateNoteShape(cv "rectangle" "solid" list(xx:(y- maxComp[w]*
                    instStep) (xx+lStep):y) 0.2)
186           )
187
188           ;Add OR4's
189           for(j 0 or4Table[list(l 0)]-1
190             if(l == 0 then
191               inNets = list(
192                 sprintf(nil "in%d<%d>" w (4*j))
193                 sprintf(nil "in%d<%d>" w (4*j+1))
194                 sprintf(nil "in%d<%d>" w (4*j+2))
195                 sprintf(nil "in%d<%d>" w (4*j+3)))
196               else
197                 ;In case there are only 3 inputs tie 4'th to ground
198                 if((signalTable[list(l w)] == 3) then
199                   inNets = list(
200                     sprintf(nil "L%d_b%d<%d>" (l-1) w (4*j))
201                     sprintf(nil "L%d_b%d<%d>" (l-1) w (4*j+1))
202                     sprintf(nil "L%d_b%d<%d>" (l-1) w (4*j+2))
203                     "gnd!")
204                 else
205                   inNets = list(
206                     sprintf(nil "L%d_b%d<%d>" (l-1) w (4*j))
207                     sprintf(nil "L%d_b%d<%d>" (l-1) w (4*j+1))
208                     sprintf(nil "L%d_b%d<%d>" (l-1) w (4*j+2))
209                     sprintf(nil "L%d_b%d<%d>" (l-1) w (4*j+3)))
210                 )
211               )
212
213               ;Check if this is last signal
214               sum = 0
215               for(j 0 w
216                 sum = sum + signalTable[list(l+1 j)]
217               )
218               if(((signalTable[list(l+1 w)] == 1) && (sum <= 2)) then
219                 sNet = sprintf(nil "out<%d>" w)
220                 dbCreateNet(cv sNet)
221               else
222                 ;Same weight net starting index depends on FAs and HAs in level of
                      lower weight
223                 sNet = sprintf(nil "L%d_b%d<%d>" l w (sigIndx+j))
224                 dbCreateNet(cv sNet)
225               )
226
227               ;Create OR4
228               createOr4(cv inNets sNet x y)
229               y = y-or4Step
```

```
230             sumOutIndx = sumOutIndx+1
231           )
232
233           ;Add OR2's
234           for(j 0 or2Table[list(l 0)]-1
235             if(l == 0 then
236               inNets = list(
237                 sprintf(nil "in%d<%d>" w (2*j))
238                 sprintf(nil "in%d<%d>" w (2*j+1)))
239             else
240               inNets = list(
241                 sprintf(nil "L%d_b%d<%d>" (l-1) w (2*j))
242                 sprintf(nil "L%d_b%d<%d>" (l-1) w (2*j+1)))
243             )
244
245             ;Check if this is last signal
246             sum = 0
247             for(j 0 w
248               sum = sum + signalTable[list(l+1 j)]
249             )
250             if(((signalTable[list(l+1 w)] == 1) && (sum <= 2)) then
251               sNet = sprintf(nil "out<%d>" w)
252               dbCreateNet(cv sNet)
253             else
254               ;Same weight net starting index depends on FAs and HAs in previous
                      stage
255               sNet = sprintf(nil "L%d_b%d<%d>" l w (sigIndx+j))
256               dbCreateNet(cv sNet)
257             )
258
259             ;Create OR2
260             createOr2(cv inNets sNet x y)
261             y = y-or2Step
262             sumOutIndx = sumOutIndx+1
263             inIndx = inIndx + 1
264           )
265         else
266           ;Add HA's
267           if((haTable[list(l w)]>0) then
268           if(l == 0 then
269             inNets = list(
270               strcat("in" sprintf(nil "%d" w) "<0>")
271               strcat("in" sprintf(nil "%d" w) "<1>"))
272           else
273             inNets = list(
274               sprintf(nil "L%d_b%d<0>" (l-1) w)
275               sprintf(nil "L%d_b%d<1>" (l-1) w))
276           )
277
278           ;Same weight net is allways an output
279           sNet = sprintf(nil "out<%d>" w)
280           dbCreateNet(cv sNet)
281
282           ;Check if this is last next weight signal
283           sum = 0
284           for(j 0 w+1
285             sum = sum + signalTable[list(l+1 j)]
286           )
287
288           if(((signalTable[list(l+1 w+1)] == 1) && (sum <= 2)) then
289             cNet = sprintf(nil "out<%d>" w+1)
```

```
290             dbCreateNet(cv sNet)
291           else
292             ;Next weight nets always start at 0
293             cNet = sprintf(nil "L%d_b%d<0>" l (w+1))
294             dbCreateNet(cv cNet)
295           )
296
297           ;Create HA
298           createHa(cv inNets sNet cNet x y)
299           y = y-instStep
300         )
301
302           ;Loop over FA's to be added in current w/l
303           for(j 0 faTable[list(l w)]-1
304           if(l == 0 then
305             inNets = list(
306               sprintf(nil "in%d<%d>" w (3*j))
307               sprintf(nil "in%d<%d>" w (3*j+1))
308               sprintf(nil "in%d<%d>" w (3*j+2)))
309           else
310             inNets = list(
311               sprintf(nil "L%d_b%d<%d>" (l-1) w (3*j))
312               sprintf(nil "L%d_b%d<%d>" (l-1) w (3*j+1))
313               sprintf(nil "L%d_b%d<%d>" (l-1) w (3*j+2)))
314           )
315
316           ;Check if this is last signal
317           sum = 0
318           for(j 0 w
319             sum = sum + signalTable[list(l+1 j)]
320           )
321
322           if(((signalTable[list(l+1 w)] == 1) && (sum <= 2)) then
323             sNet = sprintf(nil "out<%d>" w)
324             dbCreateNet(cv sNet)
325           else
326             ;Same weight net starting index depends on FAs and HAs in previous
                  stage
327             sNet = sprintf(nil "L%d_b%d<%d>" l w (sigIndx+j))
328             dbCreateNet(cv sNet)
329           )
330
331           if(((signalTable[list(l+1 w+1)] == 1) && ((sum+signalTable[list(l w+1)])
                <= 2)) then
332             sNet = sprintf(nil "out<%d>" w+1)
333           else
334             ;Next weight nets always start at 0
335             cNet = sprintf(nil "L%d_b%d<%d>" l (w+1) j)
336             dbCreateNet(cv cNet)
337           )
338
339           ;Create FA
340           createFa(cv inNets sNet cNet x y)
341           y = y-instStep
342           aliasIndx = sigIndx+j
343         )
344         )
345
346         ;Create thru wires
347         let( (cnt inIndx outIndx inNet outNet)
348           if(w==0 then
```

```
            outIndx = faTable[list(l w)]+
                haTable[list(l w)]
            inIndx = 3*faTable[list(l w)]+2*haTable[list(l w)]
          else
            if(w==(outputs-1) then
              outIndx = faTable[list(l w-1)]+
                haTable[list(l w-1)] +
                or2Table[list(l 0)]+
                or4Table[list(l 0)]
              inIndx = 4*or4Table[list(l 0)]+2*or2Table[list(l 0)]
            else
              outIndx = faTable[list(l w-1)]+
                  haTable[list(l w-1)] +
                  faTable[list(l w)]+
                  haTable[list(l w)]
              inIndx = 3*faTable[list(l w)]+2*haTable[list(l w)]
            )
          )

          cnt = signalTable[list(l w)] - inIndx
          for( j 0 cnt-1
            if(l == 0 then
              inNet = sprintf(nil "in%d<%d>" w (inIndx+j))
            else
              inNet = sprintf(nil "L%d_b%d<%d>" (l-1) w (inIndx+j))
            )
            outNet = sprintf(nil "L%d_b%d<%d>" l w (outIndx+j))
            dbCreateNet(cv outNet)

            createThru(cv inNet outNet x y)
            y = y - thruStep
          )
        )

        l = l+1
        x = x+lStep
      )
    ))
  )
)
```

H.3. Create Half Adder

```
; Create a single Half Adder
; in -> list containing input net names
; s -> sum output net name
; c -> carry output net name
; x y -> position to create instance
procedure( createHa(cv in s c x y)
  let( (ha inst i wire xx yy wirelength dx dy)
  ha = dbOpenCellViewByType("AsicCore" "ASIC_HA" "symbol")
  inst = dbCreateInst(cv ha nil x:y "R0")
  wirelength = 3*0.125
  dx = 0.0625
  dy = 0.5*0.0625

  ;Connect input nets
  for(i 0 1
    xx = x-wirelength
    yy = y-1.5*0.125-i*0.125
```

```
    wire = schCreateWire(cv "draw" "full" list(xx:yy x:yy) 0.0625 0.0625 0)
    schCreateWireLabel(cv car(wire) ((x-dx):(yy+dy)) nth(i in) "lowerRight" "R0" "
        stick" 0.03 nil)
    dbCreateConnByName(
      dbFindNetByName(cv nth(i in))
      inst
      strcat("A<" sprintf(nil "%d" i) ">")
    )
  )

  ;Connect output nets
  x = x+4.5*0.125
  xx = x+wirelength
  yy = y-1.5*0.125
  wire = schCreateWire(cv "draw" "full" list(xx:yy x:yy) 0.0625 0.0625 0)
  schCreateWireLabel(cv car(wire) ((x+dx):(yy+dy)) s "lowerLeft" "R0" "stick" 0.03
      nil)
  dbCreateConnByName(
    dbFindNetByName(cv s)
    inst
    "S"
  )

  yy = y-2.5*0.125
  wire = schCreateWire(cv "draw" "full" list(xx:yy x:yy) 0.0625 0.0625 0)
  schCreateWireLabel(cv car(wire) ((x+dx):(yy+dy)) c "lowerLeft" "R0" "stick" 0.03
      nil)
  dbCreateConnByName(
    dbFindNetByName(cv c)
    inst
    "C"
  )
  )
)
```

H.4. Create Full Adder

```
; Create a single Full Adder
; in -> list containing input net names
; s -> sum output net name
; c -> carry output net name
; x y -> position to create instance
procedure( createFa(cv in s c x y)
  let( (fa inst i wirelength dx dy xx yy)
  fa = dbOpenCellViewByType("AsicCore" "ASIC_FA" "symbol")
  inst = dbCreateInst(cv fa nil x:y "R0")
  wirelength = 3*0.125
  dx = 0.0625
  dy = 0.5*0.0625

  ;Connect input nets
  for(i 0 2
    xx = x-wirelength
    yy = y-1*0.125-i*0.125
    wire = schCreateWire(cv "draw" "full" list(xx:yy x:yy) 0.0625 0.0625 0)
    schCreateWireLabel(cv car(wire) ((x-dx):(yy+dy)) nth(i in) "lowerRight" "R0" "
        stick" 0.03 nil)
    dbCreateConnByName(
      dbFindNetByName(cv nth(i in))
      inst
```

```
23        strcat("A<" sprintf(nil "%d" i) ">")
24      )
25    )
26
27    ;Connect output nets
28    x = x+4.5*0.125
29    xx = x+wirelength
30    yy = y-1.5*0.125
31    wire = schCreateWire(cv "draw" "full" list(xx:yy x:yy) 0.0625 0.0625 0)
32    schCreateWireLabel(cv car(wire) ((x+dx):(yy+dy)) s "lowerLeft" "R0" "stick" 0.03
          nil)
33    dbCreateConnByName(
34      dbFindNetByName(cv s)
35      inst
36      "S"
37    )
38
39    yy = y-2.5*0.125
40    wire = schCreateWire(cv "draw" "full" list(xx:yy x:yy) 0.0625 0.0625 0)
41    schCreateWireLabel(cv car(wire) ((x+dx):(yy+dy)) c "lowerLeft" "R0" "stick" 0.03
          nil)
42    dbCreateConnByName(
43      dbFindNetByName(cv c)
44      inst
45      "C"
46    )
47    )
48  )
```

H.5. Create OR-2

```
1   ; Create a single Full Adder
2   ; in -> list containing input net names
3   ; s -> sum output net name
4   ; x y -> position to create instance
5   procedure( createOr2(cv in s x y)
6     let( (or2 inst i wirelength dx dy xx yy)
7     or2 = dbOpenCellViewByType("AsicCore" "ASIC_OR2" "symbol")
8     inst = dbCreateInst(cv or2 nil x:y "R0")
9     wirelength = 3*0.125
10    dx = 0.0625
11    dy = 0.5*0.0625
12
13    ;Connect input nets
14    for(i 1 2
15      xx = x-wirelength
16      yy = y-i*0.125
17      wire = schCreateWire(cv "draw" "full" list(xx:yy x:yy) 0.0625 0.0625 0)
18      schCreateWireLabel(cv car(wire) ((x-dx):(yy+dy)) nth((i-1) in) "lowerRight" "
            R0" "stick" 0.03 nil)
19      dbCreateConnByName(
20        dbFindNetByName(cv nth((i-1) in))
21        inst
22        sprintf(nil "A<%d>" i)
23      )
24    )
25
26    ;Connect output nets
27    x = x+4.5*0.125
28    xx = x+wirelength
```

```
  yy = y-1.5*0.125
  wire = schCreateWire(cv "draw" "full" list(xx:yy x:yy) 0.0625 0.0625 0)
  schCreateWireLabel(cv car(wire) ((x+dx):(yy+dy)) s "lowerLeft" "R0" "stick" 0.03
      nil)
  dbCreateConnByName(
    dbFindNetByName(cv s)
    inst
    "S"
  )
  )
)
```

H.6. Create OR-4

```
; Create a single Full Adder
; in -> list containing input net names
; s -> sum output net name
; x y -> position to create instance
procedure( createOr4(cv in s x y)
  let( (or4 inst i wirelength dx dy xx yy)
  or4 = dbOpenCellViewByType("AsicCore" "ASIC_OR4" "symbol")
  inst = dbCreateInst(cv or4 nil x:y "R0")
  wirelength = 3*0.125
  dx = 0.0625
  dy = 0.5*0.0625

  ;Connect input nets
  for(i 1 4
    xx = x-wirelength
    yy = y-i*0.125
    wire = schCreateWire(cv "draw" "full" list(xx:yy x:yy) 0.0625 0.0625 0)
    schCreateWireLabel(cv car(wire) ((x-dx):(yy+dy)) nth((i-1) in) "lowerRight" "
        R0" "stick" 0.03 nil)
    dbCreateConnByName(
      dbFindNetByName(cv nth((i-1) in))
      inst
      sprintf(nil "A<%d>" i)
    )
  )

  ;Connect output nets
  x = x+4.5*0.125
  xx = x+wirelength
  yy = y-2.5*0.125
  wire = schCreateWire(cv "draw" "full" list(xx:yy x:yy) 0.0625 0.0625 0)
  schCreateWireLabel(cv car(wire) ((x+dx):(yy+dy)) s "lowerLeft" "R0" "stick" 0.03
      nil)
  dbCreateConnByName(
    dbFindNetByName(cv s)
    inst
    "S"
  )
  )
)
```

H.7. Create Thru-Alias

```
; Create a thru connection (net alias)
; in -> input net name
```

```
3  ; out -> sum output net name
4  ; x y -> position to create instance
5  procedure( createThru(cv in out x y)
6    let( (thru inst i wire xx yy wirelength dx dy)
7    thru = dbOpenCellViewByType("AsicCore" "ASIC_thru" "symbol")
8    inst = dbCreateInst(cv thru nil x:y "R0")
9    wirelength = 3*0.125
10   dx = 0.0625
11   dy = 0.5*0.0625
12
13   ;Connect input net
14   xx = x-wirelength
15   yy = y-0.125
16   wire = schCreateWire(cv "draw" "full" list(xx:yy x:yy) 0.0625 0.0625 0)
17   schCreateWireLabel(cv car(wire) ((x-dx):(yy+dy)) in "lowerRight" "R0" "stick"
         0.03 nil)
18   dbCreateConnByName(
19     dbFindNetByName(cv in)
20     inst
21     "A")
22
23   ;Connect output nets
24   x = x+4.5*0.125
25   xx = x+wirelength
26   yy = y-0.125
27   wire = schCreateWire(cv "draw" "full" list(xx:yy x:yy) 0.0625 0.0625 0)
28   schCreateWireLabel(cv car(wire) ((x+dx):(yy+dy)) out "lowerLeft" "R0" "stick"
         0.03 nil)
29   dbCreateConnByName(
30     dbFindNetByName(cv out)
31     inst
32     "B")
33   )
34 )
```

Verilog AMS components

I.1. Poissonian cluster

```
// VerilogA for AsicCore, poisson, veriloga

`include "constants.vams"
`include "disciplines.vams"
`timescale 1ns / 1ps

module poisson_dual256(clk, out, syncOut, rst);
        parameter real lambda = 2.0; // Rate parameter (events per 2.5 ns)
        parameter real vdd = 1.1;
        parameter real T = 2.5n;                        // Clock period
        parameter real tevent = 2.49n;  // Duration a signal is asserted
        parameter integer QUEUE_SIZE = 24;
        parameter integer SEED = 10;
        parameter real risetime = 5p;
        parameter real falltime = 5p;
        integer seed1 = SEED;
        integer seed2 = SEED-5;
        integer scount;
        real r = lambda / T;
        real next_event_time, prev_event_time, dt, last_edge, next_edge;
        real u;
        real tmp[0:255];
        real currentOut[0:255];
        genvar n;
        genvar i;

        input clk, rst;
        electrical clk, rst;
        output [0:255] out, syncOut;
        electrical [0:255] out, syncOut;

        real queue_deassert_time[0:QUEUE_SIZE];
        integer queue_j[0:QUEUE_SIZE];
        integer queue_tail = 0;
        integer enabled = 0;

        analog initial begin
                u = abs({$random(seed1)}) / (1.0 * (1 << 31));
                prev_event_time = $realtime;
                next_event_time = prev_event_time + (-ln(1-u)/r);
                queue_deassert_time[0] = -0.1n;
```

```
42          end
43
44          analog begin
45                  if(V(rst)<(vdd/2)) begin
46                          for (i = 0; i < 256; i = i + 1) begin
47                                  tmp[i] = 0;
48                          end
49                  end
50
51                  //@(above(deadtime-rel_to_clk)) begin
52                  //      deadtime_correction = deadtime;
53                  //end
54                  //@(above(rel_to_clk - (T-deadtime))) begin
55                  //      deadtime_correction = -1*deadtime;
56                  //end
57
58                  @(timer(next_event_time, 0)) begin
59                          // Roll a random output where the event occurs
60                          queue_j[queue_tail] = $rdist_uniform(seed2, 0, 255);
61                          if(V(rst)>(vdd/2)) begin
62                                  for (i = 0; i < 256; i = i + 1) begin
63                                          tmp[queue_j[queue_tail]] = vdd;
64                                  end
65                          end
66                          queue_deassert_time[queue_tail] = next_event_time + tevent;
67                          queue_tail = queue_tail+1;
68
69                          // Pick a random number between 0-1 (uniformly distributed)
70                  u = $rdist_uniform(seed1, 0, 1);
71                          // Use inverse poisson transform to find time interval so that P(X=0)=
72                          // e^(-rt)=u
73                          dt = (-ln(u)/r);
74                          prev_event_time = next_event_time;
75                          next_event_time = next_event_time + dt;
76                  end
77
78                  @(timer(queue_deassert_time[0], 0)) begin
79                          tmp[queue_j[0]] = 0;
80
81                          // Shift queue
82                          queue_deassert_time[0]=0;
83                          queue_j[0] = 0;
84                          for (n=0; n<QUEUE_SIZE-1; n = n + 1) begin
85                                  queue_deassert_time[n]=queue_deassert_time[n+1];
86                                  queue_j[n] = queue_j[n+1];
87                          end
88                          queue_deassert_time[queue_tail]= 0;
89                          queue_j[queue_tail] = 0;
90                          queue_tail = queue_tail - 1;
91                  end
92
93                  @(cross(V(clk)-(vdd/2), 0)) begin
94                          scount = 0;
95                          for (i = 0; i < 256; i = i + 1) begin
96                                  currentOut[i] = tmp[i];
97                                  scount = tmp[i]>0.1 ? scount+1:scount;
98                          end
99                  end
100
101
102                 for (i = 0; i < 256; i = i + 1) begin
```

```
                        V(out[i]) <+ transition(tmp[i], 0, risetime, falltime, 1p);
                        V(syncOut[i]) <+ transition(currentOut[i], 0, risetime, falltime, 1p);
                end

        end
endmodule
```

I.2. Signal counter

```
// VerilogA for SemBackendCore, counter, veriloga

`include "constants.vams"
`include "disciplines.vams"

module counter256_E(clk, in, E, out);
        input [0:255] in;
        input E;
        input clk;
        output [0:7] out;
        electrical [0:255] in;
        electrical [0:7] out;
        electrical clk, E;
        integer cnt_b, cnt, cnt_out, r, q;
        parameter integer size_in = 256;
        parameter integer width = 8;
        parameter real threshold = 0.55,
        digHigh = 1.1,
        digLow = 0.0,
        trise = 15p,
        tfall = 15p;
        genvar i, p;

        analog initial begin
                cnt = 0;
                cnt_b = 0;
                cnt_out = 0;
        end

        analog begin
                for(i = 0; i<size_in; i = i+1) begin
                        @(cross(V(in[i])-threshold, 1, 100f))
                                if (V(E)>0.55) begin
                                        cnt = cnt+1;    // Array containing input booleans
                                end
                end

                @(cross(V(clk)-threshold, 0, 10f)) begin
                        cnt_out = cnt_b;
                        cnt_b = cnt;
                        cnt = 0; // Reset counter
                end

                q = cnt_out;
                for(p = 0; p < width; p=p+1) begin
                        r = q % 2;
                        q = floor(q / 2);
                        V(out[p]) <+ transition(r==0 ? digLow:digHigh, 0, trise, tfall);
                end
        end
endmodule
```

I.3. Error Tracker

```
// VerilogA for AsicCore, errortracker, veriloga

`include "constants.vams"
`include "disciplines.vams"

module errortracker(ref, res, clk, error);
        input [0:7] ref, res;
        electrical [0:7] ref, res;
        output error;
        electrical error;
        integer iref, ires, ierror, total_error, total_cnt, r, q, tmp, s;
        input clk;
        electrical clk;
        real rel_error, prob_error, fails, pass;

        parameter threshold = 0.55;
        parameter vdd = 1.1;

        analog initial begin
                ierror = 0;
                total_cnt = 0;
                s = 0;
                prob_error = 0;
                pass = 0;
                fails = 0;
        end

        genvar i;
        analog begin
                @(cross(V(clk)-threshold, 0, 1p)) begin
                        // Determine integer values of inputs
                        ires = 0;
                        iref = 0;
                        for (i = 0; i<8; i=i+1) begin
                                tmp = ((V(ref[i]) > threshold)) ? (2**i) : 0;
                                iref = iref + tmp;
                                tmp = ((V(res[i]) > threshold)) ? (2**i) : 0;
                                ires = ires + tmp;
                        end
                        ierror = abs(iref - ires);

                        s = s+1;
                        total_error = total_error + ierror;
                        total_cnt = total_cnt + iref;
                        r = ierror;

                        // Determine error probability
                        if(ierror == 0) begin
                                pass = pass + 1;
                        end
                        else begin
                                fails = fails + 1;
                        end
                        prob_error = (fails / s);


                        // Determine accuracy (error relative to true events)
                        if(iref == 0) begin
                                rel_error = ierror==0 ? 0:ires;
```

```
                        end
                        else if(ires == 0) begin
                                rel_error = iref;
                        end
                        else begin
                                rel_error = ((ires-iref)/iref);
                        end
                end

                V(error) <+ transition(r==0 ? 0:vdd, 0, 10p, 10p);
        end


endmodule
```

I.4. Digital to Integer

```
// VerilogA for AsicCore, errortracker, veriloga

`include "constants.vams"
`include "disciplines.vams"

module dig2int(in);
        input [0:4] in;
        parameter integer width = 5;
        electrical [0:4] in;
        integer int, tmp;

        parameter threshold = 0.55;
        genvar i;

        analog begin
                int = 0;
                for (i = 0; i<width; i=i+1) begin
                        tmp = ((V(in[i]) > threshold)) ? (2**i) : 0;
                        int = int + tmp;
                end
        end


endmodule
```

I.5. Pipeline Segment model

```
// VerilogA for AsicCore, readout_model, veriloga

`include "constants.vams"
`include "disciplines.vams"

module readout_model(clk, out);
        parameter real lambda = 2.0;
        parameter real vdd = 1.1;
        parameter real digHigh = 1.1,
        digLow = 0.0,
        trise = 15p,
        tfall = 15p;
        parameter integer seed = 1977, width =6;

        input clk;
        electrical clk;
        output [0:5] out;
```

```
18          electrical [0:5] out;
19
20          integer cnt_out, r, q;
21          genvar p;
22
23          analog initial begin
24                  cnt_out = 0;
25          end
26
27          analog begin
28                  @(cross(V(clk)-(vdd/2), 0, 10f)) begin
29                          cnt_out = $rdist_poisson(seed, lambda);
30                  end
31
32                  q = cnt_out;
33                  for(p = 0; p < width; p=p+1) begin
34                          r = q % 2;
35                          q = floor(q / 2);
36                          V(out[p]) <+ transition(r==0 ? digLow:digHigh, 0, trise, tfall);
37                  end
38          end
39
40  endmodule
```

Bibliography

[1] J. I. Goldstein, D. E. Newbury, J. R. Michael, N. W. Ritchie, J. H. J. Scott, and D. C. Joy, *Scanning electron microscopy and X-ray microanalysis*. springer, 2017.

[2] T. Koshikawa and R. Shimizu, "A monte carlo calculation of low-energy secondary electron emission from metals," *Journal of Physics D: Applied Physics*, vol. 7, no. 9, p. 1303, Jun. 1974. DOI: 10.1088/0022-3727/7/9/318. [Online]. Available: https://dx.doi.org/10.1088/0022-3727/7/9/318.

[3] H. Niedrig, "Physical background of electron backscattering," *Scanning*, vol. 1, no. 1, pp. 17–34, 1978. DOI: https://doi.org/10.1002/sca.4950010103. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/sca.4950010103. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/sca.4950010103.

[4] Y. o. Wang, Z. Dong, R.-L. Lai, and K. Kanai, *Semiconductor charged particle detector for microscopy*, US Patent 0123152A1, Apr. 2023.

[5] D. Yan, T. Wu, Y. Liu, and Y. Gao, "An efficient sparse-dense matrix multiplication on a multicore system," in *2017 IEEE 17th International Conference on Communication Technology (ICCT)*, 2017, pp. 1880–1883. DOI: 10.1109/ICCT.2017.8359956.

[6] E. Abbe, "Beiträge zur theorie des mikroskops und der mikroskopischen wahrnehmung," *Archiv für Mikroskopische Anatomie*, vol. 9, pp. 413–468, 1873. DOI: https://doi.org/10.1007/BF02956173.

[7] T. F. S. Inc., *Sem resolution*. [Online]. Available: https://www.thermofisher.com/nl/en/home/materials-science/learning-center/applications/sem-resolution.html (visited on 03/14/2025).

[8] T. SOARES, O. Jesus, E. Souza, M. Rossi, and E. OLIVEIRA, "Comparative pollen morphological analysis in the subgenera passiflora and decaloba," *Anais da Academia Brasileira de Ciências*, vol. 90, Oct. 2017. DOI: 10.1590/0001-3765201720170248.

[9] M. A. Disi, A. M. Zaki, Q. Fan, and S. Nihtianov, "High-count rate, low power and low noise single electron readout asic in 65nm cmos technology," in *2021 XXX International Scientific Conference Electronics (ET)*, 2021, pp. 1–5. DOI: 10.1109/ET52713.2021.9580005.

[10] A. M. Zaki and S. Nihtianov, "Experimental qualification of a low-noise charge-sensitive roic with very high time resolution," in *2023 IEEE 32nd International Symposium on Industrial Electronics (ISIE)*, 2023, pp. 1–6. DOI: 10.1109/ISIE51358.2023.10228077.

[11] K. P. Arat K.T. Hagen C.W., "Simulation of electron-matter interaction in electron beam lithography and metrology," Ph.D. dissertation, TU Delft, 2021. DOI: https://doi.org/10.4233/uuid:52c5c8c7-edc7-4cf8-9945-bcd94e30f9d8.

[12] A. S. M. Pojar, "Advanced resists for e-beam lithography: Processing, exposure and characterization (part iii)," in *Tech. Rep.*, 2015.

[13] A. M. Zaki and S. Nihtianov, "Challenges of high-resolution electron detection asics for sem microscopy," in *2024 9th International Conference on Mathematics and Computers in Sciences and Industry (MCSI)*, 2024, pp. 68–76. DOI: 10.1109/MCSI63438.2024.00019.

[14] M. Hazewinkel, Ed., *Encyclopaedia of Mathematics*. Springer Netherlands, 1989.

[15] M. El-Desouki, M. Jamal Deen, Q. Fang, L. Liu, F. Tse, and D. Armstrong, "Cmos image sensors for high speed applications," *Sensors*, vol. 9, no. 1, pp. 430–444, 2009, ISSN: 1424-8220. DOI: 10.3390/s90100430. [Online]. Available: https://www.mdpi.com/1424-8220/9/1/430.

[16] Y. Tochigi, K. Hanzawa, Y. Kato, *et al.*, “A global-shutter cmos image sensor with readout speed of 1-tpixel/s burst and 780-mpixel/s continuous,” *IEEE Journal of Solid-State Circuits*, vol. 48, no. 1, pp. 329–338, 2013. DOI: 10.1109/JSSC.2012.2219685.

[17] T. Geurts, B. Cremers, M. Innocent, *et al.*, “A 98 db linear dynamic range, high speed cmos image sensor,” *Proceedings of the International Image Sensor Worshop, Hiroshima, Japan*, vol. 30, p. 43, 2017.

[18] J. van Rantwijk, M. Grim, D. van Loon, S. Yates, A. Baryshev, and J. Baselmans, “Multiplexed readout for 1000-pixel arrays of microwave kinetic inductance detectors,” *IEEE Transactions on Microwave Theory and Techniques*, vol. 64, no. 6, pp. 1876–1883, 2016. DOI: 10.1109/TMTT.2016.2544303.

[19] M. Sarkar and A. Theuwissen, *A biologically inspired CMOS image sensor*. Springer, 2012.

[20] S. Kleinfelder, Y. Chen, K. Kwiatkowski, and A. Shah, “High-speed cmos image sensor circuits with in situ frame storage,” *IEEE Transactions on Nuclear Science*, vol. 51, no. 4, pp. 1648–1656, 2004.

[21] T. Sugiyama, S. Yoshimura, R. Suzuki, and H. Sumi, “A 1/4-inch qvga color imaging and 3-d sensing cmos sensor with analog frame memory,” in *2002 IEEE International Solid-State Circuits Conference. Digest of Technical Papers (Cat. No. 02CH37315)*, IEEE, vol. 1, 2002, pp. 434–479.

[22] R. Kleczek, P. Grybos, R. Szczygiel, and P. Maj, “Single photon-counting pixel readout chip operating up to 1.2 gcps/mm2 for digital x-ray imaging systems,” *IEEE Journal of Solid-State Circuits*, vol. 53, no. 9, pp. 2651–2662, 2018. DOI: 10.1109/JSSC.2018.2851234.

[23] F. Erdinger, “Design of front end electronics and a full scale 4k pixel readout asic for the dssc x-ray detector at the european xfel,” Ph.D. dissertation, Ruperto-Carola University of Heidelberg, 2016.

[24] O. C. Wells, *The Construction of a Scanning Electron Microscope and Its Application to the Study of Fibers*. 1957.

[25] A. Sakic, G. van Veen, K. Kooijman, *et al.*, “High-efficiency silicon photodiode detector for sub-kev electron microscopy,” *IEEE Transactions on Electron Devices*, vol. 59, no. 10, pp. 2707–2714, 2012. DOI: 10.1109/TED.2012.2207960.

[26] S. Kimoto and H. Hashimoto, “Stereoscopic observation in scanning microscopy using multiple detectors,” *PAPER FROM THE ELECTRON MICROPROBE, 1966, P 480- 489*, 1966.

[27] F. Villa, R. Lussana, D. Bronzi, *et al.*, “Cmos imager with 1024 spads and tdcs for single-photon timing and 3-d time-of-flight,” *IEEE Journal of Selected Topics in Quantum Electronics*, vol. 20, no. 6, pp. 364–373, 2014. DOI: 10.1109/JSTQE.2014.2342197.

[28] C. Zhang, N. Zhang, Z. Ma, *et al.*, “A 240 × 160 3d-stacked spad dtof image sensor with rolling shutter and in-pixel histogram for mobile devices,” *IEEE Open Journal of the Solid-State Circuits Society*, vol. 2, pp. 3–11, 2022. DOI: 10.1109/OJSSCS.2021.3118332.

[29] H. Seo, G. Cho, J. Kim, *et al.*, “A cmos lidar sensor with pre-post weighted-histogramming for sunlight immunity over 105 klx and spad-based infinite interference canceling,” in *2021 Symposium on VLSI Circuits*, 2021, pp. 1–2. DOI: 10.23919/VLSICircuits52068.2021.9492328.

[30] S. M. Patanwala, I. Gyongy, H. Mai, *et al.*, “A high-throughput photon processing technique for range extension of spad-based lidar receivers,” *IEEE Open Journal of the Solid-State Circuits Society*, vol. 2, pp. 12–25, 2022. DOI: 10.1109/OJSSCS.2021.3118987.

[31] A. Wall, P. Walsh, K. Sadeghipour, I. O’Connell, and D. O’Hare, “An improved linearity ring oscillator-based current-to-digital converter,” *IEEE Solid-State Circuits Letters*, vol. 5, pp. 202–205, 2022. DOI: 10.1109/LSSC.2022.3198367.

[32] U. Cini and A. Morgül, “A current-mode multi-valued adder circuit for multi-operand addition,” *International Journal of Electronics*, vol. 98, no. 6, pp. 735–751, 2011. DOI: 10.1080/00207217.2011.567039. eprint: https://doi.org/10.1080/00207217.2011.567039. [Online]. Available: https://doi.org/10.1080/00207217.2011.567039.

[33] G. Tretter, M. M. Khafaji, D. Fritsche, C. Carta, and F. Ellinger, “Design and characterization of a 3-bit 24-gs/s flash adc in 28-nm low-power digital cmos,” *IEEE Transactions on Microwave Theory and Techniques*, vol. 64, no. 4, pp. 1143–1152, 2016. DOI: 10.1109/TMTT.2016.2529599.

[34] H. T. Bui, Y. Wang, and Y. Jiang, “Design and analysis of low-power 10-transistor full adders using novel xor-xnor gates,” *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 49, no. 1, pp. 25–30, 2002.

[35] B. Gilchrist, J. H. Pomerene, and S. Wong, “Fast carry logic for digital computers,” *IRE Transactions on Electronic Computers*, no. 4, pp. 133–136, 2009.

[36] B. Parhami, *Computer arithmetic*. Oxford university press Oxford, 2010, pp. 155–176.

[37] E. E. Swartzlander, “Parallel counters,” *IEEE Transactions on computers*, vol. 100, no. 11, pp. 1021–1024, 2006.

[38] C. S. Wallace, “A suggestion for a fast multiplier,” *IEEE Transactions on Electronic Computers*, vol. EC-13, no. 1, pp. 14–17, 1964. DOI: 10.1109/PGEC.1964.263830.

[39] B. Parhami, *Computer arithmetic*. 2000.

[40] Y. Lavania, G. T. Varghese, and K. K. Mahapatra, “An ultra low power encoder for 5 bit flash adc,” in *2013 International Conference on Emerging Trends in VLSI, Embedded System, Nano Electronics and Telecommunication System (ICEVENT)*, 2013, pp. 1–5. DOI: 10.1109/ICEVENT.2013.6496578.

[41] Y. H. Wang, “On the number of successes in independent trials,” *Statistica Sinica*, vol. 3, no. 2, pp. 295–312, 1993, ISSN: 10170405, 19968507. [Online]. Available: http://www.jstor.org/stable/24304959 (visited on 05/03/2025).

[42] W. Biscarri, S. D. Zhao, and R. J. Brunner, “A simple and fast method for computing the poisson binomial distribution function,” *Computational Statistics & Data Analysis*, vol. 122, pp. 92–100, 2018, ISSN: 0167-9473. DOI: https://doi.org/10.1016/j.csda.2018.01.007. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167947318300082.

[43] A. D. Barbour and P. Hall, “On the rate of poisson convergence,” *Mathematical Proceedings of the Cambridge Philosophical Society*, vol. 95, no. 3, pp. 473–480, 1984. DOI: 10.1017/S0305004100061806.

[44] J. M. Rabaey and A. Chandrakasan, *B. Nikoli c, Digital Integrated Circuits*. Prentice Hall, 2003.

[45] P. Drennan and C. McAndrew, “Understanding mosfet mismatch for analog design,” *IEEE Journal of Solid-State Circuits*, vol. 38, no. 3, pp. 450–456, 2003. DOI: 10.1109/JSSC.2002.808305.

[46] M. Ajanya and G. T. Varghese, “Thermometer code to binary code converter for flash adc - a review,” in *2018 International Conference on Control, Power, Communication and Computing Technologies (ICCPCCT)*, 2018, pp. 502–505. DOI: 10.1109/ICCPCCT.2018.8574244.

[47] R. Sireesha and A. Kumar, “Design of low power 0.8v flash adc using tiq in 90nm technology,” in *2015 International Conference on Smart Technologies and Management for Computing, Communication, Controls, Energy and Materials (ICSTM)*, 2015, pp. 406–410. DOI: 10.1109/ICSTM.2015.7225451.

[48] S. P. Khatri, R. K. Brayton, A. L. Sangiovanni-Vincentelli, S. P. Khatri, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, “Vlsi layout fabrics,” *Cross-Talk Noise Immune VLSI Design Using Regular Layout Fabrics*, 2001.

[49] M. Afghahi and J. Yuan, “Double-edge-triggered d-flip-flops for high-speed cmos circuits,” *IEEE Journal of Solid-State Circuits*, vol. 26, no. 8, pp. 1168–1170, 1991. DOI: 10.1109/4.90071.

[50] P. J. Goodhew and J. Humphreys, *Electron microscopy and analysis*. CRC press, 2000.

[51] L. Reimer, “Emission of backscattered and secondary electrons,” in *Scanning Electron Microscopy: Physics of Image Formation and Microanalysis*. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 135–169, ISBN: 978-3-540-38967-5. DOI: 10.1007/978-3-540-38967-5_4. [Online]. Available: https://doi.org/10.1007/978-3-540-38967-5_4.

[52] K. Heinrich, “Electron probe microanalysis by specimen current measurement,” *X-ray Optics and Microanalysis*, pp. 159–167, 1966.

[53] W. Reuter, “Electron backscattering as a function of atomic number,” *Proceeding 6th International Cong x-ray optics and microanalysis. University of Tokyo Press, Tokyo*, vol. 121, 1972.

[54] J. M. Kevin McNamara, *Modeling secondary electron trajectories in scanning electron microscopes*, 2016. [Online]. Available: https://scholarsarchive.library.albany.edu/honorscollege_nano/17.

[55] L. Reimer and C. Tollkamp, “Measuring the backscattering coefficient and secondary electron yield inside a scanning electron microscope,” *Scanning*, vol. 3, no. 1, pp. 35–39, 1980. DOI: https://doi.org/10.1002/sca.4950030105. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/sca.4950030105. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/sca.4950030105.

[56] F. Akbari, “A comprehensive open-access database of electron backscattering coefficients for energies ranging from 0.1 kev to 15 mev,” *Medical Physics*, vol. 50, no. 9, pp. 5920–5929, 2023. DOI: https://doi.org/10.1002/mp.16604. eprint: https://aapm.onlinelibrary.wiley.com/doi/pdf/10.1002/mp.16604. [Online]. Available: https://aapm.onlinelibrary.wiley.com/doi/abs/10.1002/mp.16604.

[57] V. Adler and E. Friedman, “Uniform repeater insertion in rc trees,” *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 47, no. 10, pp. 1515–1523, 2000. DOI: 10.1109/81.886981.

[58] J. Cong and D. Z. Pan, “Interconnect estimation and planning for deep submicron designs,” in *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, 1999, pp. 507–510.

[59] W. J. Dally and J. W. Poulton, *Digital systems engineering*. Cambridge university press, 1998.

[60] S. Tam, “Modern clock distribution systems,” in *Clocking in Modern VLSI Systems*, T. Xanthopoulos, Ed. Boston, MA: Springer US, 2009, pp. 9–65, ISBN: 978-1-4419-0261-0. DOI: 10.1007/978-1-4419-0261-0_2. [Online]. Available: https://doi.org/10.1007/978-1-4419-0261-0_2.

[61] N. Bindal, T. Kelly, N. Velastegui, and K. Wong, “Scalable sub-10ps skew global clock distribution for a 90nm multi-ghz ia microprocessor,” *2003 IEEE International Solid-State Circuits Conference, 2003. Digest of Technical Papers. ISSCC.*, 346–498 vol.1, 2003. [Online]. Available: https://api.semanticscholar.org/CorpusID:21807231.