The Pennsylvania State University

The Graduate School

College of Engineering

ANALYSIS OF TWO-DIMENSIONAL MEDIAN FILTER HARDWARE IMPLEMENTATIONS FOR REAL-TIME VIDEO DENOISING

A Thesis in

Computer Science and Engineering

by

Jesse Scott

Submitted in Partial Fulfillment

of the Requirements

for the Degree of

Master of Science

December 2010

The thesis of Jesse Scott was reviewed and approved* by the following:

Vijaykrishnan Narayanan
Professor of Computer Science and Engineering
Thesis Adviser

Mary Jane Irwin
Robert E. Noll Professor of Computer Science and Engineering

Mahmut Kandemir
Professor of Computer Science and Engineering
Director of Graduate Affairs, Department of Computer Science and Engineering

*Signatures are on file in the Graduate School.

Abstract

The two-dimensional spatial median filter is a core algorithm for impulse noise removal in digital image processing and computer vision. While the literature presents several analyses of median filters optimized for a standard 3x3 pixel neighborhood configuration, a 5x5 neighborhood, useful for imagery exhibiting noise not conforming to the classic “salt and pepper” formation, has received little analysis. Research efforts on hardware implementations of median filters have been devoted primarily toward implementations with low latency and high throughput. A system is in development that utilizes intensified visible to near-infrared sensors and requires a 5x5 median filter to handle intensifier noise. Since the system is a battery powered unit, optimal power usage is a critical requirement in addition to low latency and high throughput. However, optimal power usage for median filtering has received little attention in the literature.

This research focuses on investigating five selected hardware implementations of a 5x5 median filter and comparing them on the basis of power dissipation. The latency, maximum clock rates, and resource utilization for some of these implementations are also analyzed. The designs considered include implementations of a radix sort-based elimination algorithm, a systolic sorting array, a Batcher sorting network, and two insertion sorting networks. The two insertion sort networks have nearly identical sorting cores, but one utilizes a fundamentally different wrapper; this will be referred to as a row insertion sort network. Also included in the analysis is a commercial off the shelf (COTS) intellectual property (IP) core. The five custom filters were designed to be fully pipelined by accepting inputs and generating median output values every pixel clock pulse with a constant latency. All five custom designs are integrated with a wrapper function that provides buffering to handle the kernel based behavior of two-dimensional filtering. Initially, a merge sort implementation was also considered, but was almost immediately eliminated because the power and resource usage was an order of magnitude greater than all other designs under review.

Table of Contents

List of Figures

List of Tables

Acknowledgments

Thank you to Mike Pusateri and Electronic and Computer Services for all of the publication, computational, and research support.

I would also like to extend my sincere appreciation to Heatfusion Inc. for supporting me financially during my educational experience.

Thanks are extended to Umar Mushtaq for being my HDL expert throughout this experience. Without him, I would still be debugging the designs or trying to run the simulations correctly.

Thank you to my father for teaching me what education means, why commitment is important, and how to make a decision. You are always missed.

Finally, I would like to thank my wife Laura for her patience through this process, support during my education, and self restraint with my poor punctuation. I love you with all my heart.

1 INTRODUCTION

Chapter 1 provides an introduction to the previous investigation that directed the need for a 5x5 median filter to mitigate the intensifier noise sourced from the visible to near-infrared imager. The destination system and design restrictions are also presented in Chapter 1. Chapter 2 covers background research on sorting techniques including hardware and software approaches of the common categories of techniques. Chapter 3 will detail the structure of the simulation test bed and explain the variable space under investigation in the simulations. Also covered in this chapter is the thought process used for selection of the designs for testing. Chapter 4 will provide details about the hardware designs selected for analysis. This includes the sorting modules as well as the wrappers used to convert the sorters to two-dimensional image filters. Chapter 5 provides the results and analysis of all five custom designs as well as the Altera IP core. A collection of secondary analyses are also presented including an investigation into the variation of pixel clock while holding all other variables constant, the differences between real and synthetic imagery for simulation, and the effects of varying aspect ratio while maintaining the total pixel count constant. Additionally, a small selection of resolution variants, implemented in the Xilinx ISE environment, is analyzed to show a relatively similar progression as compared to the Altera simulations. Chapter 6 contains conclusions as well as a summary of this research and results. This chapter also presents the top performers over the variable space under analysis as well as post research observations.

1.1 Methodology

The usefulness of a design is dictated by the flexibility of its implementation. The primary obligation for designing any image processing algorithm is to consider the engineering design process. The proper process is to initially consider all viable options and then reduce the possibility set until only a few truly practical options remain. The remaining options should then be analyzed to determine the ideal method as defined by the engineering constraints. This procedure holds true for both algorithm selection and implementation. In order to complete it, one must investigate all options, but focus on those which are fruitful for the end goal. From this explanation, it should be clear that sufficient research will be completed focusing on the narrow

scope defined, leaving related, but non-pertinent questions unanswered while always considering the final global goals.

1.2 Problem origins

The electro-optic sensor industry is migrating to digital imagers from older analog technologies. The intent is to achieve a wide range of spectral band sensitivities using low mass, power efficient, and cost effective focal plane arrays (FPAs). With migration to digital imagers, some analog technologies and techniques have been carried over to increase the functionality of select digital imagers. The technique we are focusing on in this paper is the intensification of visible to near-infrared imagers. The intensification process, although very beneficial to improving sensitivity and effective illumination range, induces scintillation noise into the sampled imagery. The occurrence rate of this noise is directly correlated to the amount of intensification that is utilized.

1.2.1 Electro-optic phenomenon

The electro-optic phenomenon that generates scintillation noise is the result of the technology used to sample photons at the FPA level. Details about the exact technology and the electro-optic phenomenon of the Electronic Bombardment Active Pixel Sensor (EBAPS) are not available because of security, proprietary, and trade restrictions that limit the dissemination of information pertaining to this sensor's technology. However, there is an excellent explanation of the EBAPS technology provided in the patent for the technology in [1]. The resulting visual effects that occur with this sensing technique are very similar to a more widely understood intensifier technology for analog night vision systems based on vacuum tube technology. During this discussion, the visual effects will be related to the analog intensifier behaviors.

1.2.2 Electro-optic technology

The following is information attained from Intevac on the ISIE line of EBAPS sensors. "The core imaging sensor is an electron bombarded Complementary Metal–Oxide–Semiconductor (CMOS) sensor, which is based on use of a Gallium Arsenide (GaAs) photocathode derived from Generation-III image intensifier technology in proximity focus with a high resolution, backside-thinned, CMOS imager anode. The electrons emitted by the photocathode are directly injected in the electron bombarded mode into the CMOS anode. [2]"

1.2.3 Visual phenomenon

The visual phenomenon that occurs as a result of the intensification process creates two major categories of noise. The more distracting of these is what is referred to as scintillation throughout the remainder of this document. There is also a secondary and less distracting noise that is uniformly present in the background of the imagery. This multiplicative background noise significantly affects the Signal to Noise Ratio (SNR) of imagery as compared to the sparse scintillation noise. However, the visual impact of scintillation noise is far more influential than multiplicative noise to a human observer. During the algorithm research process, scintillation noise removal was difficult to quantize numerically. This was the primary reason for including subjective visual analysis in addition to the numerical metric.

1.2.4 Scintillation noise distribution

The characteristic behavior of scintillation noise is the result of the technology behind the intensification process. As a result, the noise is multi-pixel extent where the peak does not quite cause complete saturation of the region it is affecting. The scintillation visually appears to have a Cauchy spatial distribution that is temporally independent frame to frame. Cauchy is a long tail distribution that closely resembles the distribution of peaks that we refer to as scintillation. Scintillation is scene content independent causing both background and foreground content to be corrupted equally, although scintillation is hard to detect and has attenuated affects in high intensity areas of the scene content. The distribution of a single scintillation event is poorly defined, but an exemplar can be seen in Figure 1-1.

1.3 The core issue

The scintillation noise is distracting to the user and negatively impacts algorithms dependent on accurate statistical information. Figure 1-1 provides a scintillation noise example from an intensified imager that is typical of the noise occurring throughout the imagery. This noise was sampled from a uniform intensity region within a raw, high resolution image. When observing the scintillation noise, note that the peak is multi-pixel in extent and not fully saturated. The image used in Figure 1-1 is a single raw frame acquired from an Intevac EBAPS ISIE10. This imager was specifically sourced for raw imagery because it uses the same technology as the sensor in the final night vision goggle that triggered this research. By observing the other areas of the imagery, one should notice there are additional inconsistencies in the imagery that are not scintillation. These variations primarily consist of the following: multiplicative background noise, bad pixels (FPA imperfections), non-uniformity, and possible optical effects. These variations are visually distracting, but not the focus of this research. Any and all affects associated with the secondary variations are not considered in the analysis of the designs. It should be noted that the final design may indirectly affect these secondary inconsistencies in a positive or negative fashion. These effects are not excluded from the visual analysis because of the inherent difficulties in separating secondary changes from the intended changes.

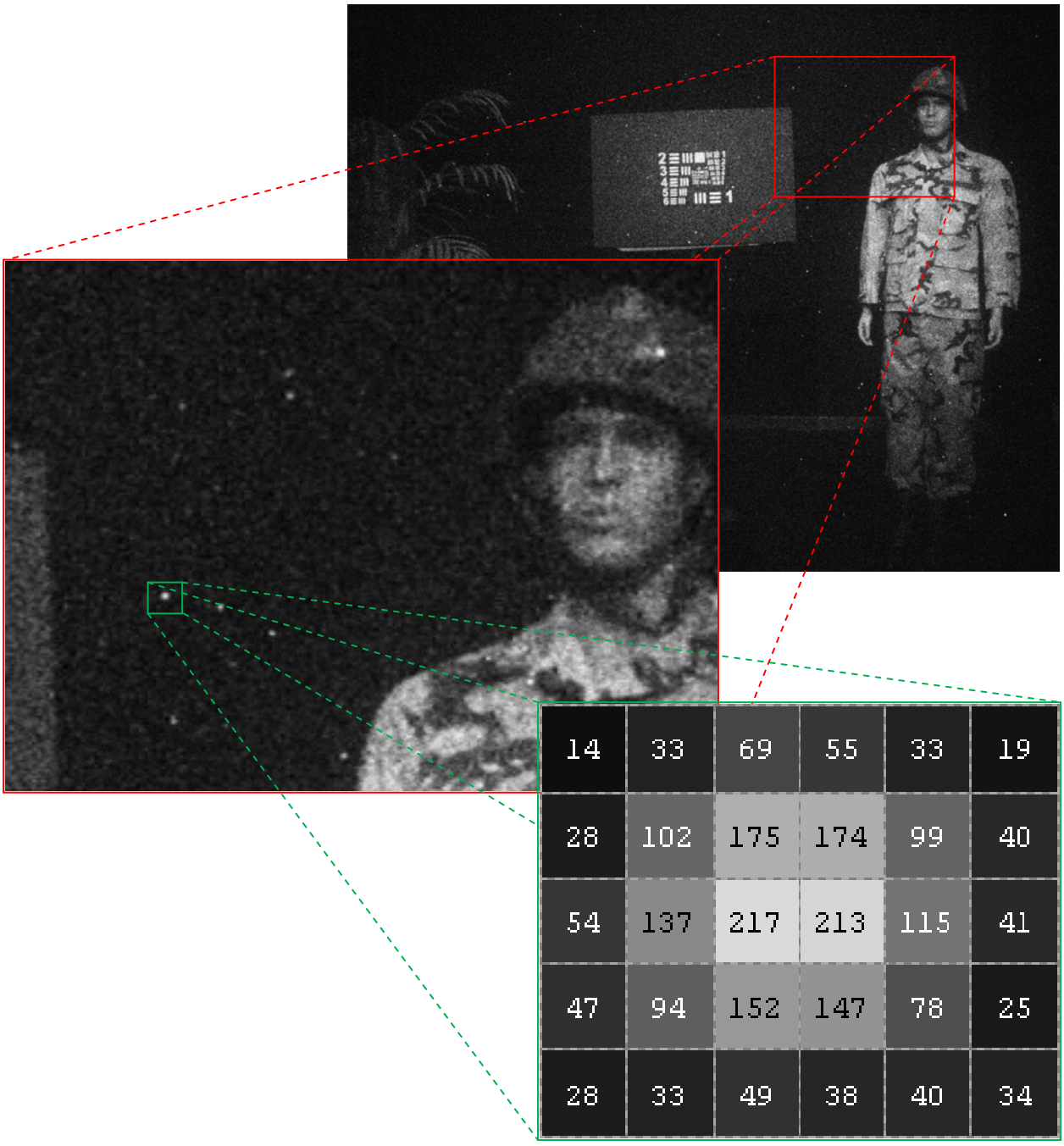

Figure 1-1 – Scintillation noise from an intensified digital imager

1.4 Engineering constraints

The constraints that are in place for the design of a scintillation removal solution are as follows:

- must use minimal power while meeting all other engineering requirements
- must significantly attenuate or completely remove scintillation noise while maximizing image quality
- must process the FPA serial pixel stream while maintaining intra-frame system latency
- must propagate all timing and control signals with appropriate latency and signal characteristics
- must be amendable to pixel synchronization with other sensor channels
- must be completed for Altera FPGA destination hardware
- must be able to process a sensor at 140.4 MHz (1800x1300 resolution at 60Hz)

1.5 Design considerations

Based on the power requirements of the system, the scintillation noise removal solution must be formulated with power minimization in mind. Goggle systems that utilize battery packs are small and light weight, limiting energy storage. Although the destination night vision system uses two matched imagers, the imagery will not have matched scintillation noise. Therefore, each instance of the solution must be able to handle random scintillation on two independent imagers while maintaining deterministic runtime and predictable results. The intra-frame latency requirement eliminates the ability to perform full frame buffering. Pixel synchronization and control signal propagation requirements force the design toward constant latency designs. The nature of the imagery being processed dictates the design must be able to handle a serial pixel stream that is at a minimum of 140.4 MHz but should be designed to handle larger sensor arrays and higher frame rates. Considering technology will only advance in resolution and sample rate, the design should not be at the limits of the technology. Moore's Law will not "save" any design; the sensor technology progresses at the same rate as FPGAs because they are both silicon based technologies.

1.6 Destination platform

The platform for implementation is a field programmable gate array (FPGA) based mobile electronic board contained within the night vision goggle system.

The FPGA supports:

- two near infrared (NIR) sensors (1800 x 1300)
- one long wave infrared (LWIR) sensor (1152 x 832)
- two displays (1440x1040)
- three DDR3 memories
- miscellaneous controls and indicators (buttons, illuminators, non-imaging sensors)

The following operations are performed on each sensor channel:

- sensor acquisition
- non-uniformity correction (NUC)
- bad pixel mapping (replacement)
- noise removal/attenuation
- spatial resample
- electronic zoom
- automatic gain control (AGC)
- spatial intensity contrast enhancement
- synchronization
- multi-spectral fusion
- gamma correction

The goggle is intended as a wearable binocular device that places the stereo near-infrared imagers directly over the user's visual path. Because the goggle replaces the view of the operator, there must be as little latency as possible; the requirement is to achieve intra-frame latency from acquisition to display. To perform all these operations for each of the three imaging channels requires significant resources within the FPGA restricting the resources available for scintillation noise removal.

1.7 Algorithm investigation

Prior to efforts to deploy a solution to the FPGA, significant algorithm development was performed to determine the best approach to attenuate the scintillation noise. During this process, MATLAB was used to process a base collection of raw sensor data similar to Figure 1-1. The data set was utilized as a source by a wide range of image processing algorithms. A total of 47 variants were tested for SNR as an analytical metric and the resulting videos were subjectively graded for visual performance. The results shown in Table 1-1 provide the algorithm description, kernel size, and SNR for selected variants of the 47 tested methods. From Table 1-1, it can be seen that a 5x5 median filter is the top performer analytically followed by the 9x9 Wiener filter and the 5x5 disk filter. The visual performance of median on the scintillation noise was judged superior to other techniques. The top three algorithms are highlighted in Table 1-1.

Size	Average	Gaussian	Median	Wiener	Disk
3x3 (Radius 1)	58.878	43.023	57.211	45.385	51.542
5x5 (Radius 2)	66.477	43.084	72.603	57.330	68.093
7x7 (Radius 3)	54.271	43.084	60.454	64.924	66.850
9x9 (Radius 4)	43.057	43.084	46.837	68.840	55.941

Table 1-1 – Summary of SNR metric results for scintillation removal

1.7.1 Median filter

Median filtering is a specific variant of statistical order filters. The basic process of median filter requires the sorting of the data set and then the selection of the middle index of the set for sets with an odd number of entries [3]. An example of this is to sort the following nine element data set {4, 7, 1, 3, 8, 9, 2, 7, 2} in ascending order then select the fifth index. This process results in a sorted set {1, 2, 2, 3, 4, 7, 7, 8, 9} where the fifth index holds the value 4. Median

filtering is a non-linear operation that has excellent performance when statistical outliers are a significant component of the input data set. This characteristic makes median filter an ideal candidate for impulse and impulse-like noise removal. The outlier count must account for less than 50% of the input data set to maintain robustness against outliers. The non-linear nature of this filter results in very little high frequency content loss while removing almost all scintillation and significantly attenuating the remaining instances.

1.7.2 Wiener filter

The Wiener filter is a statistical filter that is often used for deconvolution. The high performance in analytical analysis is the result of the removal of non-scintillation noise. The Wiener filter, published by Norbert Wiener in 1949, is based on statistics [4]. The filter attempts to temporally estimate the noise component of the image to allow for this component to be attenuated in the original image. This technique, while excellent for additive and multiplicative noise, is not well suited for scintillation removal. Its visual results were appealing, but failed to remove scintillation.

1.7.3 Disk filter

The disk filter is a simple averaging filer that only considers a disk shaped region rather than a square region. The method is linear in nature and has simple structure and implementation. The visual results are plagued with a significant reduction in high frequency content. This is because the disk filter is a variant of simple low pass filters. This technique, while performing well in terms of SNR improvement, has poor visual results because of the inherent smoothing.

2 BACKGROUND ON SORTING TECHNIQUES

During the literature review for methods to implement a sorting hardware, two primary categories for finding the median became apparent. Approximation methods apply mathematical, algorithmic, and heuristic methods to approximate or estimate the median of a given set of data. Exact methods apply different algorithmic approaches to completely or partially sort a given set of data. Both methods use sorting techniques, while the organization of the data set manipulation is structurally different.

2.1 Approximate approaches

During the background algorithm research, a decision was made to set aside all methods that did not provide exact results. Initial research into the two main categories shows visual promise, but the approximate approaches category must be considered separate from the analysis of median filtering in this paper.

While reviewing approximate methods, the literature covered a unique sub-category of approximate methods: separable median [5][6][7]. This category is representative of the most common approximate approach that appeared during the literature reviewed. The merits of approximate approaches keeps them in contention for scintillation noise removal, but must be considered as a completely different technique that is only partially related to median filtering.

2.1.1 Separable

The separable median filter is the fundamental technique in [6], [7], and [8]. The seminal paper for this technique is [5]. In this paper, Tukey refers to the separable median as the Ninther for the 3x3 kernel. More generally the term Nther, where N stands for the number of pixels within the two dimensional kernel, can be used to describe filters of any size. The separable median technique is a sequence of one-dimensional median filters that sort the columns/rows

then sorts the rows/columns. The technique produces excellent approximations when the outlier count is low, but this is not as efficient as a full median 2D filter and is shown in [5].

Separate research is currently being completed on the separable technique presented in [5] to expand the mathematical proof of the Ninther (3x3) to an Nther. The research should provided a better understanding of whether the separable technique should be considered as an acceptable method for removing scintillation. The separable method is not going to be considered in this analysis. If separable median is selected as a valid removal technique, the multiple small sorts should dramatically reduce total resource and power usage independent of the actual sorting technique selected for each of the small sorters.

2.2 Exact approaches

In reviewing exact approaches to implement a sort for finding the median, we found five core categories: iterative[9], radix [10][11][12][13], histogram [14][15][16][17], systolic system [18][19][20][21], and sorting network [9][22][23][24][25][26][29] methods. These categories are representative of the most common sorting techniques reviewed. From these categories, four approaches were chosen and implemented in five unique designs. The following is an overall description of the five core categories including specific sorting techniques that fall into the categories. Additionally, two individual techniques [27] and [8] that produce exact median values are present in the following sections. While producing exact results, their implementations have aspects that eliminate them from further consideration because of engineering constraints.

2.2.1 Iterative

The standard text on sorting is [9] presenting these methods in detail from Section 5.2.1 to 5.2.4. Many of the commonly known sorting methods are what can be referred to as iterative methods. The reason iterative methods are well known is because they are the basis of almost all software based sorting approaches. These methods are also well known because they are the first

steps used in teaching computer science at any education level [9]. There are many iterative methods including: insertion, selection, exchange, partition, and merge methods. All share similar characteristics including: read after write (RAW) and write after read (WAR) conflicts that force serial iterative execution as well as being pair-wise comparison based techniques. These methods, although very useful and practical in the world of serial CPU designs, are neither useful nor practical in the world of highly parallel FPGA and ASIC designs. It should be noted that one of the initial problems with iterative methods is the non-deterministic behavior of both individual iterations and total iterations.

Some methods, like merge and quick sorting, can be manipulated into coarsely parallel designs while retaining the primary iterative processes. For example, the merge sort technique was parallelized exactly how the algorithm naturally splits, but resulted in very poor performance. The poor performance was an order of magnitude more power dissipation and resource consumption with the initial sorter implementation. As a result, the iterative techniques were dropped from further implementation, testing, and analysis. The results of the initial performance evaluation can be seen in [28].

2.2.2 Radix

Radix sorting is fundamentally different to all other techniques reviewed. Radix sort uses bit-wise binning to determine the ordering of values. An excellent explanation of this technique is provided in [9]. This is a very uncommon process currently because of the non-bitwise nature of CPUs but in the time of punch cards this technique was used much more often. In custom hardware for FPGAs, the bitwise burden is lifted and this method produces a simple bitwise approach. Radix based methods are the only methods reviewed that do not perform direct pair-wise comparison; this results in a structure different from the other implementations. The authors of [10], [11], [12], and [13] all utilized different forms of radix sorting to perform fast median filtering for image and signal processing. The radix method can't be reduced specifically when the median is the only value needed as a result rather than creating a fully sorted set. The uniqueness of this method and its robust use of bitwise processing caused this method to be considered for further analysis.

2.2.3 Histogram

A core technique for calculating the median in image and signal processing is a technique originally presented by Huang in [14] called histogram updating or running median. Histogram based techniques in [15], [16], and [17] are all based in the original technique described in [14]. The fundamental process for histogram based median filtering is to take the histogram of the previous filter region, remove the column of pixels from that set that do not overlap with the next set, and add the column of pixels that are new from the next set. If the region is m by n then n pixels are removed and n pixels are added. The premise being that the larger the filter region, the more computational savings are gained from this approach. The median is then tracked by removing old pixels or inserting new pixels into the histogram as it moves through the image. This is also the only technique that is centered on image filtering rather than set sorting.

One of the advantages this technique has over other methods is the data reuse leveraged from the inherent overlap. This combined with the running median technique reduces the sorting memory and number of operations that are required to generate each median value in an image. This is also the pitfall when deciding to utilize this method in an FPGA environment since the reductions are offset with increased external memory bandwidth and size requirements. Since this system can't afford to utilize external memory, this approach has very little merit. The technique is very resource intensive and computationally complex in exchange for shorter execution time per median.

While [15] calculates true medians, it only does this sparsely and uses the exact values to interpolate the medians between exact points. The method achieves its speed up and resource reduction by only calculating a single median in a tiled window fashion rather than one median for each pixel in a sliding window fashion. This tiling technique is not acceptable since it lacks true medians at all pixel locations. The speedup method in [16] exploits memory and storage to manipulate regional histograms. This method makes overlapping histograms resulting in a duplication of pixels, N squared times for a NxN region. As a result, [16] utilizes huge amounts of resources that are just not available in the destination system. The method in [17] does not

transfer well to the FPGA realm because the authors attained their speed up primarily through fully utilizing built-in CPU architecture enhancements.

2.2.4 Systolic systems

The literature review presented a category of sorting that will be referred to as systolic systems. The materials in [18], [19], [20], and [21] explain the basics of systolic systems based in CPU structures. The material in [18] focuses on the complexity of many sorting technique, but it does have a excellent section analyzing systolic systems as well as sorting networks. The research in [19] covers a more focused discussion on systolic architectures including how to utilize them for sorting. The materials in [20] and [21] are specifically systolic systems used for median filtering. It should be noted that systolic systems could be categorized as sorting networks, but should not be. They originate from single instruction multiple data (SIMD) and multiple instructions multiple data (MIMD) CPU systems as opposed to sorting networks that originate from FPGA and ASIC logic design. This unique ancestry creates a separate category of methods and is substantiated by the fact that a systolic system is not fully configurable. Systolic systems are only partially controllable and are intended to be software tools, not custom hardware modules. The systolic methodology shows merits like structurally simple organization, minimal routing of wires, and modularized design.

2.2.5 Sorting network

Sorting networks are a very specific category of sorters because they are unique to the number of inputs and are often customized for the specific application and/or target device. Knuth provides an excellent description of sorting networks in [9]. He describes the fundamentals of all sorting networks as well as theoretical analysis of resources and latencies. The basic premise of sorting networks is the use of multiple 2x2 sorters, the fundamental components of sorting networks, to process the inputs using pair-wise comparisons in a larger pipelined design. The intent is to arrange the inputs into a progressively better order at each stage until the set is guaranteed to be completely sorted.

The two primary sub categories of sorting networks are based in the format of the input data to the sorter. Methods like the Batcher sorter [22] and the perfect shuffle [23] begin sorting all pixels simultaneously while methods like Weavesort [26] and insertion/selection sort [9] sort simultaneously, but stage each input pixel to sort one at a time.

The Batcher sorting technique is clearly described in [22] and has been used in networking applications like the Starlite switch [24] to perform very high speed sorting. An in-depth mathematical analysis is performed in [9] showing that there is a theoretical lower bound on the number of 2x2 sorters that are required for complete sorting. Knuth also shows in [9] that a Batcher sort is near minimum and the smallest known method excluding special cases that have been discovered for specific input sizes. The near minimum resource usage makes the Batcher sorter an ideal candidate for this designs needs.

The Weave sort [29] is essentially a parallel form of software based insertion sort and is used exactly for median calculation in VLSI [26] and for general sorting in data compression [30]. Knuth describes, in [9], how to recursively build an n input sorter from an n-1 sorter. The design is a fully parallel insertion/selection sort. The one pixel at a time methods actually present two techniques: Weavesort starts with all pixels and buffers as needed to maintain timing while insertion sort does not need all pixels at once. These sorting networks provide multiple unique approaches that include unique characteristics that make them worthy of further investigation.

2.2.6 Three-dimensional

The three-dimensional filter presented in [27] attempts to mitigate impulse noise by including the spatially correlated pixels that are temporally one sample before and after the current image. The results in [27] are clearly better for the three-dimensional filter, but come at a significant drawback in requiring two frames of buffering and one frame of latency. In addition to violating the latency constraint previously discussed, the frame buffering would impact the power budget as well by requiring a high-bandwidth external memory. While the approach in [27] does provide excellent visual results, both of the inherent drawbacks make this design unacceptable based on the engineering constraints outlined previously.

2.2.7 Row-column

The approach in [7] is a method using iterative row-column-diagonal sorting similar to the separable technique introduced in [5], but with exact median results - not approximations. This is a viable method that does provide an exact median for 3x3 regions. However, this technique does not appear to scale from the 3x3 method presented in the literature to a 5x5 technique needed for this research. The limitations in [7] currently cannot be overcome without additional research and analysis and therefore are not considered.

The approach in [8] is a method using iterative row-column sorting referred to as shear-sort. This is a viable method that does appear to be scalable to a 5x5 technique needed for this research, but is not amendable to hardware implementation. The discussion in [8] indicates a specific sequence of calculating the median, which is accomplished by sorting the rows and columns repeatedly until the order is stable. While [8] did prove that this sequence will reach steady state, the number of iterations was not shown to be deterministic; the lack of deterministic latency violates the engineering constraints.

3 SIMULATION ARCHITECTURE

To accomplish an analysis of different median filter implementations, there are two possible approaches. The designs can be tested in a physical destination system or they can be simulated utilizing sample data sets. In this research, simulation is used because the final destination system is not well equipped or available for time consuming measurements. The goggle electronic system is also not able to handle the variety of input formats that need to be evaluated.

This research utilizes the ModelSim simulation tool to produce picoseconds resolution simulation outputs. These simulation outputs will then be processed using the Altera and Xilinx HDL tools for power consumption and parsed to verify imagery from simulation results. The HDL tools will also determine the resource usage and provide maximum clock frequencies.

3.1 Test bench

The test bench design for this research is based on real imagery. While this research has academic implications, it also has industrial applications. Using test patterns or synthetic images (graphics) will not model the inputs of real world applications. The currently available sensor technology is limited to a 1280x1024 resolution with 10 bit deep pixels and 30 frames per second frame rate. The next evolutionary sensor will be a 1600x1200 resolution with 12 bit deep pixels and 60 frames per second frame rate.

Obviously, imagery can't be acquired from a sensor that is still under development. So, to keep the simulations consistent across resolutions and pixel deeps, imagery was used from a different sensor and synthetic intensification scintillation was added. Testing was also done with true scintillation filled images for evaluation of initial categories of analysis. There is also a comparison of simulations with real scintillation and synthetic scintillation to show the differences resulting from testing with synthetic content.

3.1.1 Inputs, outputs, and format

Each median filter design accepts the same input format and provides the same output format. The test bench structure is the same for all simulations. The input is a serial stream of pixels accompanied by a frame synchronization signal, a row synchronization signal, and a pixel clock. The output is a serial pixel stream accompanied by frame and row synchronization signals that all have the same latency.

3.1.2 Behavior

The test bench simulates two full frames of imagery by iterating through an array twice and generating the frame, row, and clock signals. The test bench holds an array that is the resolution and pixel depth of the input image that it is based on. For all resolutions, pixel depths, and frame rates, the test bench is configured to utilize a subset of the array content. The test bench also has the capability to tile the array to simulate resolutions that are larger in width and/or height than the originating image.

3.1.3 Data content

There are two different test bench types that are used for all simulations and results. Both are configurable for resolution, pixel depth, and frame rate.

The “real” test bench type is based on an array of 1280x1024, 8 bit pixels that is a single image captured from the real sensor, a NightVista ISIE10 EBAPS sensor manufactured by Intevac. The scene is a static sequence captured in a control setting with no motion. There are three different real test benches used during simulation that are from three different frames of real imagery. The use of multiple frames was intended to mitigate simulation errors and image variation errors that exists frame to frame. An example of one of the images used is shown in Figure 3-1.

Figure 3-1 – Example of image used in "real" test bench types

The "synthetic" test bench is an array of 1600x1200, 16 bit pixels that is generated from a set of images captured from a commercially available, visible camera with synthetic scintillation added. The imagery was captured with a Nikon D90 DSLR camera in raw mode resulting in a 4288x2848, 12 bit pixel image. The set of 16 identical pictures were taken in a controlled fashion and then summed to generate an image that has 16 bit pixel depth, but still retained all of the sensor characteristics of the 12 bit depth images. The 16 bit pixel image is mixed with synthetically generated scintillation to generate the final "synthetic" image. A single test bench was used based on the real test bench results and the time it takes to simulate two full frames. An example of one of the images used is shown in Figure 3-2. A detailed sub-window of the

synthetic image is shown in Figure 3-3 to highlight the look of the scintillation noise at the same scale as the real imagery.

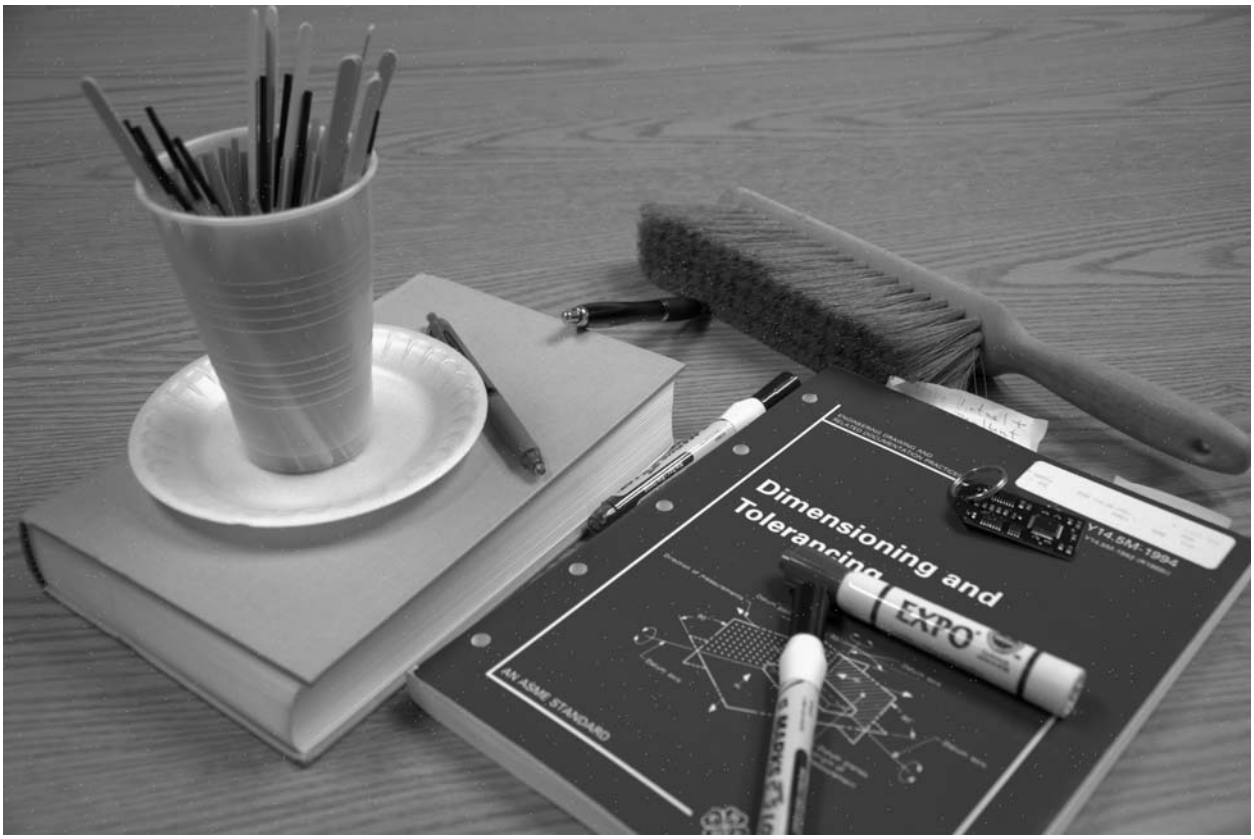

Figure 3-2 – Example of full resolution image used in "synthetic" test bench types

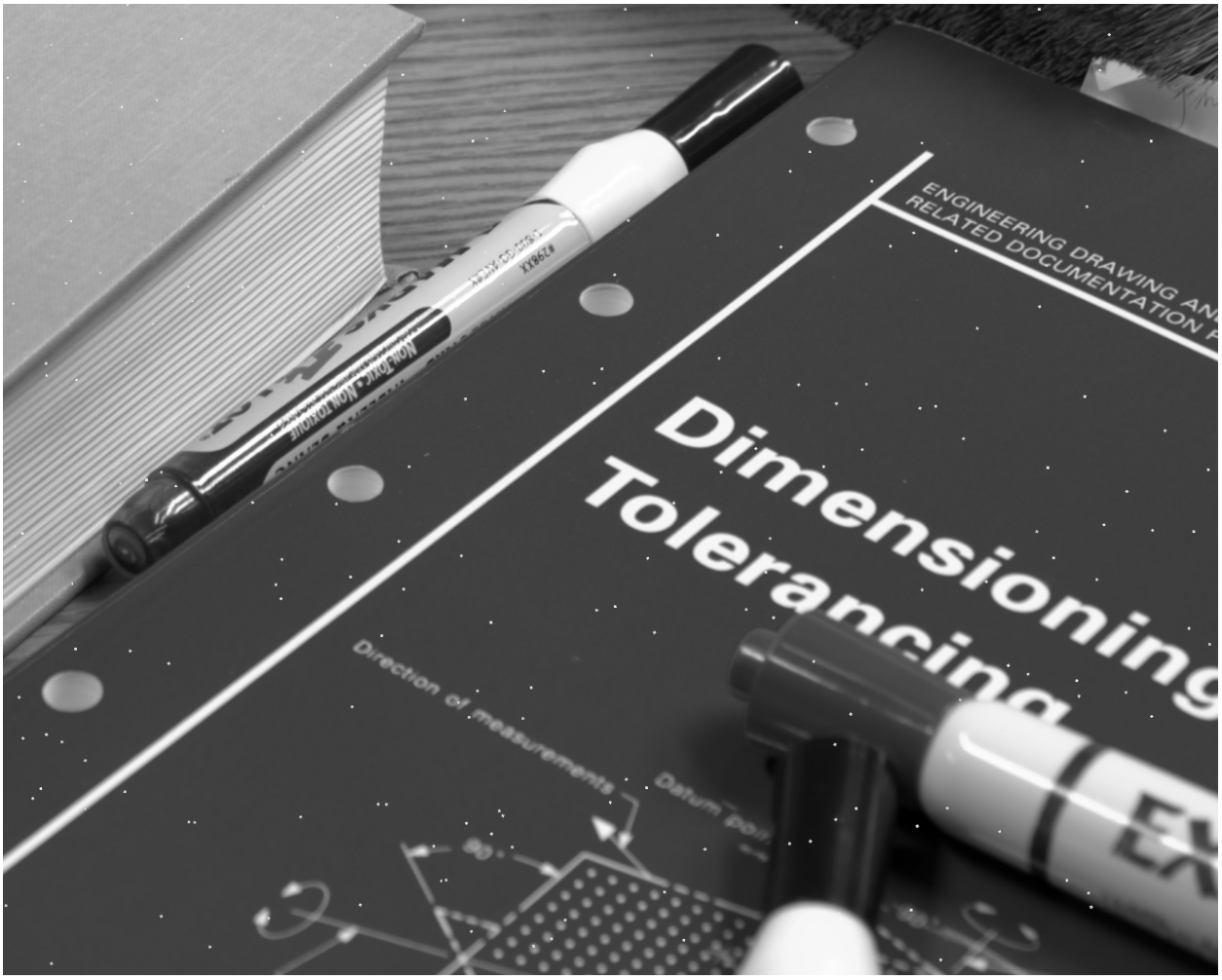

Figure 3-3 – Detailed sub-window of image used in "synthetic" test bench types

3.2 Simulation Space

There are multiple aspects to consider when performing simulations to evaluate HDL designs. The following is a list of these aspects:

- How does the image content affect the designs?
- How does the frame rate affect the designs?
- How does the aspect ratio affect the designs?
- How does the pixel depth affect the designs?
- How does the resolution affect the designs?

- Does the destination system change the relative performance of the designs?

To evaluate the above aspects, the simulation sets are divided into groups to aid in the analysis. The categories are image content, frame rate, aspect ratio, pixel depth, resolution, and destination system.

3.2.1 Image content

The image content analysis is intended to evaluate relative performance between simulations utilizing imagery with real scintillation noise and synthetic scintillation noise while holding all variables constant. Table 3-1 shows the two variants of simulations used for this analysis. The real simulation test bench set is used with all three test benches for the real category in Table 3-1 while the synthetic set is used with three test benches for the synthetic category in Table 3-1. A total of six simulations are performed for the lowest power design. To hold "all variable constant", the clock frequency is held constant, the SVGA standard is used with an 8 bit pixel depth, and 60 frames per second is used for the frame rate.

Scintillation Type	Width	Height	Clock (MHz)	FPS (Hz)	Mb/s	Depth
Real	800	600	28.8	60	219.7265625	8
Synthetic	800	600	28.8	60	219.7265625	8

Table 3-1 – Scope of variable space for real versus synthetic simulations

3.2.2 Frame rate

The frame rate analysis is intended to evaluate the effects of varying clock frequency while holding all other variables constant. The clock rate was manipulated to generate frame rates of 30, 60, 120, 240, and 540 frames per second. The resolution is 800x600 and the pixel depth is 8 bits. These frame rate simulations were performed for the lowest power design using the one of the three real test benches. The configurations are shown in Table 3-2.

Name	Width	Height	Clock (MHz)	FPS	Mb/s	Depth
SVGA	800	600	14.4	30	109.86	8
SVGA	800	600	28.8	60	219.73	8
SVGA	800	600	57.6	120	439.45	8
SVGA	800	600	115.2	240	878.91	8
SVGA	800	600	259.2	540	1977.54	8

Table 3-2 – Scope of variable space for frame rate simulations

3.2.3 Aspect ratio

The aspect ratio analysis is intended to evaluate the effects of the width of a frame while all other variables are held constant. The aspect ratio of 1:1, 5:4, 4:3, 3:2, 16:10, and 16:9 are all standard while the remaining entries in Table 3-3 were added to enhance the effects of wide and narrow aspect ratios. The sample image is tiled because it is not always large enough for the test bench region of simulations. The test bench set used for these simulations is the real set and was run for each variant for each of the three test benches in the set. A total of 39 aspect ratio simulations are performed for the lowest power design. Both bit depth and frame rate are held constant, while the variation in aspect ratio dictated some latitude in pixel clock rates.

Ratio	Width	Height	Clock (MHz)	FPS (Hz)	Mb/s	Depth
1:12	200	2400	28.80000	60	219.73	8
1:3	400	1200	28.80000	60	219.73	8
1:1	693	693	28.81494	60	219.84	8
5:4	775	620	28.83000	60	219.96	8
4:3	800	600	28.80000	60	219.73	8
3:2	849	566	28.83204	60	219.97	8
16:10 (8:5)	880	550	29.04000	60	221.56	8
16:9	912	513	28.07136	60	214.17	8
16:3	1600	300	28.80000	60	219.73	8
64:3	3200	150	28.80000	60	219.73	8
256:3	6400	75	28.80000	60	219.73	8
768:1	19200	25	28.80000	60	219.73	8
19200:1	96000	5	28.80000	60	219.73	8

Table 3-3 – Scope of variable space for aspect ratio simulations

3.2.4 Pixel depth

The pixel depth analysis is intended to evaluate the effects of the depth of a pixel while all other variables are held constant. The pixel depths of 8, 12, and 16 are all standard depths and are entries in Table 3-4. The test bench set used for these simulations is the synthetic set and was run for each variant for one of the test benches in the set. A total of 3 pixel depth simulations are performed for the Batcher design while both resolution and frame rate are held constant.

Name	Width	Height	Clock (MHz)	FPS	Mb/s	Depth
SVGA	800	600	28.8	60	219.73	8
SVGA	800	600	28.8	60	329.59	12
SVGA	800	600	28.8	60	439.45	16

Table 3-4 – Scope of variable space for aspect ratio simulations

3.2.5 Resolution

The resolution analysis is intended to evaluate relative performance between designs over a range of input formats. Based on the information presented in Table 3-5, each design under simulation is tested for the following variable ranges. The resolutions 320x240, 640x480, 800x600, and 1600x1200 are tested for 30 and 60 frames per second with 8, 12, and 16 bit pixel depths. This means that for each design, 24 variants are simulated.

The sample points selected are limited by the maximum resolution that was found possible for simulation. Larger simulations may be possible with more sophisticated computing resources. Additional test benches were attempted. but they failed to execute because of lack of available memory, as will be explained in Section 5.4.1.

Name	Width	Height	Clock (MHz)	FPS	Mb/s	Depth
QVGA	320	240	2.3	30	17.58	8
QVGA	320	240	4.6	60	35.16	8
VGA	640	480	9.2	30	70.31	8
SVGA	800	600	14.4	30	109.86	8
VGA	640	480	18.4	60	140.63	8
SVGA	800	600	28.8	60	219.73	8
UXGA	1600	1200	57.6	30	439.45	8
UXGA	1600	1200	115.2	60	878.91	8
QVGA	320	240	2.3	30	26.37	12
QVGA	320	240	4.6	60	52.73	12
VGA	640	480	9.2	30	105.47	12
SVGA	800	600	14.4	30	164.79	12
VGA	640	480	18.4	60	210.94	12
SVGA	800	600	28.8	60	329.59	12
UXGA	1600	1200	57.6	30	659.18	12
UXGA	1600	1200	115.2	60	1,318.36	12
QVGA	320	240	2.3	30	35.16	16
QVGA	320	240	4.6	60	70.31	16
VGA	640	480	9.2	30	140.63	16
SVGA	800	600	14.4	30	219.73	16
VGA	640	480	18.4	60	281.25	16
SVGA	800	600	28.8	60	439.45	16
UXGA	1600	1200	57.6	30	878.91	16
UXGA	1600	1200	115.2	60	1,757.81	16

Table 3-5 – Scope of variable space for resolution simulations

3.2.6 Destination system

The destination system analysis is intended to evaluate relative performance between simulations in different development environments while holding all variables constant. Table 3-6 shows the two environments, Altera and Xilinx, used for simulation in this analysis. The synthetic simulation test bench set is used with one of the test benches for all variants in Table 3-6. For each development environment, the 320x240, 640x480, 800x600, and 1600x1200 resolutions are used with 8 bit pixel depth and 30 frames per second frame rate. A total of 8

simulations are performed for the Batcher design. To accommodate differences in proprietary components, the sorting modules are wrapped with row buffers that are structurally the same, but utilize the destination systems unique block RAM interfaces and behaviors.

Software	Format	Width	Height	Depth	Clock (MHz)	FPS (Hz)
Xilinx	QVGA	320	240	8	2.30400	30.00
Xilinx	VGA	640	480	8	9.21600	30.00
Xilinx	SVGA	800	600	8	14.40000	30.00
Xilinx	UXGA	1600	1200	8	57.60000	30.00
Altera	QVGA	320	240	8	2.30400	30.00
Altera	VGA	640	480	8	9.21600	30.00
Altera	SVGA	800	600	8	14.40000	30.00
Altera	UXGA	1600	1200	8	57.60000	30.00

Table 3-6 – Scope of variable space for Altera versus Xilinx simulations

4 MEDIAN HARDWARE IMPLEMENTATIONS

Median hardware implementation for the FPGA is broken into two parts: the sorter and the wrapper. The sorter provides the mechanism to take the 25 pixel intensities and sort them for the median. For each sorter, a diagram depicting the structure of the design will be provided. The wrapper provides the mechanism to turn the sorter into a two-dimensional filter by taking the serial pixel stream, buffering it, and providing the sorter with 25 values to sort each clock cycle. The valid combinations of wrappers and sorters will also be enumerated.

4.1 Sorters

While all the implementations discussed have the ability to provide fully sorted lists of the input values, the designs presented take advantage of partial sorting. Partial sorting utilizes the fact that many implementations systematically order the data predictably. Thus, the median index is available well before the complete sort is finished or the median is the only value propagated to the last stage of the sorter.

In this application, the 5x5 median filter has a set size of 25 values. This means that as soon as the 13^{th} value in the sorted output set is available, the work is complete. These designs do not require a fully sorted output set; just the median value. Therefore, each sorter is designed to accept 25 inputs and produce one output.

4.1.1 Radix based elimination

Radix sorter is not a comparison based sort, but an elimination sort. The general radix sorter splits data into separate bins based on each bit: from most significant to least significant [9]. This binning breaks the input set into ordered groups; the process is applied to each bin of the next most significant bit. Each final group represents a unique value from the input set and each group may have more than one value if there were duplicate values.

The radix based elimination implementation leverages the single output requirement by performing bit-by-bit elimination of the non-median values from the input set. Each stage serves to eliminate values that cannot be the median; the number of stages is determined not by the number of inputs, but by the bit-width of the inputs. Each stage focuses on a bit index, working from the most to the least significant bit through the successive stages. A flag bit is used for each input to indicate whether it has been eliminated at any of the prior stages. If the flag bit is set, the input is not considered as a potential median. Each stage consists of three substages: the add substage sums the indexed bit at that stage of each input; the compare substage determines if the generated sum is greater than n/2 by setting a threshold bit; and the multiplex stage assigns each input a value for the next stage. If the flag bit has not yet been set, the index bit and the threshold bit are compared. If the two bits match, the input is allowed to proceed to the next stage unchanged. If the index bit is low, but the threshold bit is high, the dirty bit is set and the input is assigned the minimum possible value. If the index bit is high, but the threshold bit is low, the dirty bit is set and the input is assigned the maximum possible value. If more than one input emerges from the last stage with their dirty bits false, the inputs are identical, and the first is used as the median. The latency for the module is three clock cycles per stage, with the number of stages based on the input pixel bit-depth plus one. The basic hardware structure is illustrated in Figure 4-1 and is an implementation for 8 bit pixel depth. The first eight stages are elimination stages while the final search stage is used to select a non-dirty input. Figure 4-1 also illustrates the subcomponents of the elimination and search stages.

The implementation previously explained allows for a new 25 pixel set to be accepted at the input stage every clock cycle. The pipeline design of the stages as well as their substages allows the use of this system for real-time serial pixel processing. The latency is also pixel depth times three plus three pixel cycles. So, for 8, 12, and 16 bit pixels the latencies are 27, 39, and 51 cycles, respectively.

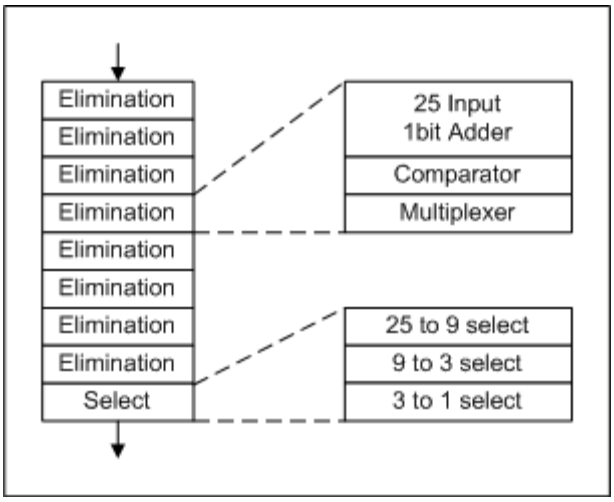

Figure 4-1 – General Structure of radix based elimination

4.1.2 Systolic system

Systolic system is one of the selected designs composed entirely of simple two input or 2x2 sorters. These 2x2 sorters take two input values and sort them low to high for output. The idea behind the arrangement of the 2x2 sorters is to allow the values to percolate to their correct locations by passing them through enough stages of 2x2 sorters; in effect, a pipelined and parallel emulation of a bubble sort. Inputs are fed to a stage of 2x2 sorters, numbered 0 to n/2 from top to bottom. Sorter i generates a smaller and a larger output, with the smaller output feeding sorter i-1 of the next stage and the larger output feeding sorter i+1 of the next stage. After n such stages, a sorted output emerges. Since the entire design is constructed from 2x2 sorters and delay registers, a fresh set of inputs can arrive every clock cycle. Figure 4-2 depicts the systolic network using 13-12 networks modules. The internal structure of 13-12 networks is also illustrated in Figure 4-2 using sorter (S) blocks and buffer (B) blocks representing a sequential collection of 13 2x2 sorters then 12 2x2 sorters.

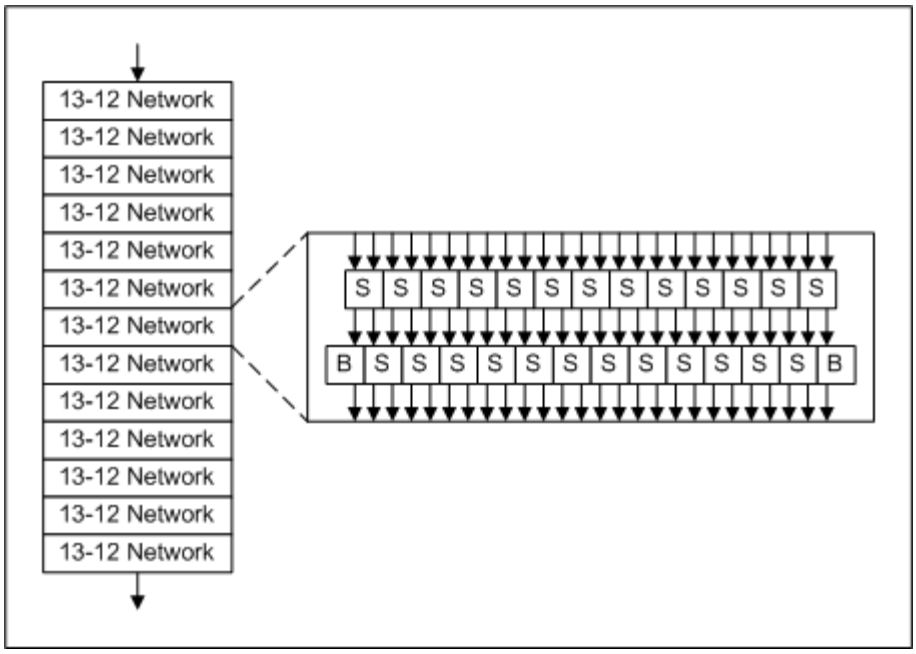

Figure 4-2 – General structure of a systolic system

The systolic technique, as explained in [19], uses repetitive organization of identical structures to achieve the desired outcome. Therefore, this technique does not attempt to reorganize the computational structure for a more efficient implementation, but does eliminate computational structures that are not required for single value median output. This elimination is consistent in general systolic structures as they are used as CPU systems that are not highly configurable. The complete systolic design implemented is shown in Figure 4-3. The figure shows a full 25 input to 25 output systolic sorter that has the output reduction shown in red outline and gray shaded components. Since systolic sorters are intrinsically designed for even input sets, the first input is reduced out of the design to make a 25 input rather than a 26 input sorter. The width of each connection between primary components grows to accommodate the pixel depth.

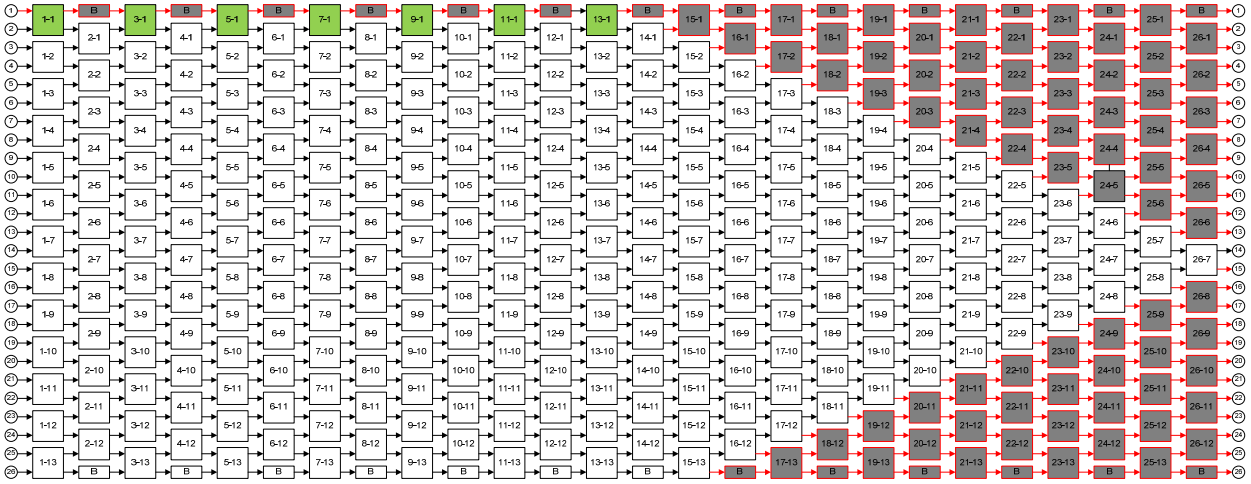

Figure 4-3 – Detailed structure of reduced systolic system

The systolic system implementation previously explained allows for a new 25 pixel set to be accepted at the input stage every clock cycle, while the pipeline design of the stages allows the use of this system for real-time serial pixel processing. The latency of this systolic system is equal to the set size plus one. Additionally, for 8, 12, and 16 bit pixels the latency is constant, but the maximum clockable frequency of the design decreases slightly as pixel depth increases. The most unique characteristic of this design is that the interconnection between primary components has zero cross-over, resulting in simple routing and very high clockable frequencies.

4.1.3 Batcher sorter

Batcher sort [22] is another design based on 2x2 sorters and is found extensively in networking hardware [24]. The premise behind the structure of the algorithm is to ensure that inputs to a sorting block are presorted. In this manner, Batcher sort emulates the principle behind merge sort, but improves on it by utilizing a fully parallel merge mechanism better suited to a hardware implementation [9]. Inputs to the design go to a stage made up of 2x2 sorters, resulting in pairs of inputs being sorted. Groups of two pairs are then fed to a stage comprised of 4x4 sorters. The 4x4 sorter is implemented using a two stage pipeline with a pair of 2x2 sorters in the first stage and a single 2x2 sorter with two delay registers in the second stage. Subsequent

stages follow a similar approach of building mxm sorters, where m is a power of two, from a pipelined arrangement of 2x2 sorters and delay registers. Since each element can accept fresh inputs every clock cycle, the design can be fully pipelined. Each mxm stage introduces a latency of log(m), and the number of stages required is the ceiling of log(n), where n is the number pixels in the input set. The left side of Figure 4-4 depicts the five stages of the batcher sorter using 2, 4, 8, 16, and 32 input switches. The right side of Figure 4-4 depicts how a 4x4 sorter is built from 2x2 sorters and buffers.

The Batcher sorter implementation previously explained allows for a new 25 pixel set to be accepted at the input stage every clock cycle. Additionally, the pipeline design of the stages allows the use of this system for real-time serial pixel processing. Following the structures shown in Figure 4-4, a 2x2 sorter is one stage, a 4x4 sorter is two stages, an 8x8 sorter is three stages, a 16x16 sorter is four stages, and a 32x32 sorter is five stages, totaling 15 stages of 2x2 sorters. Additionally, for 8, 12, and 16 bit pixels the latency is constant, but the maximum clockable frequency of the design decreases slightly as pixel depth increases. While this design has the shortest latency, the nature of the structure results in significant interconnect overlap which results in slower, more complex overall structure once placed and routed on a FPGA.

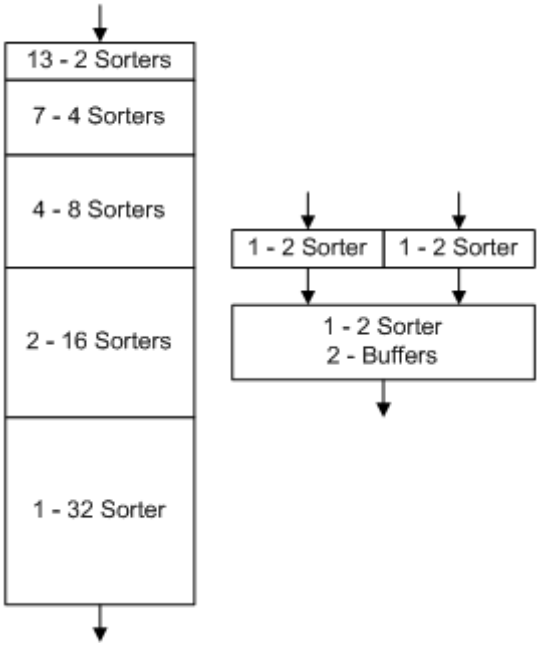

Figure 4-4 – General structure of a Batcher sorter

The complete detailed Batcher sorter implementation is shown in Figure 4-5. This diagram shows multiple aspects of Batcher sorters. The design is for a 32 pixel input set since the Batcher structure can only be made for set sizes that are powers of two. Using the single output approach and decreasing the input set to 25 values, the design is reduced by removing gray shaded components and reducing green shaded components from 2x2 sorters to buffers. The single output portion of the reduction eliminates the most complex wiring of the design at the 15^{th} stage. Note: all square components are 2x2 pixel sorters and all rectangle components are pixel buffers except for previously mentioned shaded components.

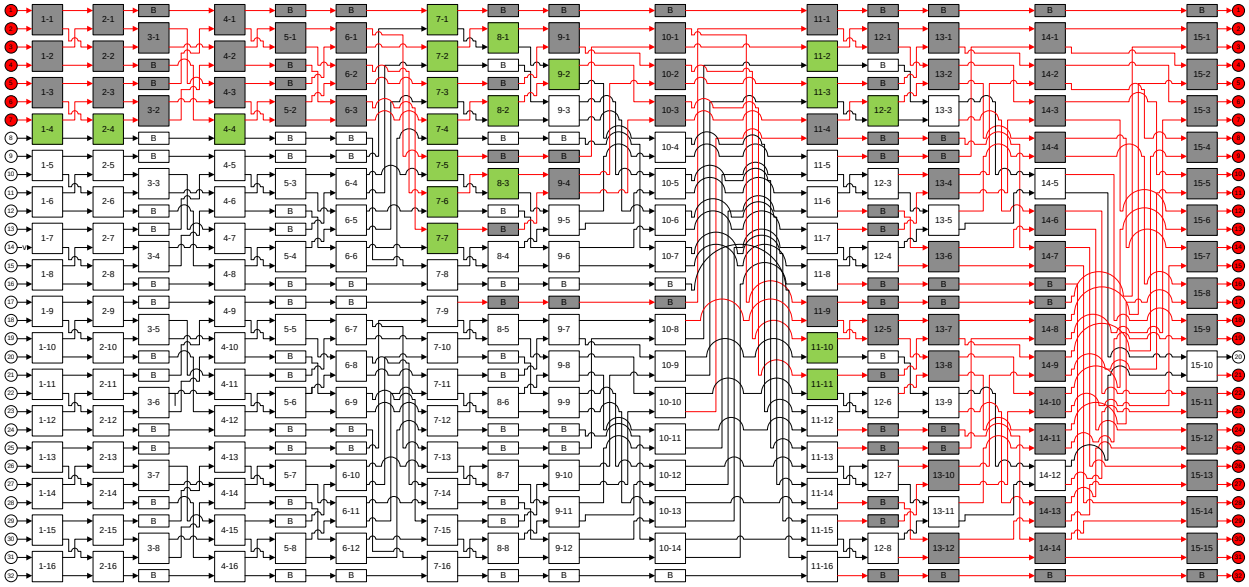

Figure 4-5 – Detailed structure of Batcher sorter

4.1.4 Weavesort

The Weavesort [26] is essentially a parallel insertion sorter that is based in the odd-even sorting techniques presented in [9]. Knuth describes, in [9], how to recursively build an n input sorter from an n-1 sorter. Weavesort is actually a specific implementation of a parallel insertion/selection sorter. Weavesort was originally designed for VLSI implementation and was analyzed in [26]. Weavesort starts with the 25 input pixel set and uses buffers as needed to maintain timing. Figure 4-6 shows a full 25 input Weavesort design with the output reduction to a single median output shown in gray highlighting. The design shows the insertion of each of the 25 pixels one at a time and then the bubbling of the last value to the top, similar to bubble sorting. The design reduces significantly since only the median is needed at the output. The full sorter has a latency of two times the set size minus one, where this design has a latency of 49 cycles. The reduction eliminates not only the buffers and sorters in gray shading, but also the latency. The latency reduction for the median (13^{th} index) is one less than the remaining 12 original outputs. This makes the reduced design 49 minus 11 equaling 38 stages. Again, the design in Figure 4-6 only utilizes 2x2 sorters and buffers, similar to other designs considered.

This design is very easy to build based on the details provided by [9]. The simplicity of the design makes this one of the few designs which can be iteratively built for any input set size. This design also can be built to a specific input set unlike other sorting network systems. In exchange for the simplicity of structure, the design uses a significant amount of both 2x2 sorters and buffers and has the longest latency of any other design considered.

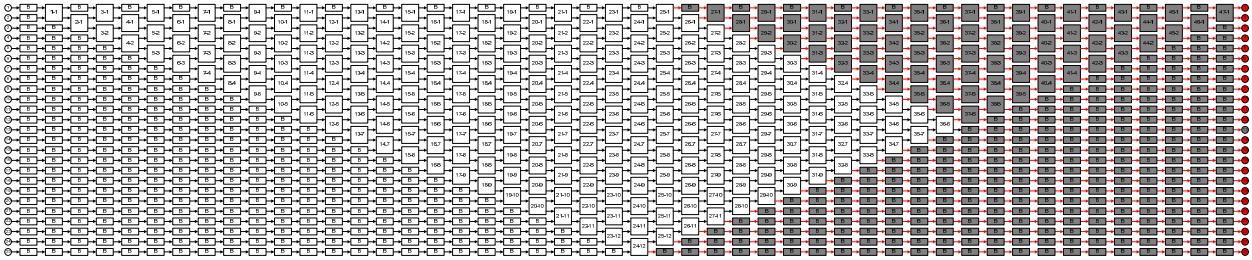

Figure 4-6 – Detailed structure of Weavesort

4.1.5 Insertion sort

The insertion sort design is unique as compared to all other sorters under consideration. The insertion sort design does not accept 25 values per cycle. The insertion sort design accepts five values every clock cycle, where each of the five values are sent to five insertion points of the design, making 25 inputs. The intent of changing the input method allows for the two-dimensional nature of the filter to be exploited by eliminating the need to locally buffer the region filtered. The differences in the other designs will be described in detail in Section 4.2.

The insertion sort design is based on the insertion/bubble sort presented in [9]. The basics of the insertion sort design are similar to the Weavesort presented in [26], but with the variation of not requiring all 25 pixels inputs at every clock cycle. Figure 4-7 shows the full insertion/bubble sort design from [9] with the gray shaded items indicating components that can be removed to create a reduced design that uses only five new pixels each clock cycle. To better illustrate the reduced design of the insertion sort design that is under consideration, Figure 4-8 shows the final component architecture. While this design is very similar to Weavesort, this design utilizes a

different wrapper function that inherently requires fewer resources and also reduces the buffering required at the input of the sorter.

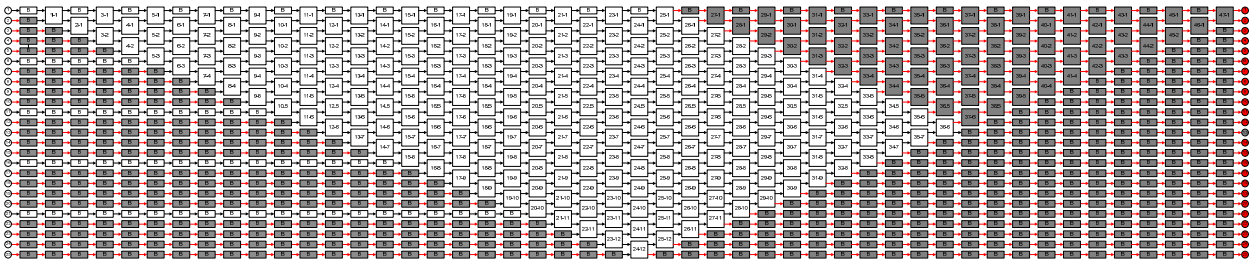

Figure 4-7 – General structure of a insertion sort

The detailed diagram of the insertion sort design is shown in Figure 4-8. This design could ideally be reduced to even eliminate the remaining buffers, but BRAM allocation limitations for the wrapper functions prevented this from happening. While the design accepts only five new pixels each clock cycle, this structure is actually inserting 25 pixels by inserting the five new pixels at the current time step and four time steps ahead. Note: all square components are 2x2 pixel sorters and all rectangle components are pixel buffers. Because of the design's unique structure, latency is calculated slightly differently. The latency is the same as for Weavesort minus n+1, where n is the width or height of the two-dimensional region and n squared is equal to the sorter's input set size. In this specific case, the latency is 38 minus 5 plus 1 equaling 34 stages. The four less cycles are because the five points of insertion in the design are all simultaneously inserted at points that are 0, 1, 2, 3, and 4 sets in the future. The insertion sort design has an added benefit of having no overlapping interconnections, resulting in simple and fast paths between components. This design can also be adapted for a variety of filter sizes where the number of inputs per cycle is the height of the filter and the number of repeated insertions of each input is the filter width.

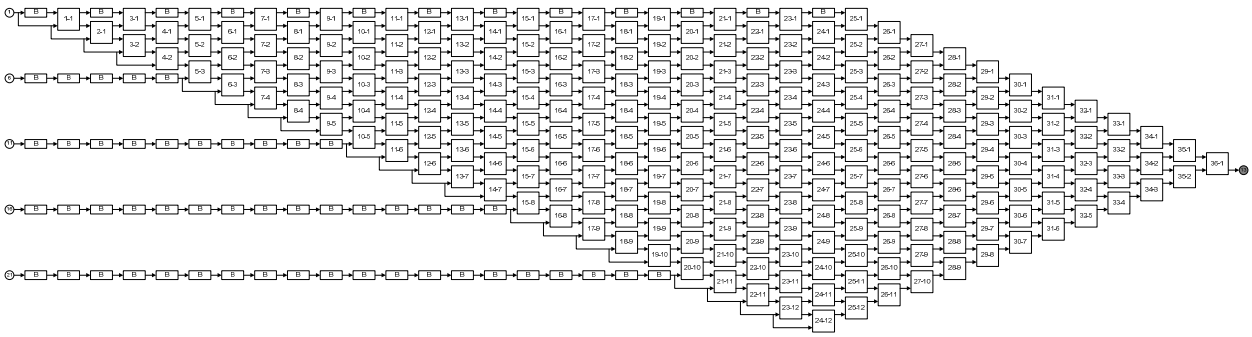

Figure 4-8 – Detailed structure of insertion sort

4.2 Wrappers for conversion to filters

All five hardware designs we have discussed meet the goal of sorting 25 unique values and determining the exact median, but do not filter two-dimensional regions alone. To accomplish this, we needed to develop a specific wrapper to allow the modules to serve as a median filter for the imagery from the sensors.

4.2.1 Row buffer

The number of rows needed in the row buffer is directly associated to the height of the kernel filter. In general, the number of rows needed in the row buffer is one less than the kernel height which in this case is five minus one equaling four rows.

The row buffer accepts one pixel every clock cycle and produces four of the five pixels that represent the leading edge of the kernel filter region. The required wrapper design incorporates a four row buffer storing the four most recent rows of the image data stream. The row buffer when combined with the most recent pixel in the fifth row provides the newest column of data for a 5x5 image region.

The goal is to operate the complete median filter with the minimum possible latency using the most recently input pixel as the lower right corner of the 5x5 region, resulting in the minimum possible latency from the row buffer component of the overall filter. The latency is

dependent on the number of pixels in a row of the imagery, W. It is equal to 2W+3 pixel clock cycles. Image borders are handled via a wrap-around method. Horizontally, the last pixels of the previous row serve as the wrap-around border to the beginning of the next row and vice versa, like a circular buffer. Vertically, the bottom of the previous frame serves as the wrap-around border to the beginning of the next frame and vice versa.

The row buffer is composed of four separate one row buffers. This is done to allow four values, one from each buffer, to be read out every clock cycle, where read operations are then executed from one of the two ports on each row buffer. Simultaneously, the new pixel value is written into one of the four row buffers via the other available port. The row buffers are allocated from BRAM and read/write addresses are generated via counters and multiplexors.

4.2.2 Local buffer

The local buffer stores the five most recent pixels from each row so that the 25 value set can be passed to the sorter every clock cycle. This local buffer is composed of 25 registers that are arranged in a five deep by five wide shifting buffer. This shifts five pixels out and five pixels in every clock cycle. The four of the five new pixels come from the row buffer and with the remaining pixel provided by the input pixel stream. The size of the local buffer is directly associated with the size of the kernel area of the filter which in this case is 5x5. The latency of this component in a wrapper is equal to the kernel width which in this case is five.

4.2.3 Wrapper for 25 input filter

There are two wrappers used to convert all sorters into filters. The first one is the 25 input wrapper, designed to provide a 25 value sorter with the appropriate input data. To achieve the correct wrapper for a 25 value filter, the row buffer and local buffer must be included. The 25 input filter is used for the following sorters: radix based elimination, systolic system, Batcher sorter, and Weavesort. All of these designs accept the entire 25 pixel set every clock cycle. To

accomplish this task, the row buffer must provide the newest column to the local buffer so that the buffer can be sorted and shifted every clock cycle.

4.2.4 Wrapper for 5 input filter

The second of the wrappers is the five input wrapper, designed to provide a 25 input sorter with the appropriate input data. To achieve the correct connections for a 25 input sorter, the row buffer is the only module that needs to be included. This is done because this wrapper is only used for the insertion sort. The insertion sort becomes a filter by connecting the five pixels of the newest column directly to the sorter. This sorter is the single design that only needs the five new pixels, not all 25 for the sort set. It automatically routes the five new values to different time steps in the pipeline to give the correct 25 values at each clock cycle.

4.3 Altera COTS IP core

The designs implemented to perform median filtering are intended to represent a collection of low power designs. While this is true, the fact is that these designs should have a relative comparison as a reference to the currently available commercial off-the-shelf (COTS) hardware designs. COTS designs are intellectual property (IP) that are intended for general designs and are created for specific destination platforms. In this specific case, the COTS IP core available from Altera is used as the relative comparison point to the custom designs.

While the Altera IP is compared to the custom designs, this does not imply that the designs have exactly the same input, output, and behavior; they are similar enough for a basic comparison. In this case, the Altera core has a slightly more complicated input and output than the custom designs. The Altera IP has data ready output and a data valid input. These signals are needed for the design because, unlike the other designs considered, the module does not accept a new input pixel every clock cycle. This module manages the data flow by signaling that it is ready for new pixels and accepts the values when each pixel is indicated as valid. As a result of the inconsistent input pixel stream, the design does not have a constant stream out of the

module and is not able to process a full frame of imagery during the time it takes a full frame to be generated.

To solve this problem, the module would require pre and post buffering to create constant pixel streams; it would also require clocking the module faster than the clock of the originating pixel stream. These add-ons would consume large amounts of resources, but would not give a fair comparison. To perform simulations similar to those being performed on the other designs, the required test benches were modified to accept data based on the input and output signaling, that the module requires. As a result, the time period analyzed for the Altera IP is slightly larger than the other designs because the Altera IP module is clocked at the pixel speed.

5 Results and Analysis

Chapter 5 provides the results and analysis of all five custom designs as well as the Altera IP core. Additionally, a small selection of resolution variants, implemented in the Xilinx ISE environment, is analyzed to show a relatively similar progression as compared to the Altera simulations. Two stages of testing were completed during the analysis of the median filter implementations. The first round focused on sorters alone for a fair comparison [26]. The second round used the lessons learned from the first round of sorter analysis, as well as encapsulating the sorters into wrappers to make comparisons with the final designs including the Altera IP core.

In order to better understand the behavior of the custom designs, a logic free design was synthesized to acquire static, I/O, and dynamic power offsets of 569.72, 41.34, and 0 mW, respectively for Altera destination FPGA. The total Altera offset is 611.06 mW of which none is dynamic power since there were no active components in a logic free design. The static power offset for the Xilinx destination system is 112.38 mW, which is the total offset. As a result of removing the static FPGA power costs, the effects of the designs are much easier to observe. This offset procedure is similar to removing the DC component from an AC signal on an oscilloscope.

5.1 Image content

These simulations were performed to provide a comparison of power consumption between different imagery. Two different test bench types were created for the research and are from different imagers. This section is intended to highlight the similarities and differences between these imagery types.

5.1.1 Results

The simulations were performed for a single resolution and design of 800x600 with 8 bit pixel resolution at 60 frames per second using the Batcher design. This results in a comparison that uses two different source images for the simulation while holding all other variables constant. Figure 5-1 shows all power categories for each of the source images. The specifics of the test benches were covered in Section 3.1.3.

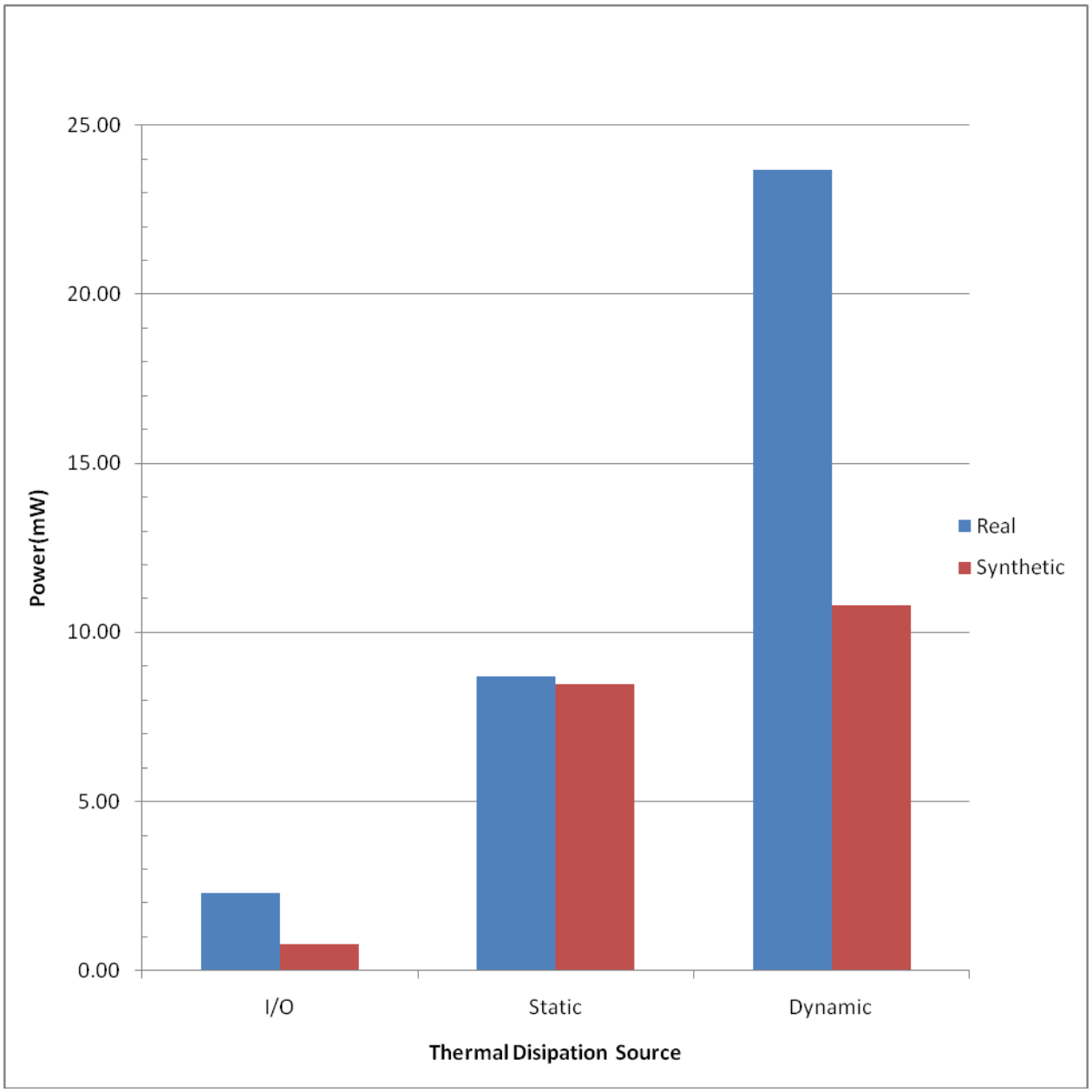

Figure 5-1 – Power consumption for Batcher design using different imagery

5.1.2 Analysis

Reviewing the results shown in Figure 5-1 allows two points to be inferred:

- the actual image content can have a significant effect on the dynamic power, but not the static power
- even though the statistical properties of the scintillation noise in the synthetic imagery are similar to the target sensor imagery, the power dissipated filtering synthetic imagery differs substantially compared to real imagery

It is suspect that the difference in dynamic power between the real and synthetic imagery is that the synthetic test imagery is generated using a sensor that has significantly better signal to noise ratio (SNR) than the actual intensified sensor. The low SNR of the actual sensor is a natural side effect of the intensification process; it causes pixel intensity variances to be locally larger as compared to the synthetic imagery. The conclusion of this analysis is that the local uniformity of imagery has a significant effect on active power consumption and no effect on static power.

5.2 Sorters

The first round of implementation, testing, and analysis had the scope of the merge, radix, systolic, and Batcher sorting modules alone. As the wrapper was a common element for all of these designs, it was ignored for this portion of evaluation to focus on the algorithmic implementations. Each sorter was evaluated using four metrics to determine the modules' capability to operate within the scope of the destination application: latency, hardware resources, power dissipation, and clock frequency. The first round evaluation was completed and published in [26].

5.2.1 Results

Figure 5-2 through Figure 5-5 present comparisons of the sorter designs for four metrics. Figure 5-2 shows the constant input to output latency of the four designs. Figure 5-3 shows the hardware resource requirements for the four sorter designs and Figure 5-4 indicates the power dissipation of each sorter. Figure 5-5 indicates maximum clocking frequency to verify whether the designs meet the 140 MHz minimum requirement.

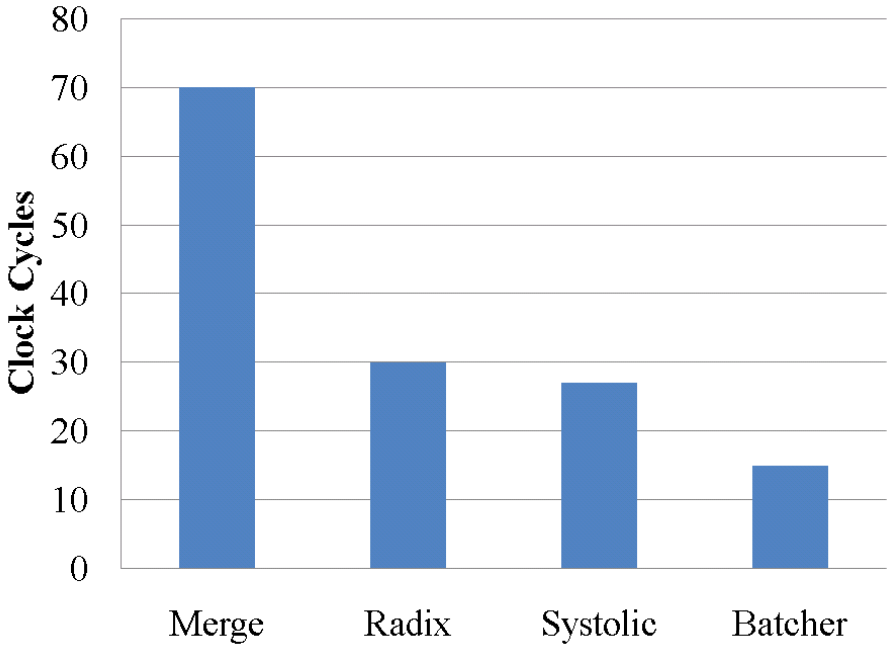

Figure 5-2 – Latency of the initial sorter designs

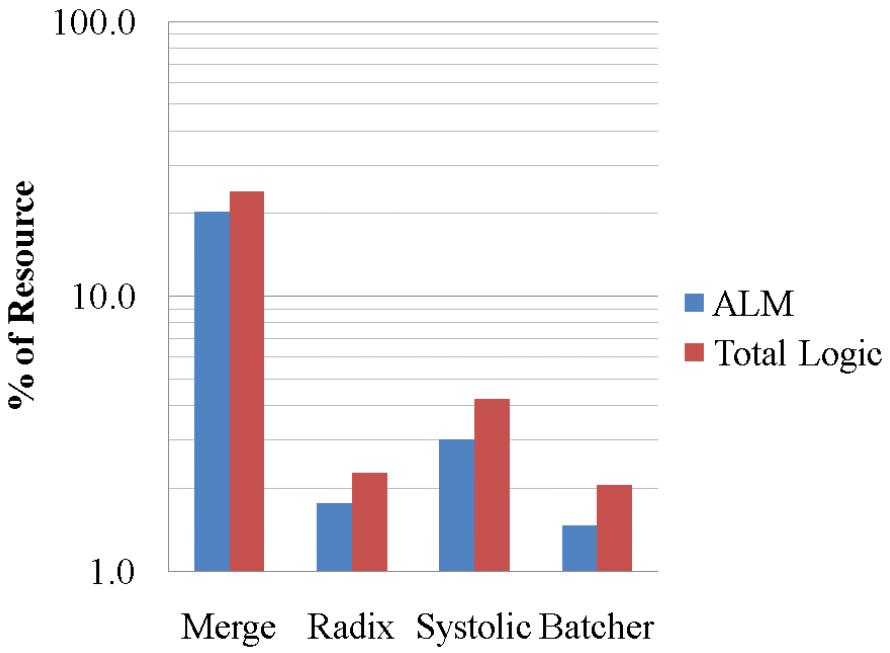

Figure 5-3 – Resource requirements of the initial sorter designs

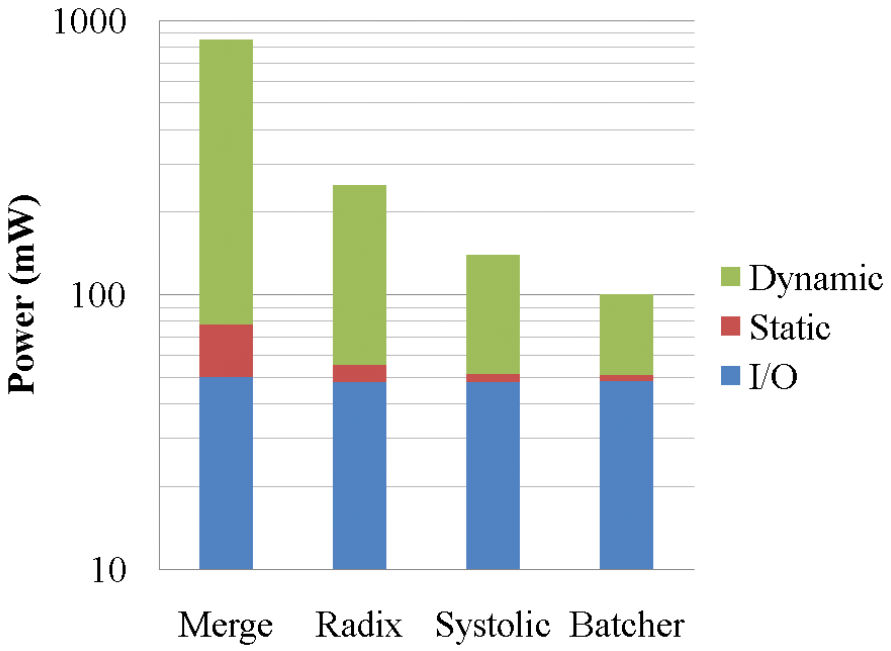

Figure 5-4 – Power dissipation of the initial sorter designs

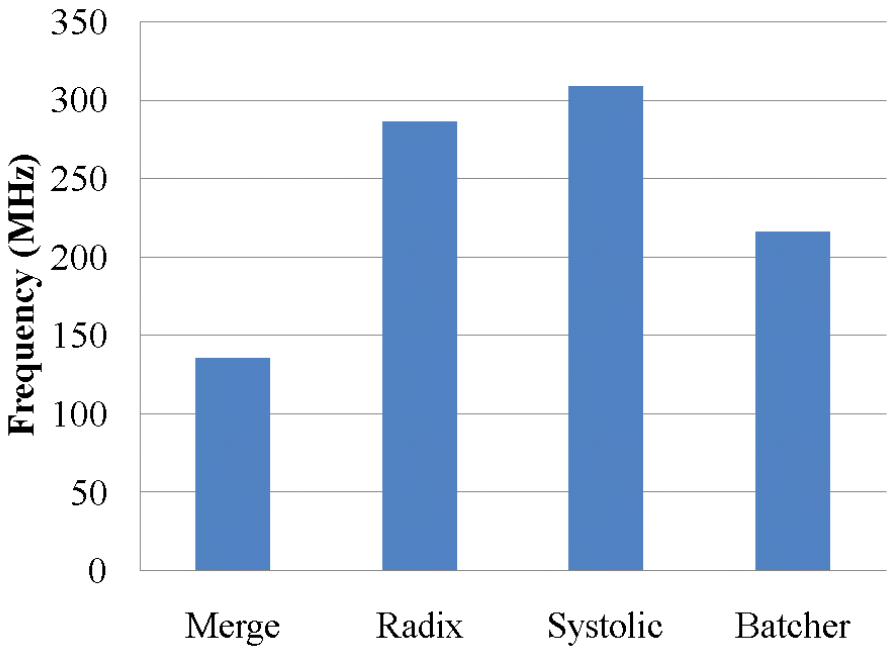

Figure 5-5 – Maximum clock frequency of the initial sorter designs

5.2.2 Analysis

The lessons learned from the first round of designs and implementations are significant. While the merge sort technique is known to be very good in terms of software implementation [7], it proved to have a poor hardware implementation. Not only does it consume nearly ten times the power of the other sorters, but it also does not meet the minimum required clock frequency needed. Although the first round was only a limited design set, the results suggest that the best performing designs have the characteristic of sorting networks. This observation resulted in the selection of additional designs similar to sorting networks. As far as power consumption is concerned, the resources required only have a basic correlation to the total power consumption. The merge sort results were included as a point of reference for evaluation. However, the results caused the merge sort implementation to be eliminated from consideration. Merge sort resources were about ten times the other designs, but the radix design used the least resources while having 50-100% more power consumption than the Batcher and systolic sorters respectively. The differences are likely based on the fundamentally different structures of the components in each design. The merge, Batcher, and systolic designs behave similarity because they use 2x2 sorters as their fundamental components while radix uses bitwise comparison and multiplexing as fundamental components to perform sorting.

5.3 Initial filter simulations and analyses

A collection of initial simulations and analyses were performed to present the effects of varying:

- Frame rate
- Aspect ratio
- Pixel depth

These simulations were completed to better understand how the three dimensions of filters affect power dissipation independently.

5.3.1 Frame rate

The variation of frame rate is actually a surrogate for variation of clock frequency while holding the design and implementation constant. The only variation is with the pixel clock frequency to allow a better understanding of how clock frequency affects overall power dissipation.

5.3.1.1 Results

The results of simulation of frame rate variation were performed for the Batcher design. The frame rates considered were selected to show the full range of clock frequencies that can be achieved with the design. In this case, the Batcher design was simulated for 30, 60, 120, 240, and 540 frames per second. This collection encompasses standard sensor frame rates that can be expected for an 800x600 with 8 bit pixel depth resolution and specialty frame rates up to the maximum possible simulation speed. The corresponding pixel clock rates for the simulated frame rates are shown in Table 5-1. Figure 5-6 shows the total power consumption results for each of the frame rates separated into the primary components of dynamic, I/O, and static power. Figure 5-7 illustrates the power consumption trend for a fixed resolution and pixel depth. The figure also provides the slope and intercept for a linear trend line on the active, static, and total power dissipation.

Name	Width	Height	Clock (MHz)	FPS	Mb/s	Depth
SVGA	800	600	14.4	30	109.86	8
SVGA	800	600	28.8	60	219.73	8
SVGA	800	600	57.6	120	439.45	8
SVGA	800	600	115.2	240	878.91	8
SVGA	800	600	259.2	540	1977.54	8

Table 5-1 – Frame rate simulation details

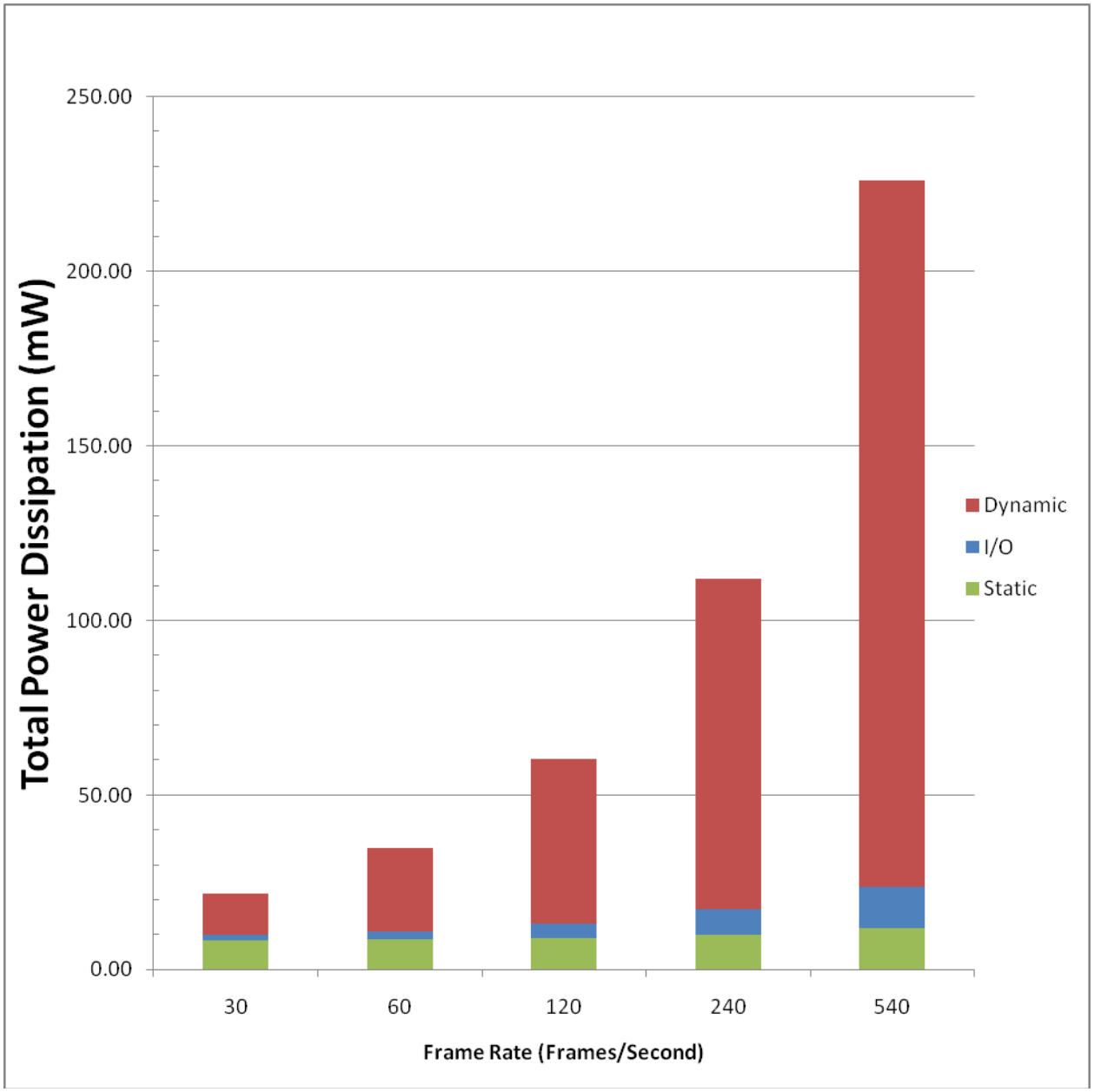

Figure 5-6 – Total power consumption for Batcher design over multiple frame rates

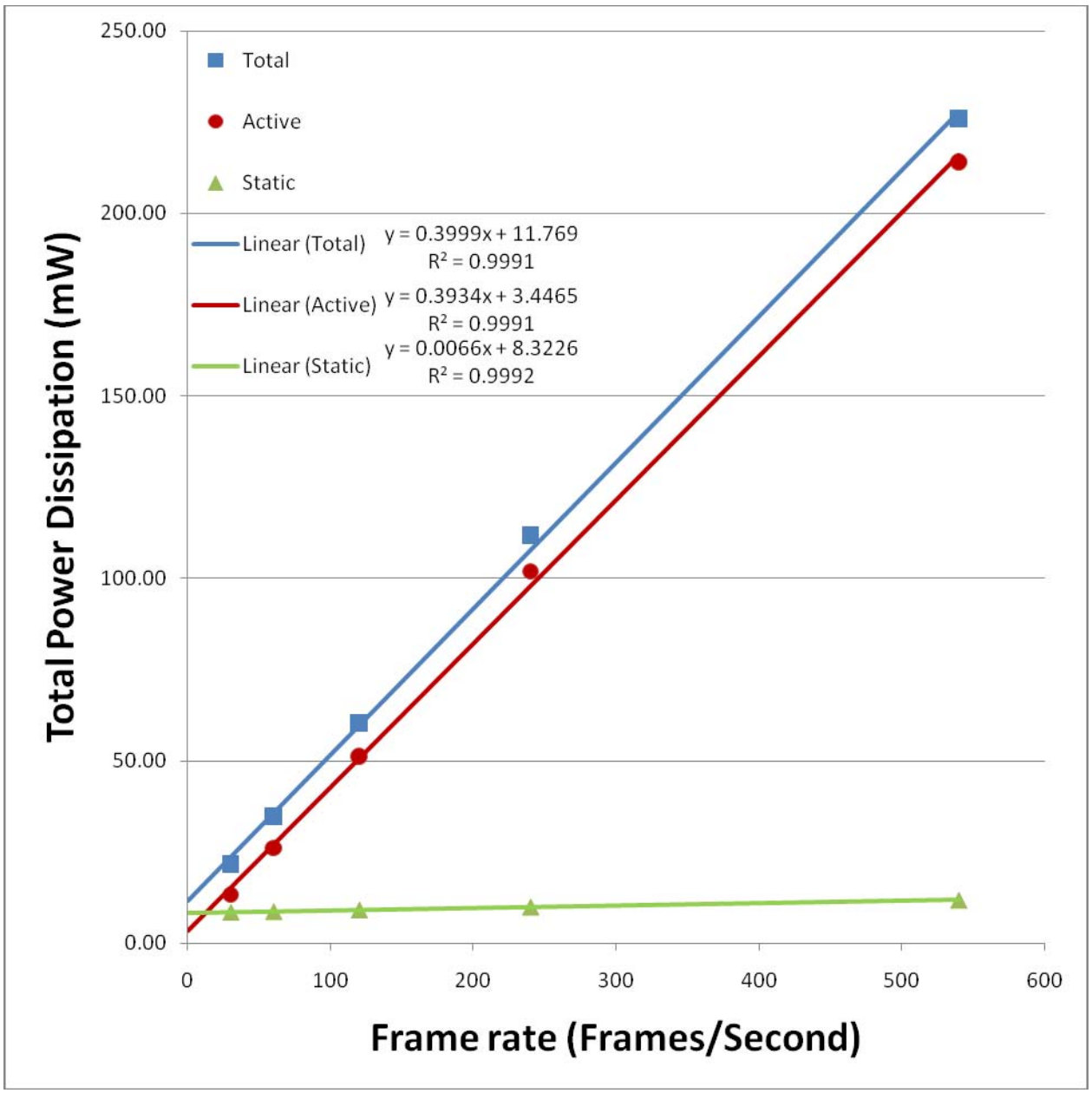

Figure 5-7 – Power consumption trend for Batcher design over multiple frame rates

5.3.1.2 Analysis

As can be seen from Figure 5-6, the static power is nearly constant across all frame rates; the linear fit of the data gives a small positive slope. This non-constant power dissipation is the

result of variations during the place and route phase of synthesis that occur because the design tools adjust to accommodate higher frequencies.

Figure 5-7 shows that the active power also trends linear growth; the effect of higher frame rate is more significant on the growth of active power as would be expected. While this analysis was performed on a single design and configuration, I expect the trend would be the same for other configurations or designs. This expectation is supported by the results that are presented in Section 5.4.1. The clock rate is a core component to understanding how different designs are affected by image resolution.

5.3.2 Aspect ratio

The aspect ratio is actually a surrogate for variation of BRAM blocks while holding all other design and implementation variables constant. The variation of BRAM blocks allows for a better understanding of how row buffering affects overall power dissipation. Specifically BRAM usage is tied to number of pixels in the image row and the pixel bit depth. These simulations hold the total pixel count, and hence the pixel clock, as constant as possible while maintaining constant row widths throughout the image. An 800x600 image, with 8 bit pixel resolution, was chosen as the 4:3 aspect ratio image and was used as a pixel count baseline for the remainder of the aspect ratios. Again, the Batcher sorter was used for the comparison.

5.3.2.1 Results

The resolutions selected are actually from two different categories: standard and custom aspect ratios. Table 5-2 shows the details of the aspect ratios selected including the width, height, ratio, and exact clock frequency. The italic entries are the entries corresponding to standard aspect ratios used in sensor and display technologies, while the remaining entries are custom aspect ratios that were selected to test the limits of the row buffer designs.

Figure 5-8 shows the total power consumption results for each of the aspect ratios separated into the primary components of dynamic, I/O, and static power. Figure 5-9 illustrates the power

consumption trend for a fixed pixel clock and pixel depth. The figure also provides the slope and intercept for a linear trend line on the active, static, and total power dissipation.

Ratio	Width	Height	M9Ks	M512Ks	Clock (MHz)	Mb/s
1:12	200	2400	1	0	28.80000	219.73
1:3	400	1200	2	0	28.80000	219.73
1:1	*693*	*693*	*3*	*0*	*28.81494*	*219.84*
5:4	*775*	*620*	*3*	*0*	*28.83000*	*219.96*
4:3	*800*	*600*	*3*	*0*	*28.80000*	*219.73*
3:2	*849*	*566*	*4*	*0*	*28.83204*	*219.97*
16:10 (8:5)	*880*	*550*	*4*	*0*	*29.04000*	*221.56*
16:9	*912*	*513*	*4*	*0*	*28.07136*	*214.17*
16:3	1600	300	6	0	28.80000	219.73
64:3	3200	150	12	0	28.80000	219.73
256:3	6400	75	23	0	28.80000	219.73
768:1	19200	25	12	1	28.80000	219.73
19200:1	96000	5	0	6	28.80000	219.73

Table 5-2 – Aspect ratio simulation details with standard ratios in italics

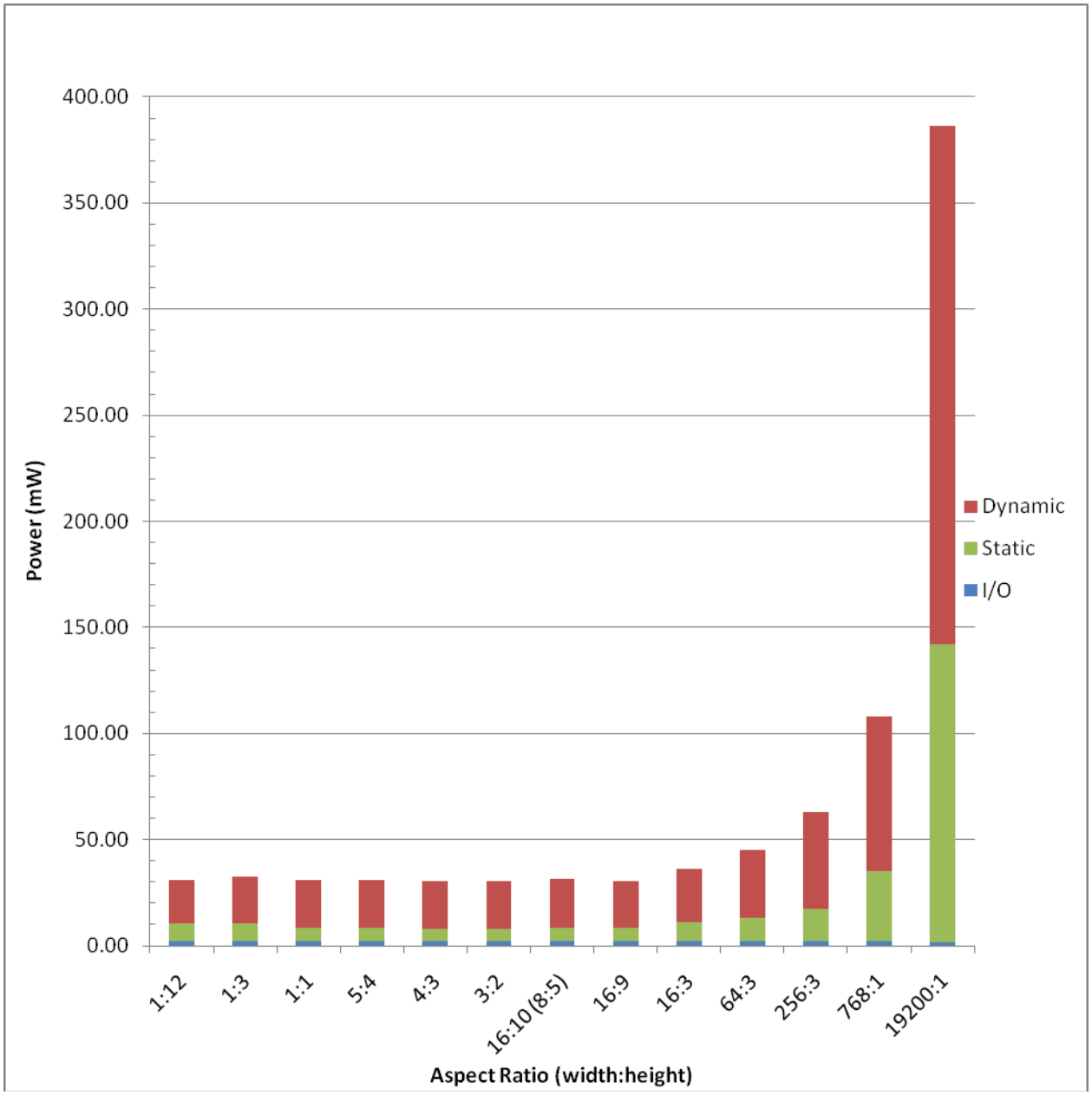

Figure 5-8 – Total power consumption for Batcher design over multiple aspect ratios

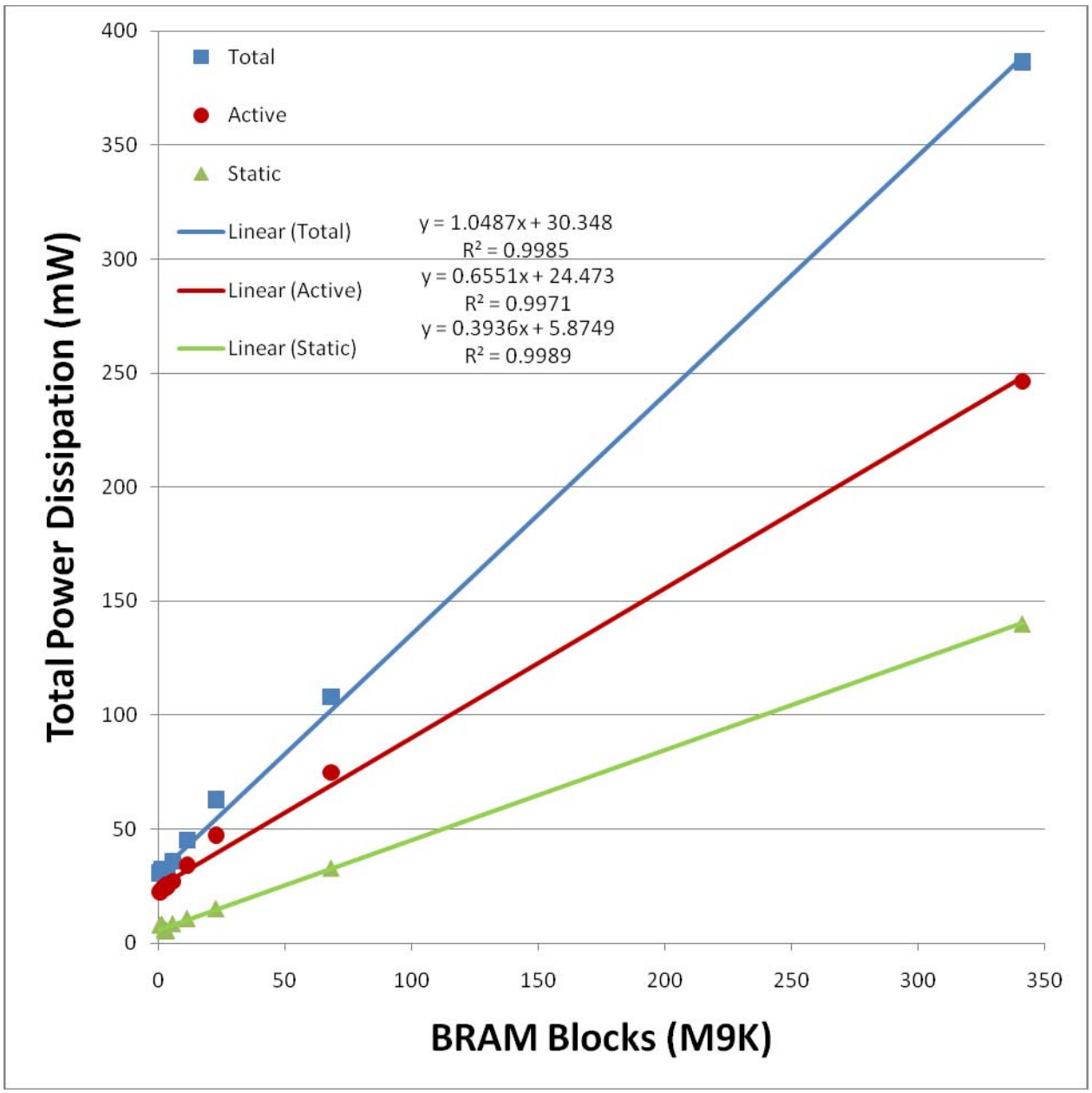

Figure 5-9 – Power consumption trend for Batcher design over multiple aspect ratios

5.3.2.2 Analysis

The variation of aspect ratio in no way affects the sorting component of the overall filter; it affects the row buffer only. The separation of power consumption behavior is the reason why this collection of simulations was completed. The total power dissipation is depicted in Figure

5-8 with separate subcategories for all three dissipation methods, showing that both static and active power dissipation increases with ratio width even when the clock rate is held constant. The power consumption increase is solely the result of storing more pixels in the row buffer component of the filter's wrapper. Figure 5-9 shows static power grows linearly as the number of M9K (9000 bit) BRAM blocks increase with a slope of 0.3936 mW/M9K and active power grows linearly as the number of M9K BRAM blocks increase with a slope of 0.6551 mW/M9K. This means that the BRAM affects both the static and dynamic power dissipation independent of the clock frequency.

Note that the power required for row widths up to 1600 pixels is relatively constant, between 30 and 40 mW, using 1-6 M9ks. Significant power consumption occurs only when a large amount of M9K and M512K BRAM blocks are utilized as the row width increases. Figure 5-9 depicts the trend line for the total power consumption based on bandwidth. It also shows that image width is the primary influence on static power behavior since pixel clock has minimal impact on static powers. The nature of the BRAM block allocation creates a discrete step behavior in the power dissipation trend. This discrete step behavior is not unique to median filtering, but for any neighborhood-based filter employing a row buffer scheme.

5.3.3 Pixel depth

The variation of pixel depth is actually a surrogate for variation of FPGA resources including LUTs (look up tables), BRAM blocks, and flip flops. The pixel depth simulations only vary the depth of the input pixel bus, but this variation changes the usage of all three resources. Simulations were performed to evaluate the performance of a single design as the pixel depth was varied while all other variables were held constant, including the pixel clock rate. The simulations include 800x600 with 60 FPS frame rate as the basis for the total pixel clock rate and the Batcher sorter was selected as the design.

5.3.3.1 Results

The pixel depths selected cover the range common to standard digital night vision sensors. Table 5-3 shows the details of the pixel depths selected including the width, height, ratio, and exact clock frequency. Figure 5-10 shows the total power consumption results for each of the pixel depths separated into the primary components of dynamic, I/O, and static power. Figure 5-11 illustrates the power consumption trend for a fixed pixel clock and resolution. The figure also provides the polynomial trend line on the active, static, and total power dissipation.

Name	Width	Height	Clock (MHz)	FPS	Mb/s	Depth
SVGA	800	600	28.8	60	219.73	8
SVGA	800	600	28.8	60	329.59	12
SVGA	800	600	28.8	60	439.45	16

Table 5-3 – Pixel depth simulation details

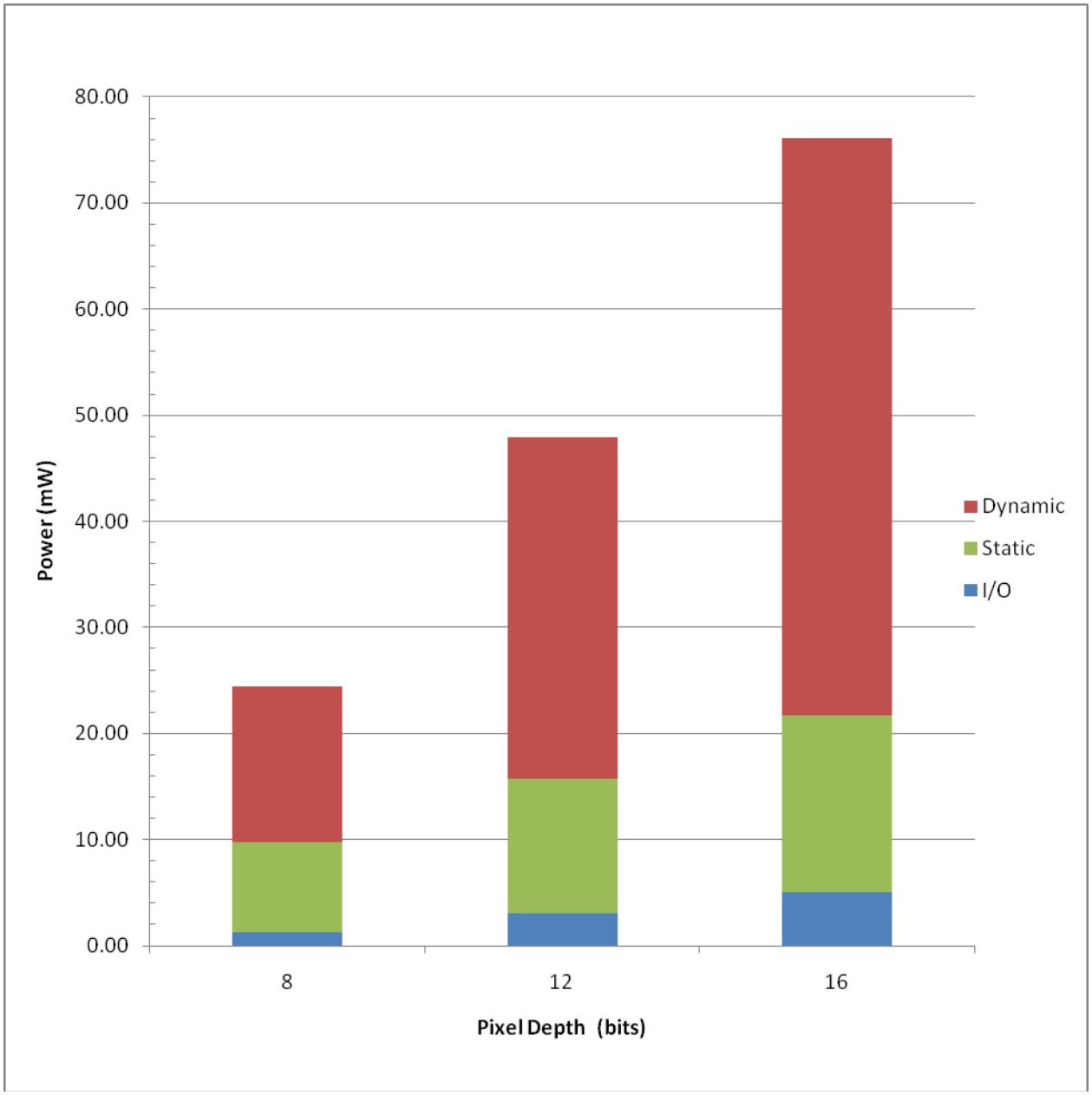

Figure 5-10 – Total power consumption for Batcher design over multiple pixel depths

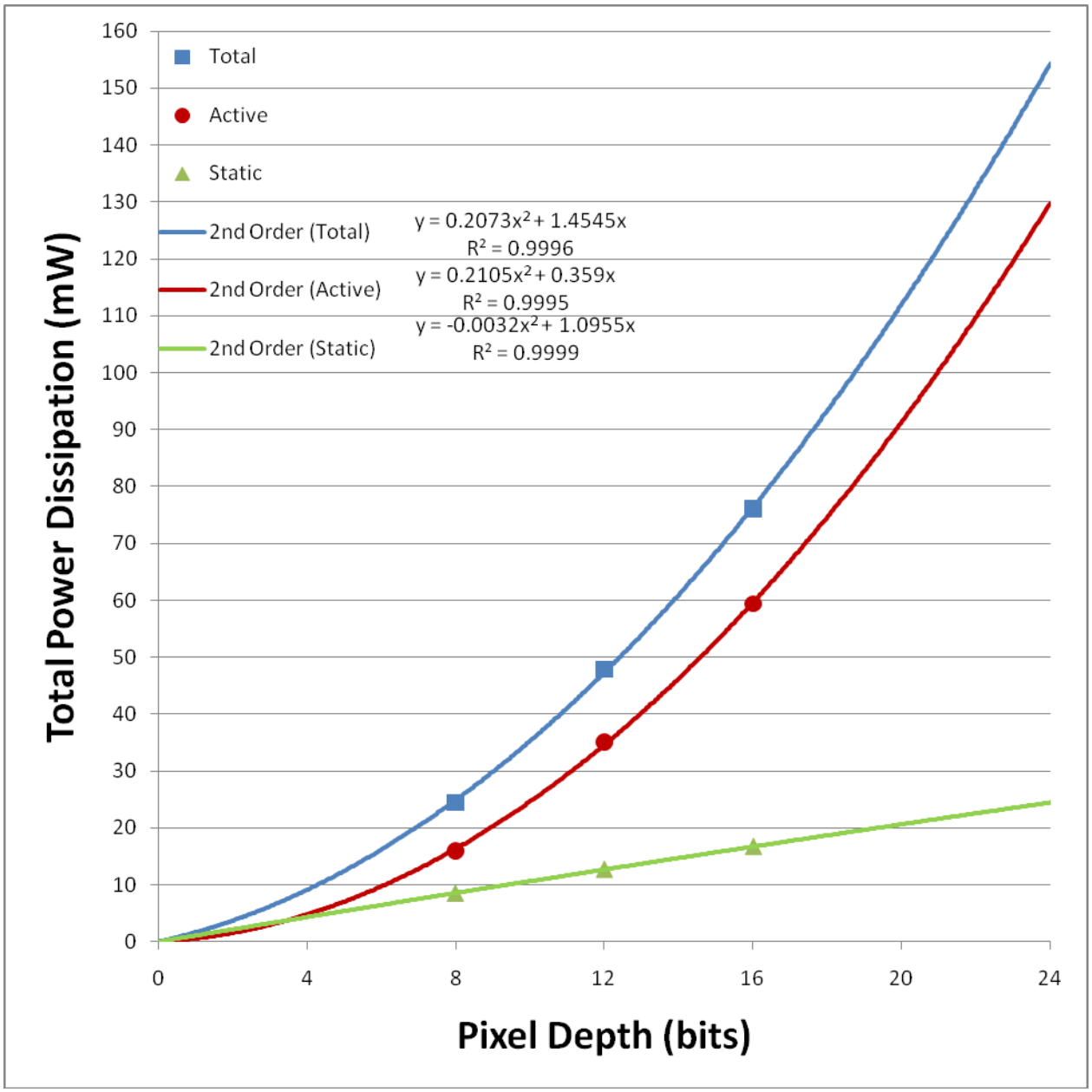

Figure 5-11 – Power consumption trend for Batcher design over multiple pixel depths

5.3.3.2 Analysis

The variation of pixel depth in no way affects the clock rate of the overall filter. However, increasing pixel depth increases:

- the number of BRAM blocks for the row buffer
- the number of flip flops for the local buffer
- the number of LUTs for the sorter

The diversity of power consumption behavior is the reason why this collection of simulations was completed. The total power dissipation is depicted in Figure 5-10 with separate subcategories for all three dissipation methods. The figure also shows that both static and active power dissipation increases with bit depth despite the constant clock rate. The power consumption increase is the result of storing more pixels in the row buffer, increasing the flip flop count in the local buffer, and increasing the LUT usage in the sorter.

The static and active powers both grow with a second order polynomial trend. They both are the summation of linear row buffer growth, linear flip flop growth, and linear LUT growth. The combination of static power growth is nearly linear since the all hardware grows linearly. However, the active power's polynomial growth results from the complexity increase in the sorting module specifically in the comparison of the component of each 2x2 sorter.

Figure 5-11 shows the trend line for active, static, and total power. The figure indicates that pixel depth has a polynomial affect on total power when clock rate and resolution are held constant. The active power is the primary source of second order of the total power trend and static power is the primary source of the first order of the total power trend.

5.4 Primary filter simulations and analyses

A set of primary simulations and analyses were performed to evaluate the effects of different:

- Sorter designs
- Destination platforms
- Image content

The pixel clock rate simulations explain the first half of the power dissipation behavior when changing the input image resolution. The aspect ratio simulations explain the second half of the

power dissipation behavior when changing the input image resolution. This is pertinent since variations in resolution affect the number of BRAM blocks and the clock frequency. Consider these initial analyses when reviewing the following sections.

5.4.1 Sorter designs

An important aspect to consider when analyzing any imagery based processing is the resolution of the image being processed. In this case, a small collection of standard resolutions were selected to represent the range of FPA resolutions that would utilize this type of image processing. A review of the standard resolutions was performed and it resulted in the information shown in Table 5-4. The maximum clock speed that any design will be capable of is limited by the FPGA resources. The specific resource that limited all designs considered was the maximum clockable speed of the BRAM. Its speed is limited to 265 MHz for the selected speed rating on the Altera Stratix FPGA selected. The standard resolutions within the speed limitation for 30 and 60 frames per second are shown in Table 5-4 using italics. While resolutions are semi-formalized, frame rates are much less standardized. To separate frame rate from the considerations, Table 5-4 shows the megapixels for each resolution independent of frame rate.

Name	Width	Height	Resolution (MP)	Pixel Clock 30 FPS (MHz)	Pixel Clock 60 FPS (MHz)	Aspect Ratio
QQVGA	160	120	0.019200	*0.576000*	*1.152000*	4:3
CGA	320	200	0.064000	*1.920000*	*3.840000*	8:5
QVGA	320	240	0.076800	*2.304000*	*4.608000*	4:3
VGA	640	480	0.307200	*9.216000*	*18.432000*	4:3
NTSC	720	480	0.345600	*10.368000*	*20.736000*	3:2
WGA	800	480	0.384000	*11.520000*	*23.040000*	5:3
WVGA	856	480	0.410880	*12.326400*	*24.652800*	107:60
PAL	768	576	0.442368	*13.271040*	*26.542080*	4:3
SVGA	800	600	0.480000	*14.400000*	*28.800000*	4:3
WSVGA	1024	600	0.614400	*18.432000*	*36.864000*	128:75
XGA	1024	768	0.786432	*23.592960*	*47.185920*	4:3
HD720	1280	720	0.921600	*27.648000*	*55.296000*	16:9
XGA+	1152	864	0.995328	*29.859840*	*59.719680*	4:3
WXGA	1280	800	1.024000	*30.720000*	*61.440000*	8:5
SXGA-	1280	960	1.228800	*36.864000*	*73.728000*	4:3
WXGA+	1440	900	1.296000	*38.880000*	*77.760000*	8:5
SXGA	1280	1024	1.310720	*39.321600*	*78.643200*	5:4
SXGA+	1400	1050	1.470000	*44.100000*	*88.200000*	4:3
WSXGA+	1680	1050	1.764000	*52.920000*	*105.840000*	8:5
UXGA	1600	1200	1.920000	*57.600000*	*115.200000*	4:3
HD1080	1920	1080	2.073600	*62.208000*	*124.416000*	16:9
WUXGA	1920	1200	2.304000	*69.120000*	*138.240000*	8:5
QXGA	2048	1536	3.145728	*94.371840*	*188.743680*	4:3
WQXGA	2560	1600	4.096000	*122.880000*	*245.760000*	8:5
QSXGA	2560	2048	5.242880	*157.286400*	314.572800	5:4
WQSXGA	3200	2048	6.553600	*196.608000*	393.216000	25:16
QUXGA	3200	2400	7.680000	*230.400000*	460.800000	4:3
WQUXGA	3840	2400	9.216000	276.480000	552.960000	8:5
HSXGA	5120	4096	20.971520	629.145600	1258.291200	5:4
WHSXGA	6400	4096	26.214400	786.432000	1572.864000	25:16
HUXGA	6400	4800	30.720000	921.600000	1843.200000	4:3
UHDTV	7680	4320	33.177600	995.328000	1990.656000	16:9
WHUXGA	7680	4800	36.864000	1105.920000	2211.840000	8:5

Table 5-4 – List of standard resolutions

The limitations of ModelSim became apparent through experimental testing. An image of 1600x1200 (UXGA) is the largest resolution that could be generated as a test bench and loaded by the simulation tool without tiling of lower resolutions. This limitation was determined by trying the resolutions of 2048x1536 (QXGA) and 3200x2400 (QUXGA), both of which failed to load on a 64 bit, 8 core, 16 GB PC. These resolutions were attempted because they are the logical resolution steps above 1600x1200. Initially, the 3200x2400 resolution was attempted because it was the largest resolution that could be created using the synthetic imagery generated for the synthetic test bench. This resolution was converted into a simulation test bench, but the simulation tool was not able to successfully load the file error free. The 2048x1536 resolution was also converted to a test bench and was properly loaded by the simulation tool, but the tool could not process the input because it required more than the 4 GB of RAM that the OS was capable of providing to a single process. The 1600x1200 resolution was successful, but temporarily required 3.8 GB of RAM to load the test bench file. Based on the restrictions imposed by the FPGA technology, the simulation tool, and the fixed aspect ratio, the clock frequencies in Figure 5-12 were selected to represent the range of viable simulations performed. The included resolutions are: 320x240 (QVGA), 640x480 (VGA), 800x600 (SVGA), and 1600x1200 (UXGA).

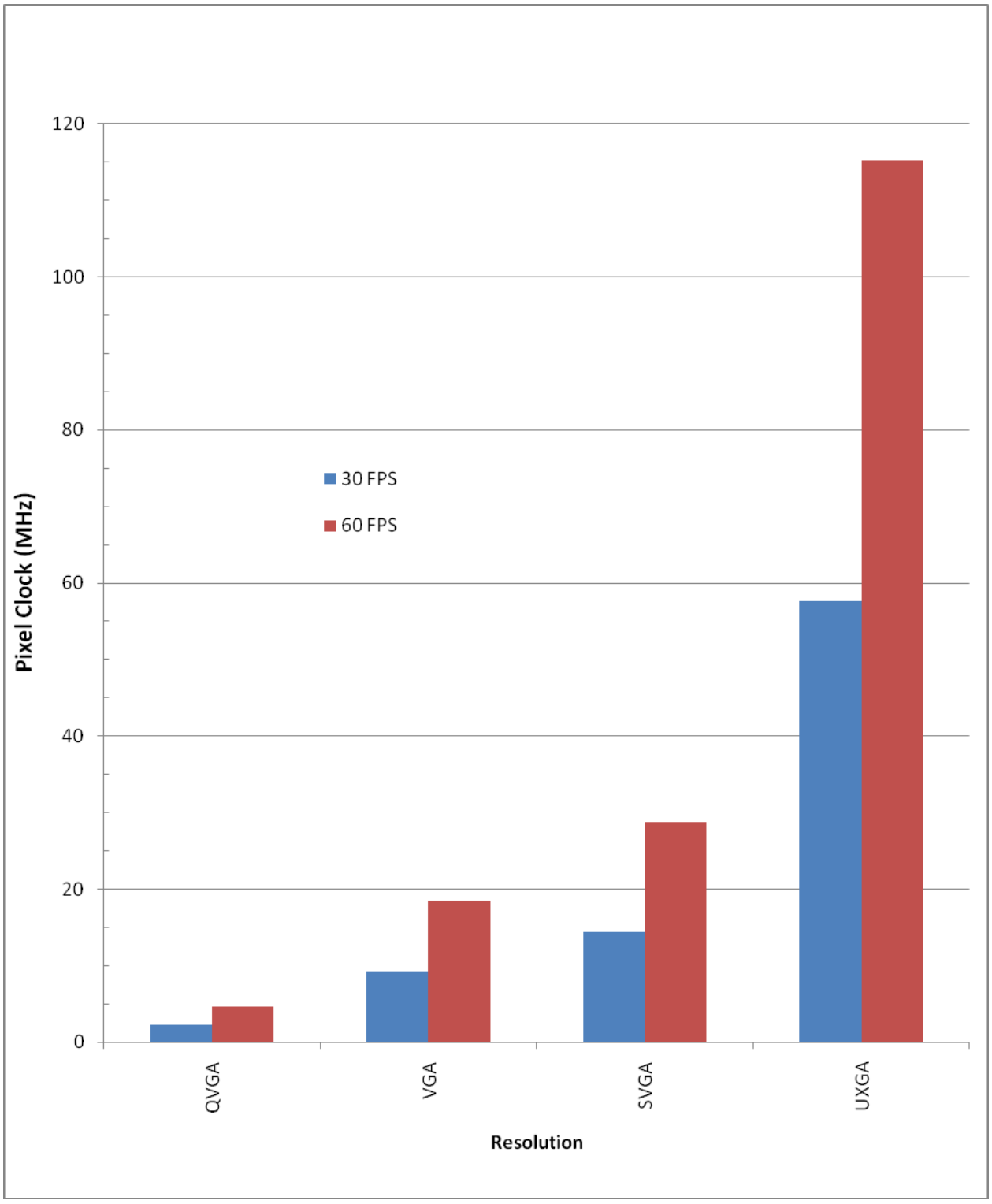

Figure 5-12 – Clock frequencies of the resolutions selected for simulation

5.4.1.1 Results

The simulations of all the defined resolutions and frame rates were performed for each of the designs presented in Section 4.1. The simulations were performed for three different pixel depths of 8, 12, and 16 bits. Figure 5-13 shows total power results for the full range of pixel depths, resolutions, frame rates, and designs. The designs are separated by color and the pixels depths are separated by marker shape. The results are graphed as power dissipation versus data bandwidth. As a result, there are alignments of some simulation bandwidth values that have different clock rates. This helps highlight the difference between a wider pixel and a faster clock rate when evaluating the power with respect to bandwidth. Please be aware that all result diagrams use logarithm based 2 scales on both the X and Y axis because of the large dynamic range of both variables; this unfortunately leads to linear trend lines appearing curved. The axis dimensions are held constant between figures to facilitate comparisons.

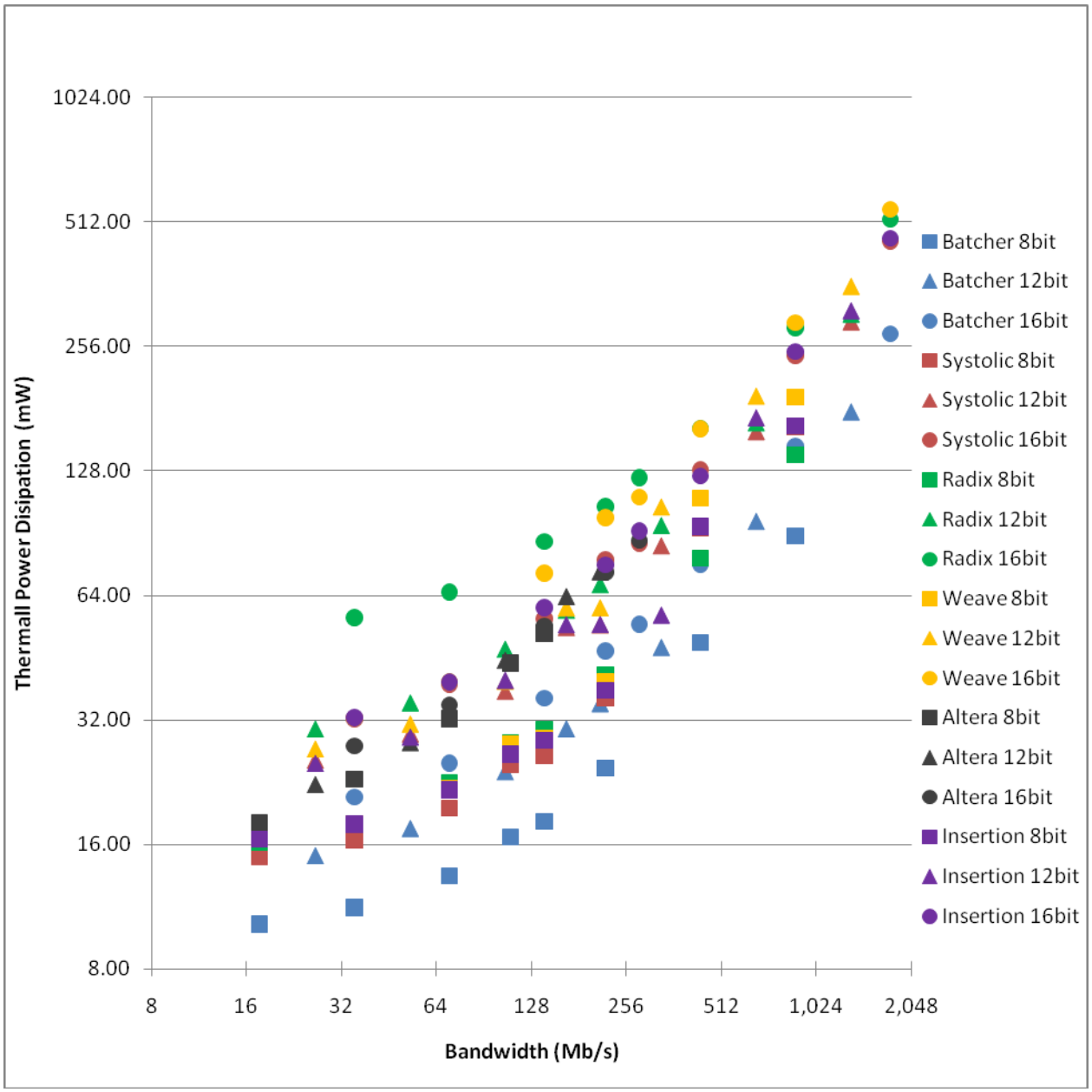

Figure 5-13 – Total power consumption for all sorters, resolutions and bit depths

To reduce the complexity of Figure 5-13, the 8, 12, and 16 bit results are separated into separate figures via Figure 5-14, Figure 5-15, and Figure 5-16 respectively. This allows for a more detailed view of the relationship between designs by holding the pixel depth constant. The three figures also have trend lines for each data set.

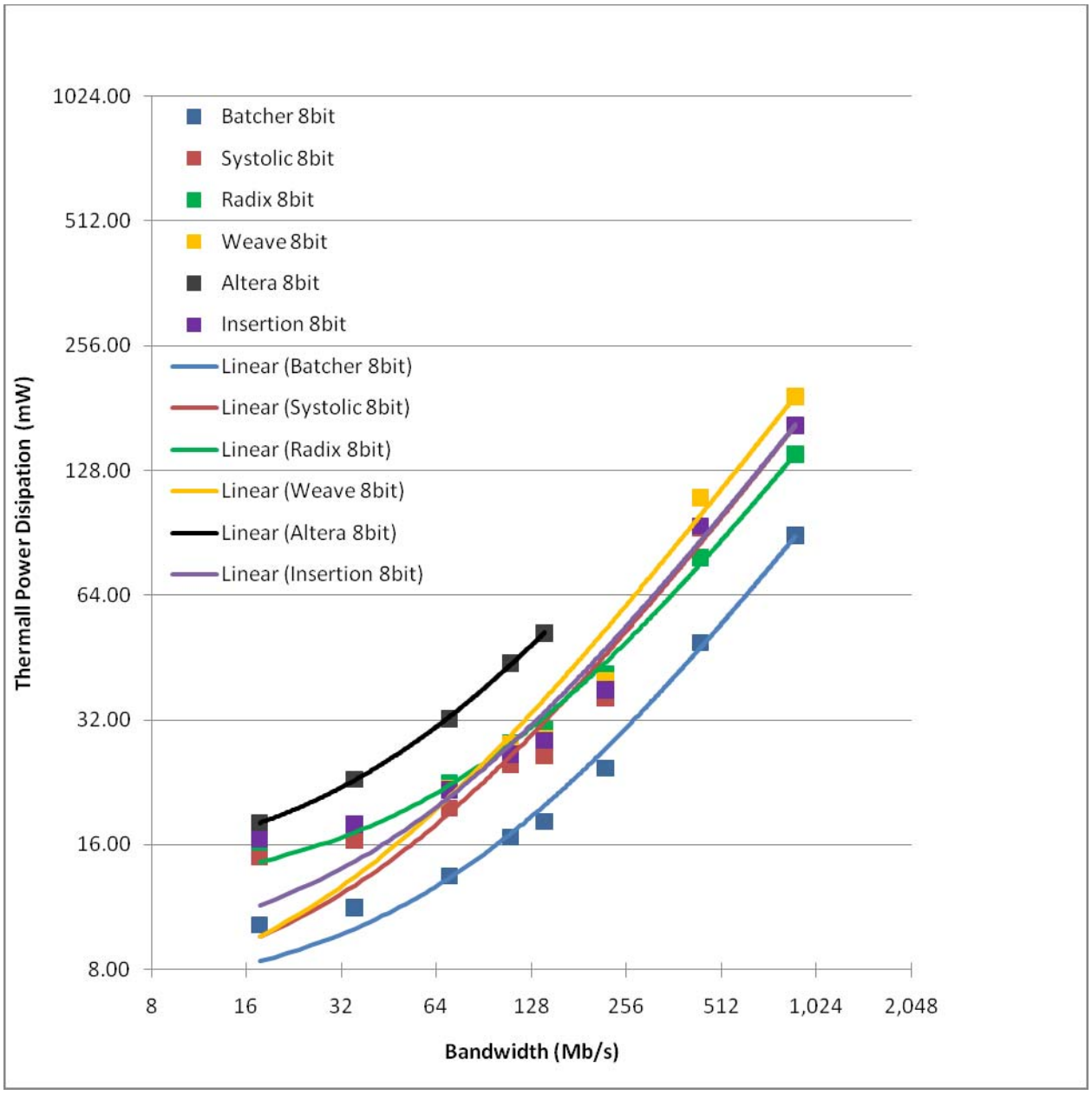

Figure 5-14 – Total power consumption for 8-bit resolutions

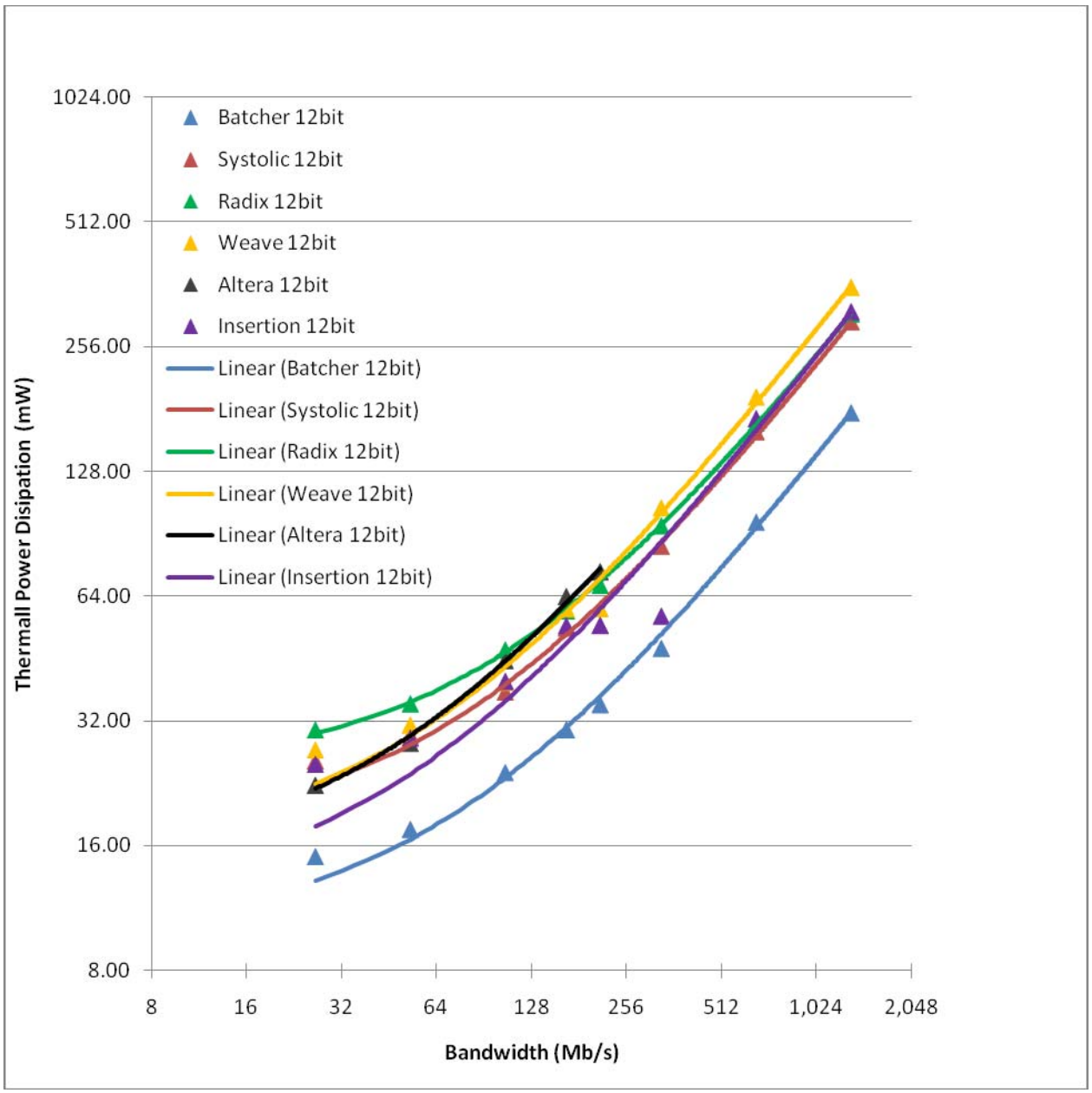

Figure 5-15 – Total power consumption for 12-bit resolutions

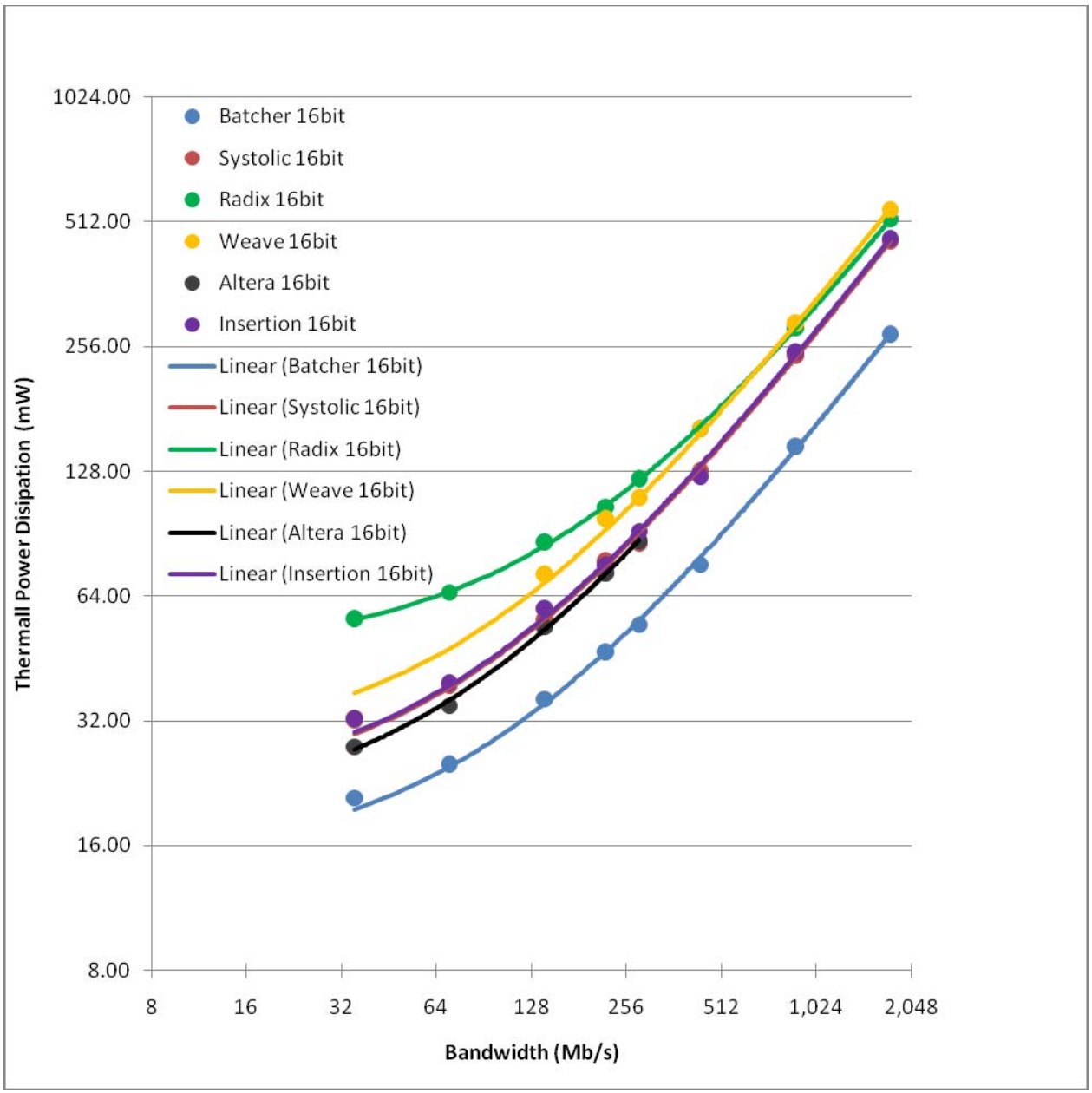

Figure 5-16 – Total power consumption for 16-bit resolutions

5.4.1.2 Analysis

As can be seen from the linear fit trend lines for all designs in Figure 5-14, Figure 5-15, and Figure 5-16, the Batcher design appears to always trend lower than all other designs. It should

also be observed that the radix design grows significantly slower than all but the Batcher design at all pixel depths.

To provide a better understanding of the linear fits for each variant, the slope, intercept, and R squared value are present in Table 5-5. The R squared values show that, for the simulated data points, the linear model is a good fit for each design and pixel depth. While the intercept value is important, the slope dictates the growth rate trend. Figure 5-17, Figure 5-18, and Figure 5-19 present the trend line results for each bit depth extrapolated above and below the test points. For higher bandwidths, the Batcher design is far superior to all other designs.

A close inspection of the Batcher trend line in Figure 5-17 shows that for very low bandwidth processing, below 8 Mb/s, the Weavesort design may consume less power than the Batcher design. However, at low data rates, the trend line isn't fully supported by the data; both at 16 and 32 MB/s, the data for Weavesort and Batcher remain above the trend line. This is the result of discrete nature of BRAM block allocation as explained in Section 5.3.2. Therefore, a more detailed trend line, using more simulations points, would appear as a step function at small image widths and as a continuous (but linear) function at large image widths. The step points are correlated with the BRAM block allocation steps. The linear sections are then correlated to the behavior for varying the clock rate, through resolution changes, within each BRAM step.

Design	Pixel depth	Slope	Intercept	R squared
Batcher	8	0.0935	6.7388	0.9966
Systolic	8	0.1799	6.4280	0.9883
Radix	8	0.1461	11.9820	0.9982
Weave	8	0.2140	5.8367	0.9857
Altera	8	0.2758	13.1950	0.9996
Insertion	8	0.1783	8.3022	0.9889
Batcher	12	0.1273	9.8348	0.9986
Systolic	12	0.2102	16.9310	0.9986
Radix	12	0.2150	24.1680	0.9999
Weave	12	0.2603	15.6700	0.9977
Altera	12	0.2859	14.4080	0.9953
Insertion	12	0.2260	11.8840	0.9839
Batcher	16	0.1482	14.3700	0.9996
Systolic	16	0.2498	20.9740	0.9996
Radix	16	0.2689	46.7450	0.9999
Weave	16	0.2985	26.8820	0.9992
Altera	16	0.2441	18.7600	0.9990
Insertion	16	0.2539	21.1100	0.9994

Table 5-5 – Linear trend line values for resolution variation

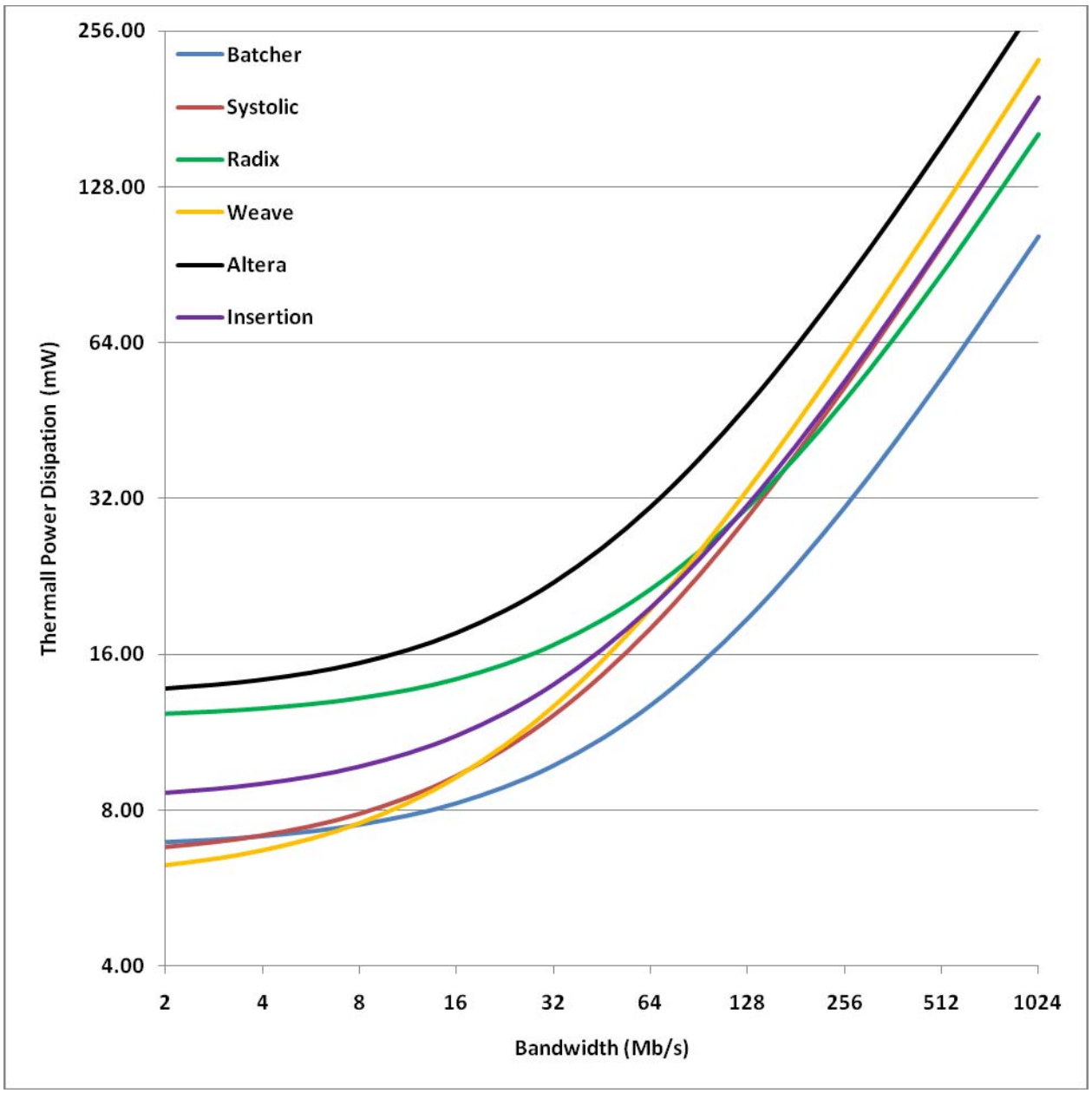

Figure 5-17 – Power consumption trend for 8-bit resolutions

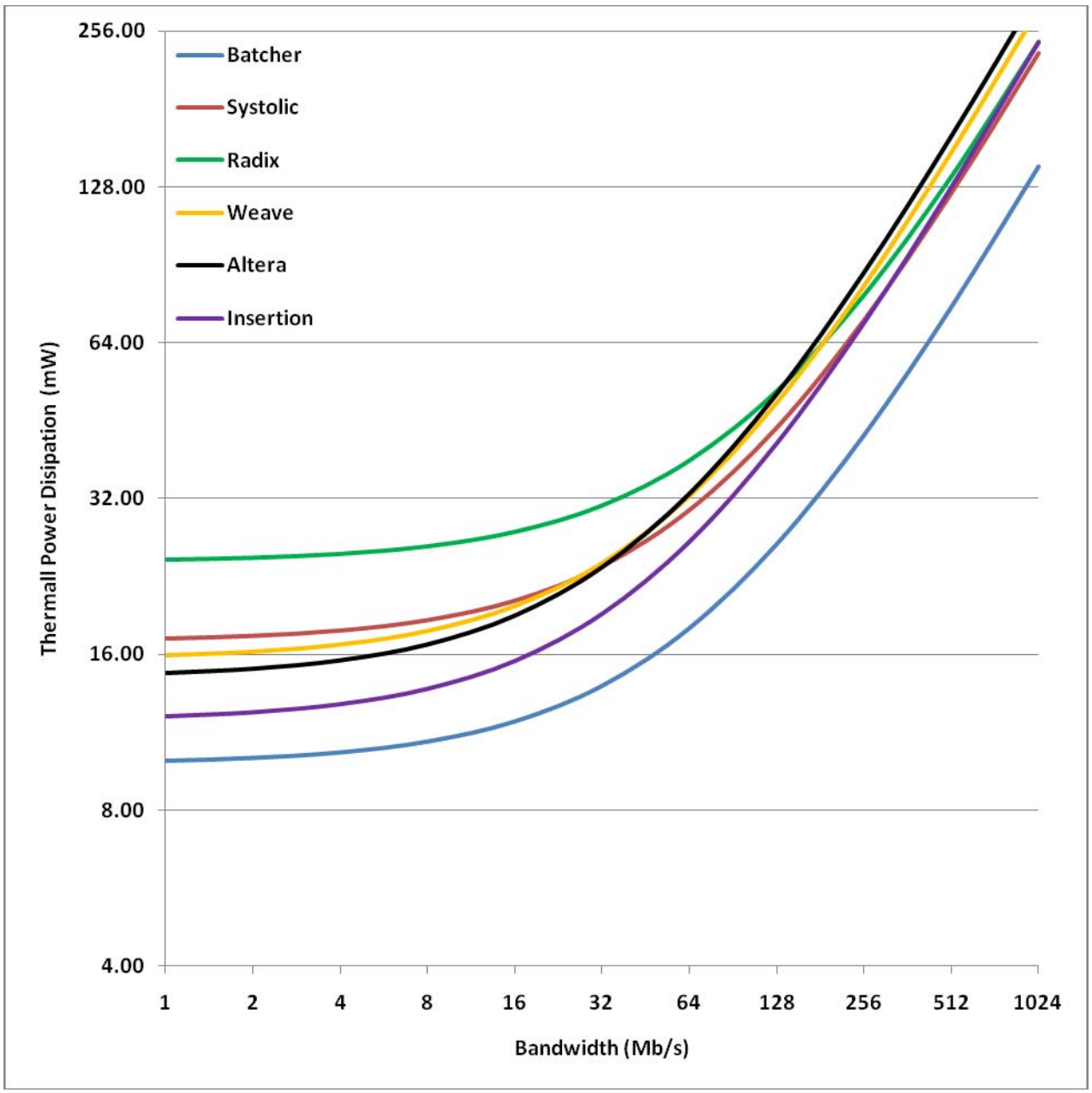

Figure 5-18 – Power consumption trend for 12-bit resolutions

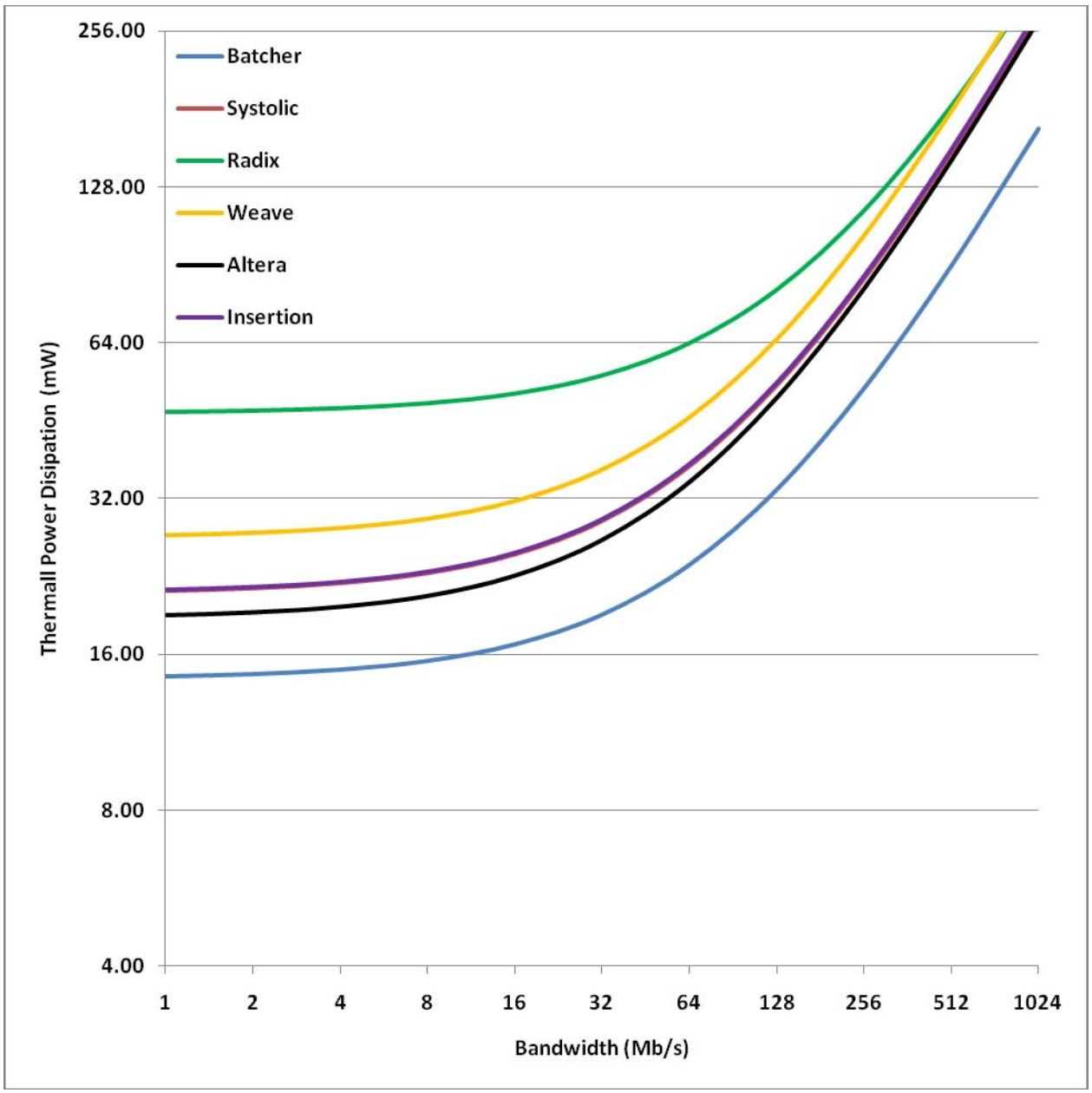

Figure 5-19 – Power consumption trend for 16-bit resolutions

To provide a different point of view for analysis from Figure 5-13 through Figure 5-16, the results for each design are divided into separate figures via Figure 5-20 through Figure 5-25. This separation allows for a more detailed view of the relationship between pixel depths in each design while holding the implementation structure constant. This diagram format also allows the

total power to be broken down into the sub-components to see how the increase in resolution affects the different forms of power dissipation. Examination of the individual designs shows that all have a clear separation between 8, 12, and 16 bit power consumption and have the expected second order power growth, except for the Altera design.

The Altera design appears to dissipate nearly the same power independent of pixel depth, but there is still a small increase - just not as significant as other designs. The reason for this is unknown since the implementation is also unknown. A speculation is that the IP core allocates the maximum pixel depth for every pixel depth and pads the unused bits as need. This would explain the slight reduction in total power that occurs with bit depth.

The remaining designs show that a high pixel depth, low resolution design is more power hungry than a low pixel depth, high resolution design, even if both have the same bandwidth requirements. This active power trend is exactly as expected and is explained in the earlier pixel depth analysis from section 5.3.3.

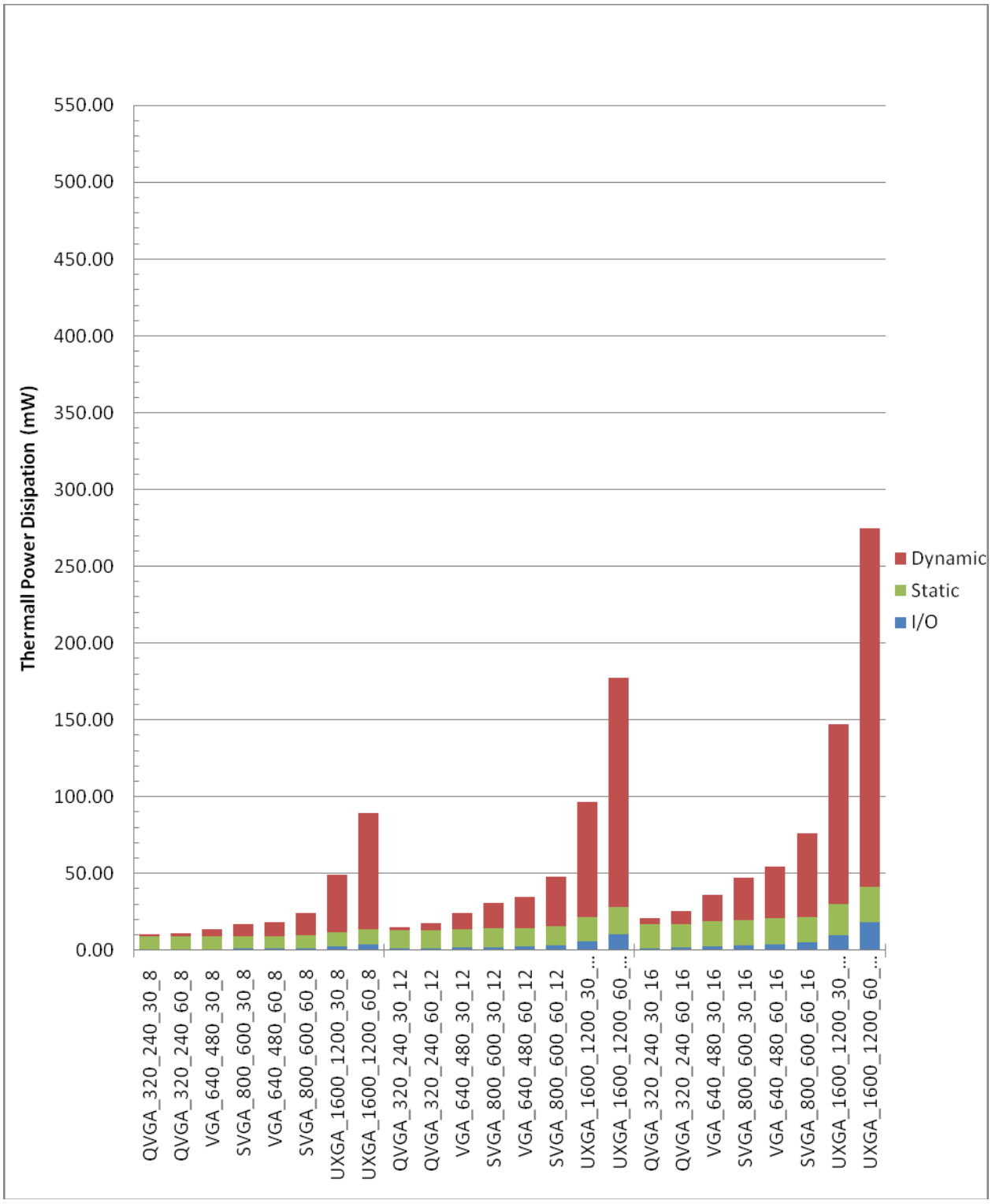

Figure 5-20 – Detailed power consumption for Batcher design

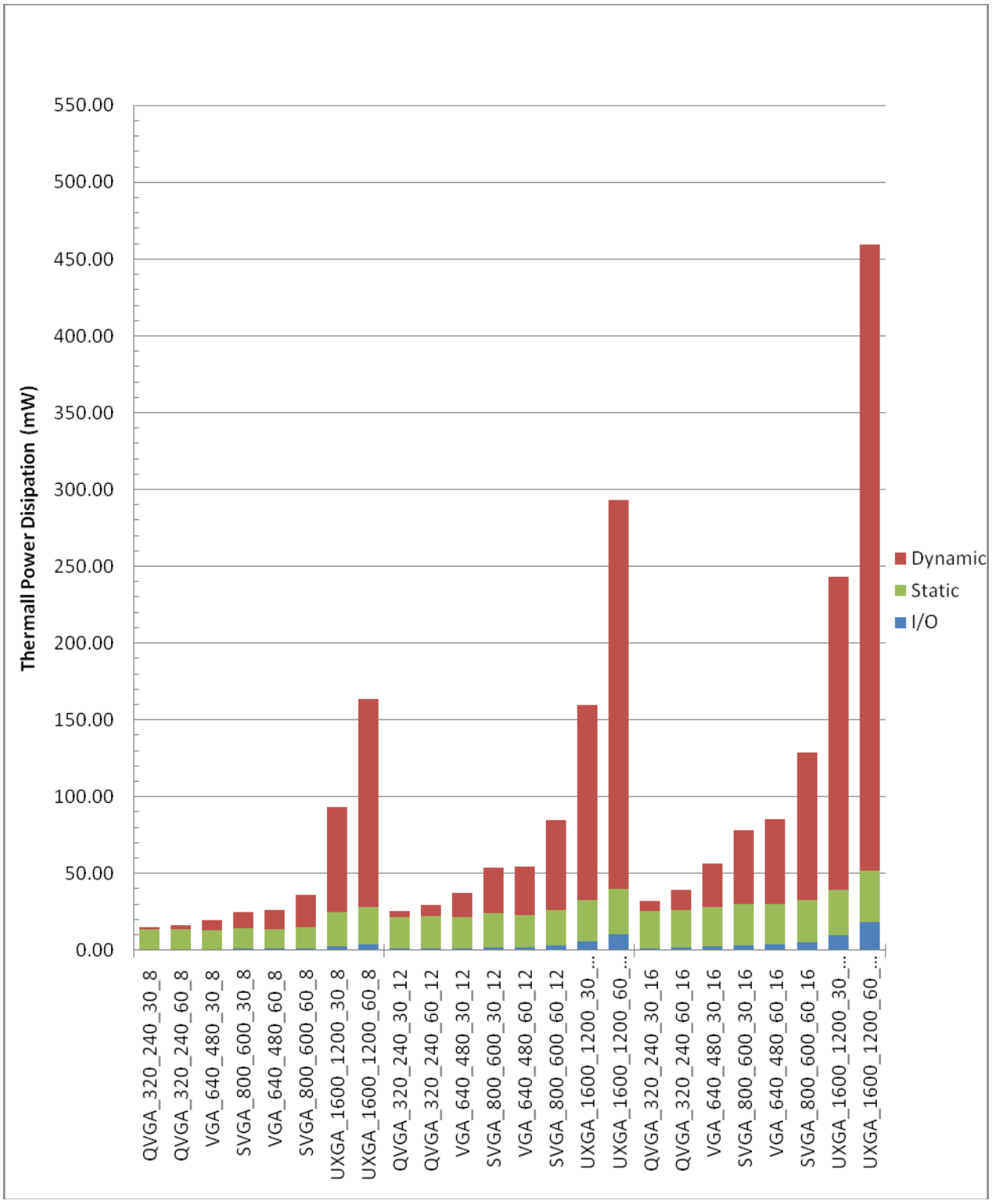

Figure 5-21 – Detailed power consumption for systolic design

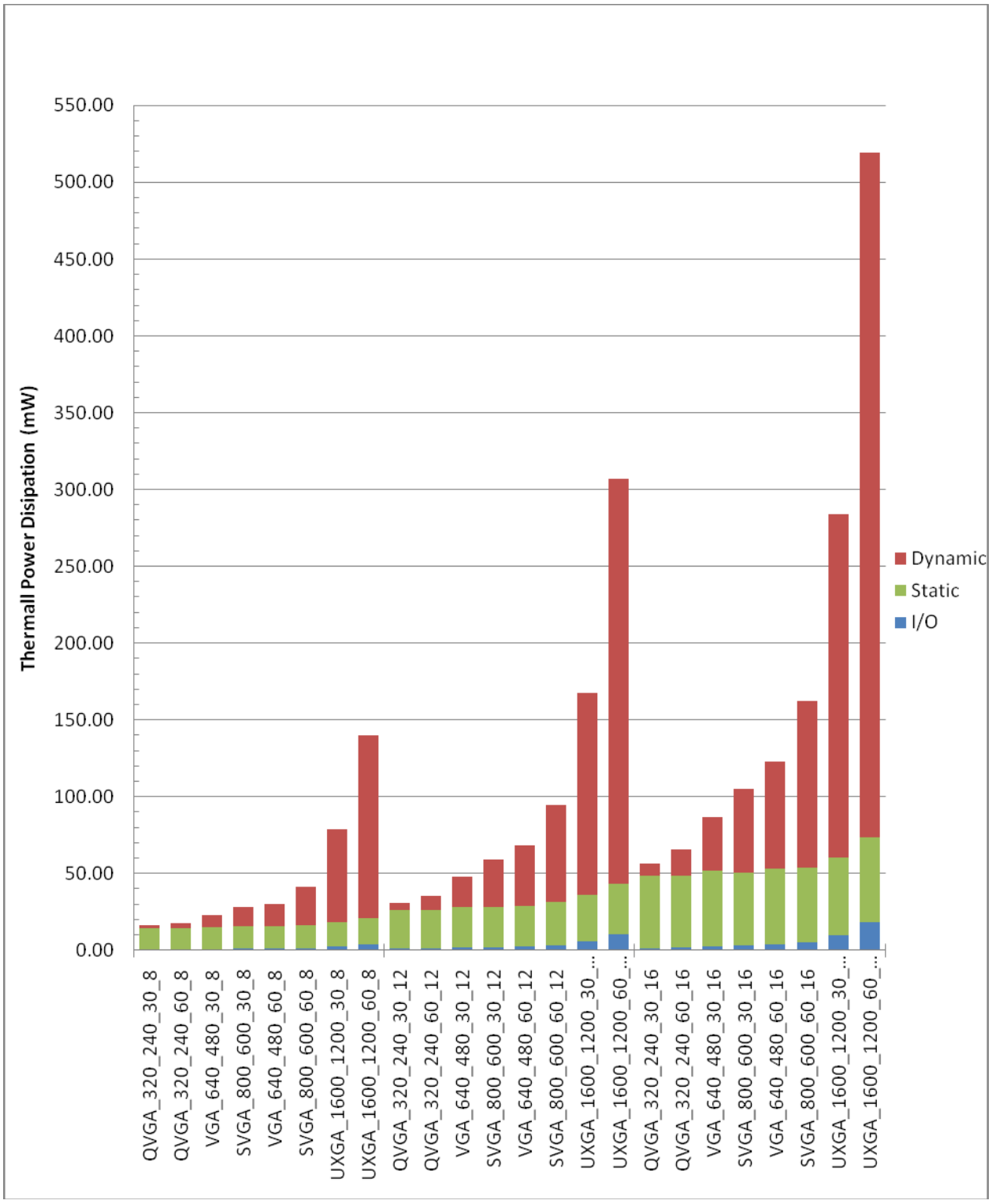

Figure 5-22 – Detailed power consumption for radix design

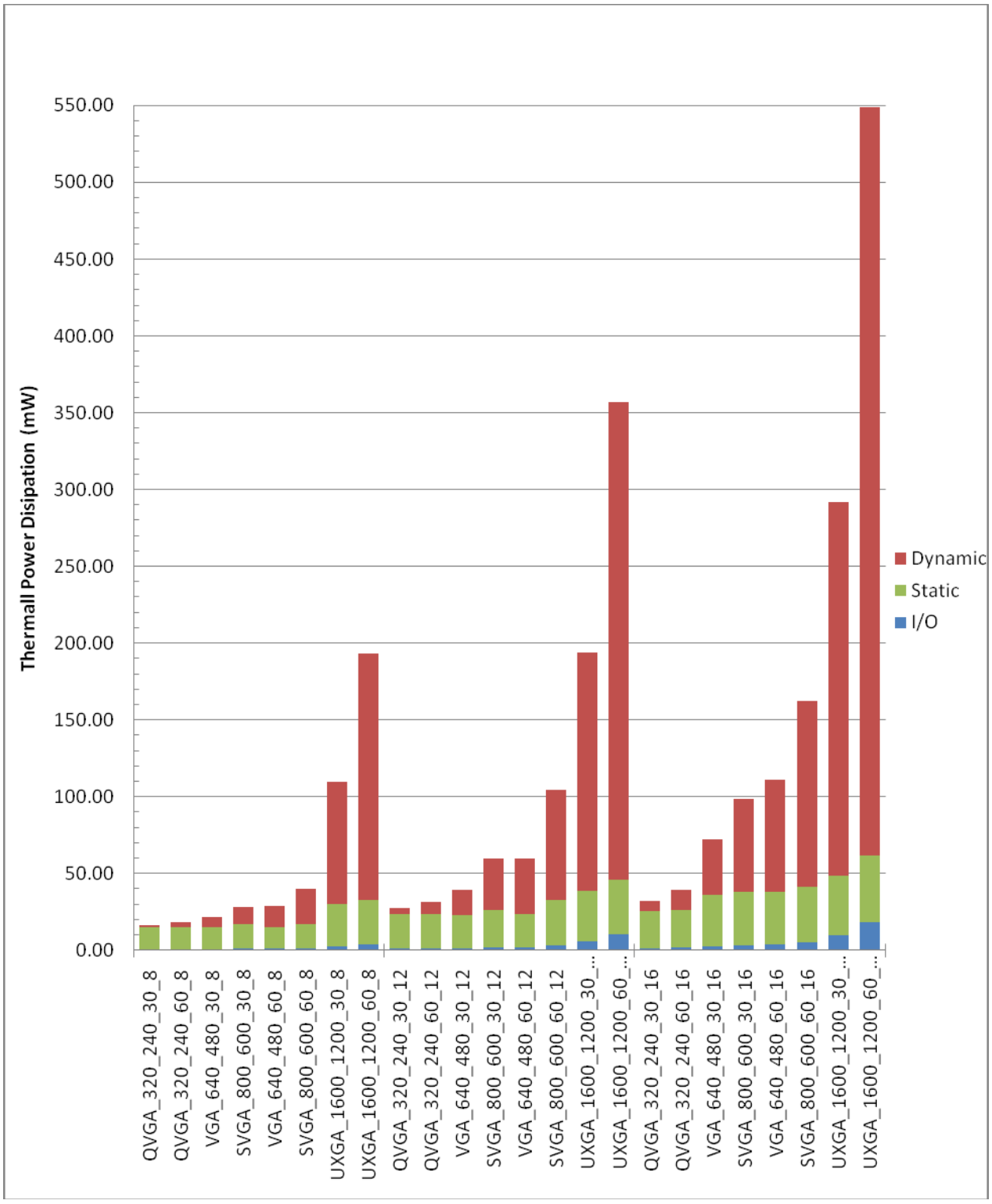

Figure 5-23 – Detailed power consumption for Weavesort design

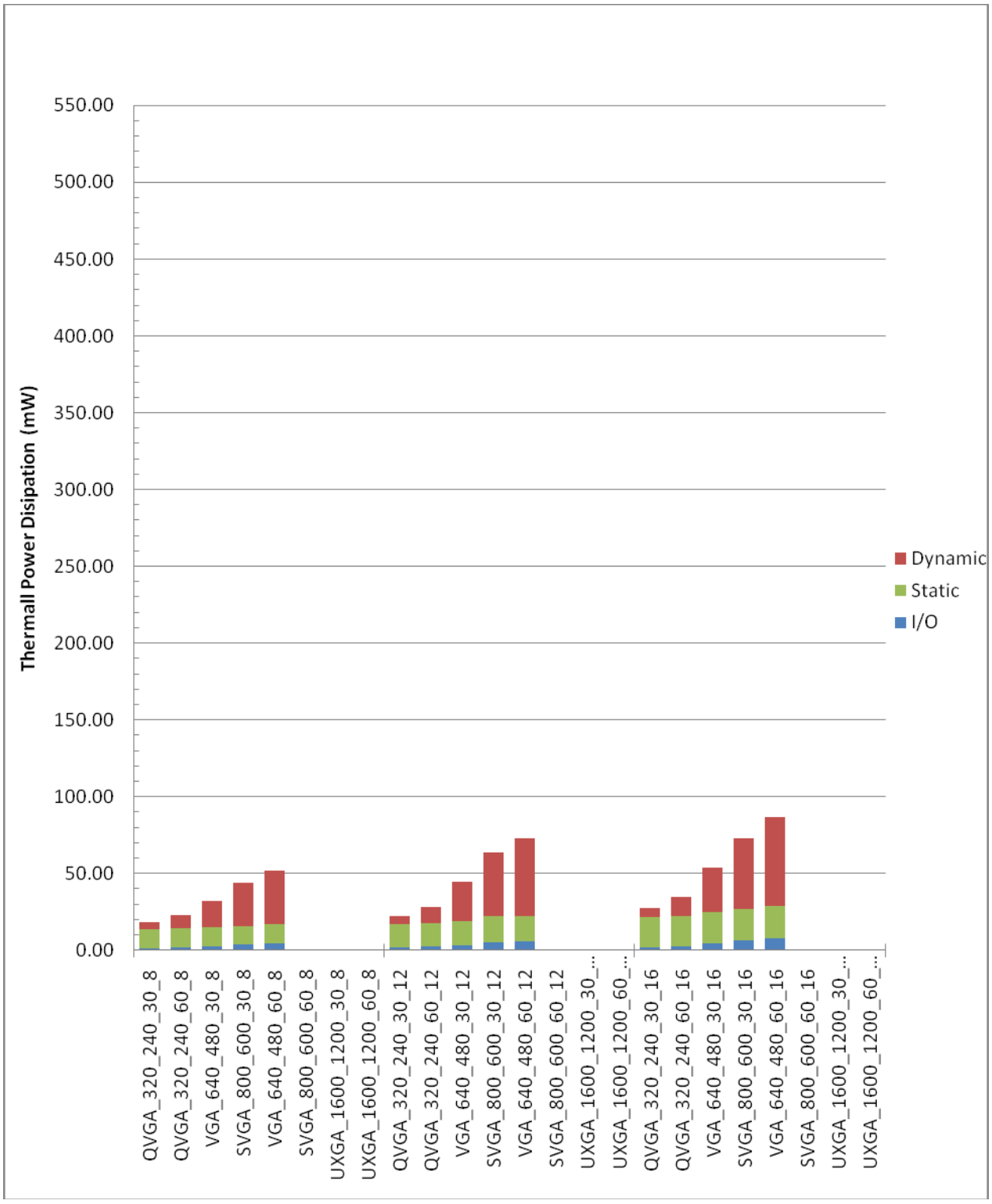

Figure 5-24 – Detailed power consumption for Altera design

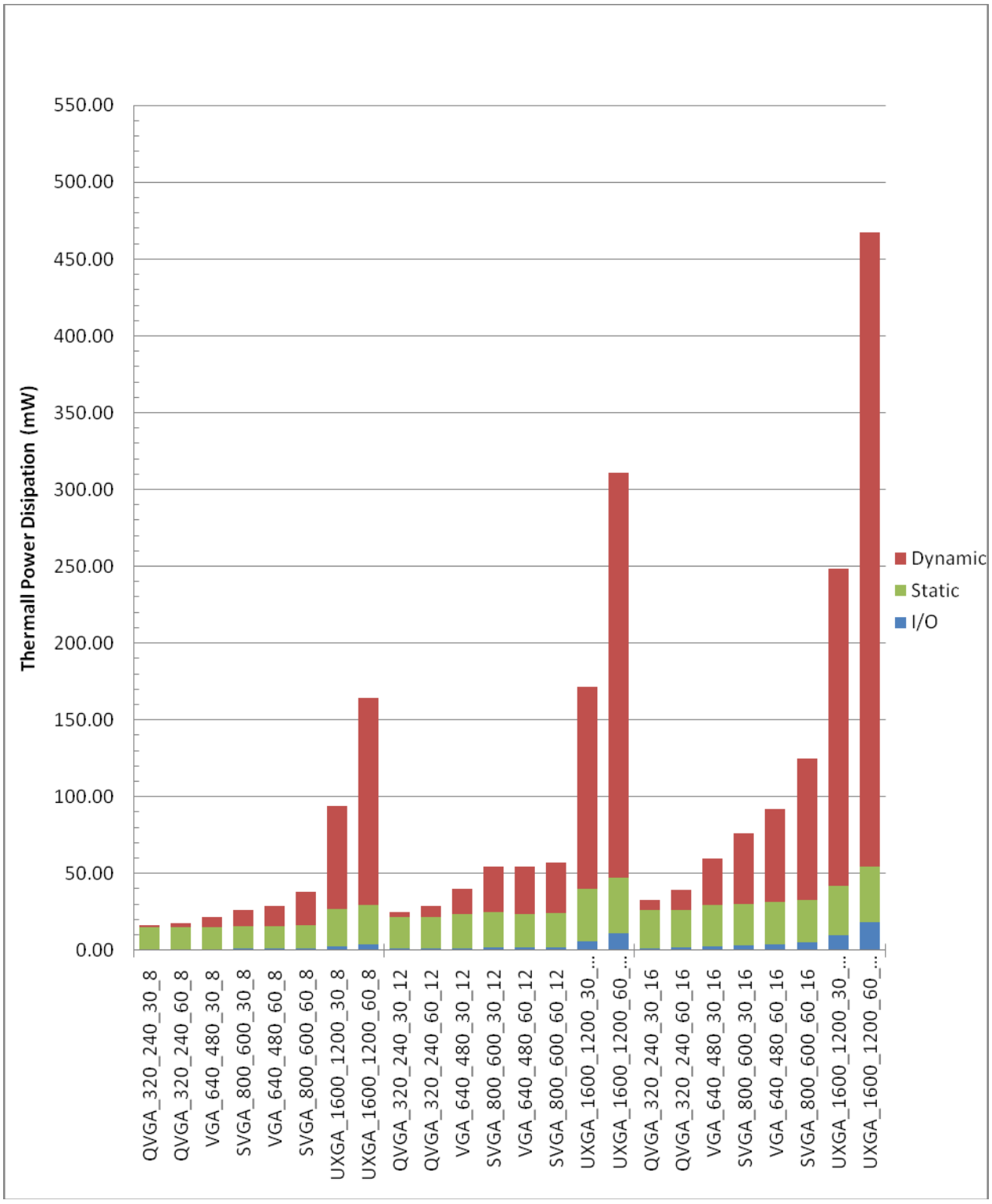

Figure 5-25 – Detailed power consumption for insertion design

5.4.2 Destination system

The following collection of simulations were performed to provide a comparison between different destination systems. The first system is from the Altera Stratix III family of FPGAs, specifically the EP3SL150F1152C4 chip [32]. The second system is from the Xilinx Spartan VI family of FPGAs, specifically the XC6SLX150-3FFG900 chip [33].

5.4.2.1 Results

Figure 5-26 shows the four resolutions from the primary simulations for the Altera platform along with the same resolutions in the Xilinx platform. The Batcher design is used for each resolution with an 8 bit pixel depth at 30 frames per second. The power dissipation has been normalized for each design using the maximum resolution's power, causing a zero to one unit less power scale. The normalization aids in the comparison of the power growth trend between different systems.

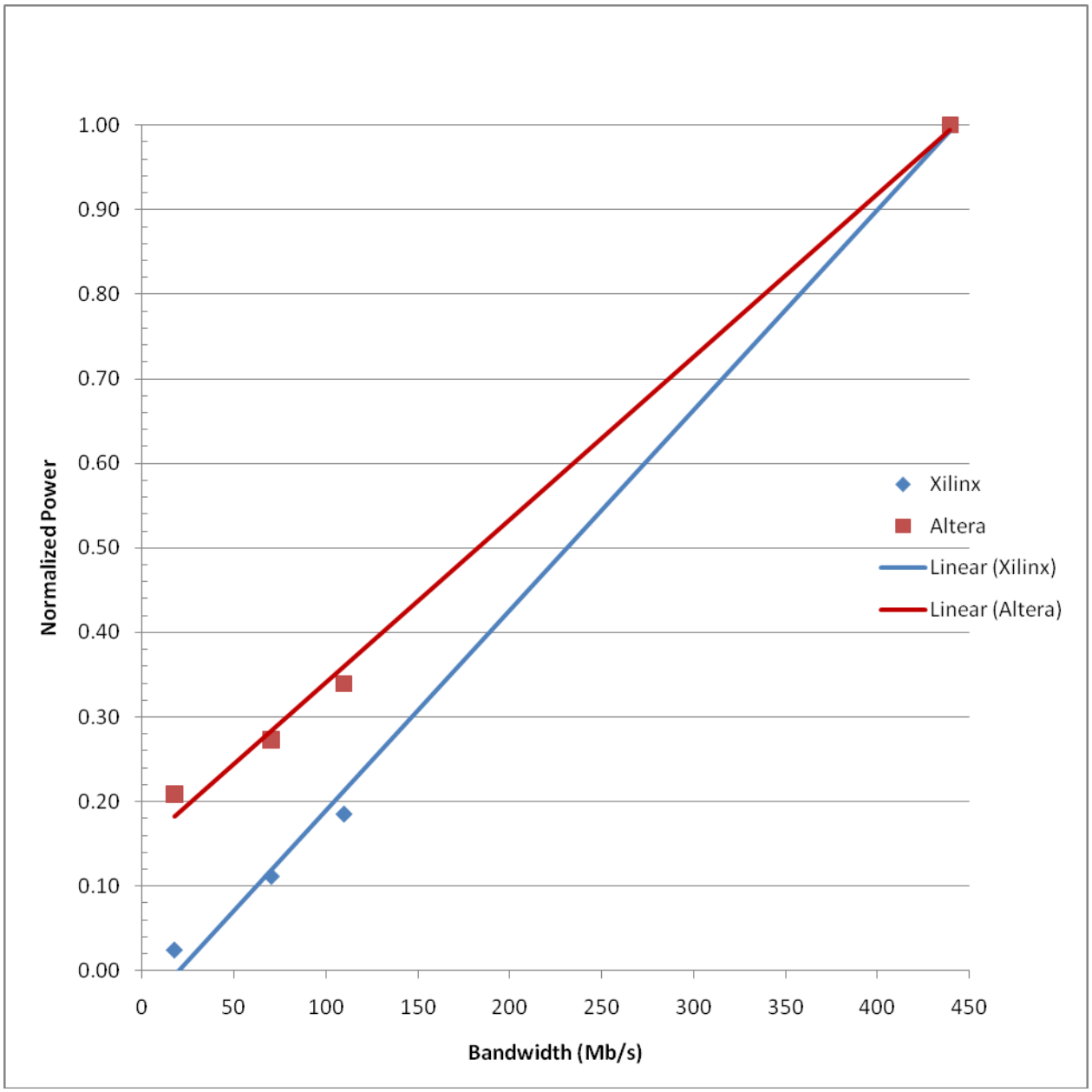

Figure 5-26 – Normalized power consumption for selected resolutions in both platforms

5.4.2.2 Analysis

The goal is to compare power consumption of two identical designs on two completely different systems. While this comparison seems straight forward, the designs utilize different destination specific components resulting in similar but not exactly the same design on each

platform. Functionally and behaviorally, both the local buffer and Batcher sorter are the same on both systems, but the row buffer instantiations are different for each platform caused by differences in each destination systems BRAM interface and structure.

A second difference is the destination systems have no specific relation to one another, other than having sufficient resources to instantiate the designs. Each system's internal organization is significantly different, but the both are built from the fundamental components of flip flops, LUTs, and BRAM bits. Therefore, the relative power consumption on each system should behave relatively the same.

A review of the data presented in Figure 5-26 shows that while the Xilinx simulations consume less normalized total power, each destination system has a linear trend line. Since both platforms have linear progression, the conclusion is that the sorters have common power dissipation behavior on both platforms.

6 CONCLUSIONS

Chapter 6 contains conclusions as well as a summary of this research and results. This chapter will also present the top performers over the variable space under analysis. Post research observations are also included in this chapter.

6.1 Summary of research

Our process of algorithm development via MATLAB and implementation via HDL proved efficient by allowing algorithm evaluation to be performed while abstracted from the algorithm details. The median filter showed significant scintillation noise removal while not drastically affecting the clarity or sharpness of the image.

Our HDL implementations provide deterministic, low power, and low latency operation. The analysis of sorters showed the Batcher sorting network is superior to the other designs implemented. The Batcher implementation of the median filter outperforms the preexisting Altera IP core for imagery with pixel clock rates within the range that was simulated and trends show that it will outperform up to the maximum pixel rate possible. The Batcher design is also predicted to outperform all clock rates greater than the maximum clock rate for all pixel depths analyzed.

The conclusions made from the initial analyses result in the understanding that the row buffer width controls a discrete step power performance that has linear behavior within each discrete step with respect to pixel clock rate. The conclusions also show that power grows with a polynomial rate as pixel depth increase.

6.2 Post research observations

One of our original design assumptions was that a five row buffer would result in a very simple addressing scheme compensating for the additional memory requirements. Our analysis

of the initial comparison to the Altera core caused us to reconsider this assumption and we modified the row buffer implementation. The usage of a minimal four row buffer has a part in making all of our designs superior to the Altera core for any configuration or sorter.

We also found that the speed of our initial implementation was limited not by the sorting algorithm, but rather by the row buffer. Originally, the buffer included a single dual port memory controller accessing the whole buffer. This architecture required the use of a memory clock six times faster than the pixel clock. This memory clock rate limits the design to pixel clock frequencies of around 65 MHz. Because this was inadequate for our current requirements, we divided the buffer into four separate allocated rows via a four tap controller. These improvements allowed the overall design's maximum clock rate to be equal to the BRAM's maximum clock frequency rather than a portion of the FPGA's maximum clock frequency. The four tap design allowed one port on each row buffer to read every clock cycle while the other buffer port was used to write the incoming pixel.

In the course of implementation research, we came across the concept of a separable median presented originally in [3] by Tukey. Several individuals have successfully ported versions of this concept to hardware [4][5][29]. The initial evaluation of 5x5 separable median suggests that its SNR improvement and visual quality are both poorer than the 5x5 median, but may still be acceptable while requiring substantially less resources. The customer is in the process of determining if the reduction in resource requirements for a separable median justifies the degradation of image quality.

REFERENCES

[1] John Maynard Keynes, “A Treatise on Probability,” Part 2, Chapter 17, Section 5, page 201, 1921.

[2] Norbert Wiener, “Extrapolation, Interpolation, and Smoothing of Stationary Time Series,” New York: Wiley. ISBN 0-262-73005-7. 1949.

[3] J. W. Tukey, “The ninther, a technique for low-effort robust (resistant) location in large samples.” In: Contributions to Survey Sampling and Applied Statistics in Honor of H.O. Hartley, edited by H. A. David, pp. 251-258, New York: Academic Press, 1978.

[4] P. M. Narendra, “A separable median filter for image noise smoothing," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 3, pp. 20-29, January 1981.

[5] G. L. Bates and S. Nooshabadi, "FPGA implementation of a median filter,” Proceedings of IEEE Region 10 Annual Conference on Speech and Image Technologies for Computing and Telecommunications, vol. 2, pp. 437-440, December 1997.

[6] I. D. Scherson and S. Sen, "Parallel sorting in two-dimensional VLSI models of computation," IEEE Transactions on Computers, vol. 38, no. 2, pp. 238-249, February 1989.

[7] D. E. Knuth, The Art of Computer Programming: Sorting and Searching. Reading, Massachusetts: Addison Wesley, vol. 3, April 1997.

[8] E. Ataman, V. Aatre, and K. Wong, "A fast method for real-time median filtering," IEEE Transactions on Acoustics, Speech and Signal Processing, vol.28, no. 4, pp. 415-421, August 1980.

[9] P.E. Danielsson, "Getting the Median Faster," Computer Graphics and Image Processing, vol.17, no.1, pp. 71-78, September 1981.

[10] V.B. Rao and K. Rao, "A new algorithm for real-time median filtering," IEEE Transactions on Acoustics, Speech and Signal Processing, vol.34, no.6, pp. 1674-1675, December 1986.

[11] K. Benkrid, D. Crookes, and A. Benkrid, "Design and implementation of a novel algorithm for general purpose median filtering on FPGAs," IEEE International Symposium on Circuits and Systems, vol. 4, pp. 425-428, 2002.

[12] T. Huang, G. Yang, and G. Tang, "A fast two-dimensional median filtering algorithm," IEEE Transactions on Acoustics, Speech and Signal Processing, vol. 27, no.1, pp. 13-18, February 1979.

[13] Y. Zhao and G. Taubin, "Real-Time Median Filtering for Embedded Smart Cameras," IEEE International Conference on Computer Vision Systems, pp. 55-55, January 2006.

[14] B. Weiss, “Fast median and bilateral filtering,” ACM Transactions on Graphics, vol. 25, no. 3, pp. 519-526, July 2006.

[15] S. Perreault and P. Hebert, "Median Filtering in Constant Time," IEEE Transactions on Image Processing, vol. 16, no. 9, pp. 2389-2394, September 2007.

[16] C. D. Thompson, "The VLSI Complexity of Sorting," IEEE Transactions on Computers, vol. C-32, no. 12, pp.1171-1184, December 1983.

[17] H. T. Kung, “Why systolic architectures?,” In Advanced Computer Architecture, D. P. Agrawal, Ed. Los Alamitos, CA: IEEE Computer Society Press, 1986, pp. 300-309.

[18] M. Karaman, L. Onural, and A. Atalar, "Design and implementation of a general-purpose median filter unit in CMOS VLSI," IEEE Journal of Solid-State Circuits, vol. 25, no. 2, pp. 505-513, April 1990.

[19] M. Vega-Rodríguez, J. Sánchez-Pérez, and J. Gómez-Pulido, “An FPGA-based implementation for median filter meeting the real-time requirements of automated visual inspection systems,” Proceedings of the 10th Mediterranean Conference on Control and Automation, August 2007.

[20] K. E. Batcher, “Sorting Networks and their Applications,” Spring Joint Computer Conference of American Federation of Information Processing Societies, vol. 32, pp. 307-314, 1968.

[21] H. S. Stone, "Parallel Processing with the Perfect Shuffle," IEEE Transactions on Computers, vol. C-20, no. 2, pp. 153-161, February 1971.

[22] A. Huang, “STARLITE: A Wideband Digital Switch,” IEEE Global Telecommunications Conference, Communications in the Information Age, 1984.

[23] K. Chung, “A fast pipelined median filter network,” Signal Process, vol. 51, no. 2, pp. 133-136, June 1996.

[24] W. P. Goodwin and S. K. Das, “Implementing parallel sorting algorithms on a linear array of transputers,” In Proceedings of the 1989 ACM/IEEE Conference on

Supercomputing, Reno Nevada, November 12 - 17, 1989. Supercomputing '89. ACM, New York, NY, 789-796.

[25] J. S. Kim and H. W. Park, "Adaptive 3-D median filtering for restoration of an image sequence corrupted by impulse noise," Signal Processing: Image Communication, vol. 16, no. 7, pp. 657-668, 2001.

[26] J. Scott, M. Pusateri, M. U. Mushtaq, "Comparison of 2D median filter hardware implementations for real-time stereo video," 37th IEEE Applied Imagery Pattern Recognition Workshop, 2008, AIPR '08, pp.1-6, 15-17 Oct. 2008.

[27] A. Mukhopadhyay, “WEAVESORT - A New Sorting Algorithm for VLSI,” University of Central Florida, Dept. of Computer Science. Report No. TR-53-81, March 11, 1981.

[28] A. Mukherjee, N. Motgi, J. Becker, A. Friebe, C. Habermann, M. Glesner, "Prototyping of Efficient Hardware Algorithms for Data Compression in Future Communication Systems," 12th IEEE International Workshop on Rapid System Prototyping, p. 0058, RSP '01, 2001.

[29] D. Cline, K. B. White, and P. K. Egbert, "Fast 8-Bit Median Filtering Based on Separability," IEEE International Conference on Image Processing, vol.5, pp. 281-284, September 2007.

[30] V. Aebi and J. Boyle, "Electron bombarded active pixel sensor," U. S. Patent 6,285,018, July 20, 1999.

[31] “NightVista® M611 Low Light Level Camera,” March 2010. [Online]. Available: http://www.intevac.com/files/M611Datasheet.pdf. [Accessed: July. 26, 2010].

[32] “Stratix III Device Handbook, Volume 1,” July 2010. [Online]. Available: http://www.altera.com/literature/hb/stx3/stx3_siii5v1.pdf. [Accessed: July 29, 2010].

[33] “Spartan-6 FPGA Data Sheet: DC and Switching Characteristics,” July 2010. [Online]. Available: http://www.xilinx.com/support/documentation/data_sheets/ds162.pdf. [Accessed: July 29, 2010].