

# Memory-Aware Optimization of FPGA-Based Space Systems

Nicholas Wulf, Alan D. George, and Ann Gordon-Ross

NSF Center for High-Performance Reconfigurable Computing (CHREC)

Department of Electrical and Computer Engineering, University of Florida, Gainesville, FL 32611  
{wulf, george, ann}@chrec.org

**Abstract**—Designing FPGA-based space systems that meet mission goals of high performance, low power, and high dependability is challenging. Our previous work established a framework to help designers of FPGA-based space systems to consider a wide range of designs, evaluate the power and dependability of those designs, and narrow the large design search space down to a significantly reduced Pareto-optimal set. To further improve and extend our framework’s ability to evaluate and optimize increasingly complex aerospace systems, this paper details our framework’s memory extension, which enables memory-aware analysis by refinements to our framework’s original analysis. The memory-aware analysis more accurately predicts a system’s power and dependability by modeling three memory resources: internal-memory capacity, internal-memory bandwidth, and external-memory bandwidth. We demonstrate the importance of our framework’s memory extension by investigating a case study based on an enhanced version of a hyperspectral-imaging satellite mission. After analyzing 22 unique Virtex FPGA devices and optimizing each for power and then dependability, the framework selects four Pareto-optimal designs, ranging from very-low power to high dependability. Results of the framework’s memory extension show that memory resources may limit the performance of an FPGA-based space-system design and contribute significantly towards power and dependability results.

## TABLE OF CONTENTS

1. INTRODUCTION .....	1
2. BACKGROUND .....	2
3. MEMORY-AWARE ANALYSIS .....	3
4. CASE STUDY .....	6
5. RESULTS AND ANALYSIS .....	8
6. CONCLUSIONS .....	11
ACKNOWLEDGMENTS .....	11
REFERENCES .....	11
BIOGRAPHY .....	13

## 1. INTRODUCTION

Demand is increasing for high-performance space computing systems as a result of limited space-to-Earth (downlink) bandwidth and increasingly capable sensors capturing enormous amounts of data. Traditional computing systems are unable to meet mission goals of high performance, low power, and high dependability in the harsh space environment. Unlike traditional software-programmable microprocessor-based processing devices, system designers can modify a field-programmable gate array’s (FPGA) device configuration to specifically target a mission’s

application, thereby increasing performance and reducing power. However, although FPGA devices typically consume less power than traditional processing devices, FPGAs are particularly susceptible to upsets caused by radiation, possibly requiring fault-tolerant (FT) strategies to mitigate these upsets at the cost of increased power or decreased performance.

Another challenge is determining the most appropriate device configuration, which dictates the type and quantity of FPGA resources (e.g., flip-flops (FF), look-up tables (LUT), digital-signal processing (DSP) units, block RAMs (BRAM), and input/output (I/O) pins) that the system requires. FPGA devices offer multiple ways to define the same operations using different FPGA resources (e.g., multiply operations with and without using DSP units), so the designer’s choice of device configuration may affect the system’s performance, power, and/or dependability. Therefore, FPGA-based space-system designers must find the appropriate combination of FPGA device, FT strategy, and device configuration to meet the mission’s goals. Given the large design space afforded by these numerous design options, FPGA-based space-system design is a daunting, laborious task without reliable automated design tool assistance.

Another design option that further complicates FPGA-based space-system design involves choosing a sufficient amount of memory and allocated memory bandwidth (i.e., rate at which data transfers within the system) considering mission-specific data capture rates and downlink bandwidth. An FPGA has three standard memory resources that affect system performance: internal-memory capacity (IMC), internal-memory bandwidth (IMB), and external-memory bandwidth (EMB). IMC represents an FPGA’s on-chip memory storage (i.e., BRAMs). IMB is the on-chip transfer rate of the data between the IMC and the FPGA’s processing operations. EMB is the external transfer of data between the IMC and an off-chip external-memory device through an on-chip external-memory port.

Considering these memory resources is an important and non-trivial task for FPGA-based space-system designers. Insufficient IMB or EMB may bottleneck performance since operations cannot quickly process data if there is deficient IMB to move data quickly through the device or EMB to move data quickly into/out of the device (which is even common in high-performance terrestrial computing [1]). Increasing the number of external-memory ports can increase the EMB, but each port requires FFs, LUTs and I/O pins. This extra resource usage not only increases power but also

decreases dependability since FPGA resources are only vulnerable when in use. Conversely, if these resources were already being used by operations, reducing the operation count to add external-memory ports can negatively impact performance. Alternatively, storing large amounts of data in IMC to avoid high EMB may decrease dependability as well, since the IMC is typically less dependable than the external-memory devices, which may be hardened against radiation and/or protect data with error-correcting codes. Verifying that an FPGA-based space system has sufficient IMB and EMB and evaluating the design option between high IMC usage and high EMB adds an extra burden on system designers already struggling to meet the mission's goals, further necessitating the use of a comprehensive automated design tool.

To address some of these design challenges, in prior work [2] [3] we proposed an FPGA-based space-system design framework as a tool to aid system designers in pruning the large design space based on a subset of the design options. Our framework iterated through a set of FT strategies and an internal device database (which stores previously generated data on FPGA devices and the available operations for those devices) to analyze hundreds of designs with respect to device choice and FT strategy while considering the designer-defined mission environment (e.g., particular orbit around Earth) and application (i.e., precision and types of operations). Our framework's analysis predicted the designs' power and dependability and discarded any designs that did not meet the mission's goals. Finally the framework used the power and dependability results to determine the Pareto-optimal set of designs, which was much smaller than the original design search space and highlighted the tradeoffs in power and dependability across the range of Pareto-optimal designs. Although our framework proved to be a useful design optimization tool for FPGA-based space systems, our framework did not consider IMC, IMB, or EMB since the framework lacked the ability to handle BRAMs, external-memory ports, and external-memory devices.

To further improve and extend our framework's ability to evaluate and optimize increasingly complex aerospace systems, this paper details our framework's memory extension, which enables memory-aware analysis through refinements to our framework's original analysis. Memory-aware analysis more accurately predicts a system's power and dependability by modeling the IMC, IMB, and EMB memory resources. Due to the non-trivial and application-dependent nature of the EMB/IMC relationship, the designer must specify this relationship in addition to the application before the framework can begin analysis. The memory extension expands our framework's analysis to a more complete and accurate holistic design view by not neglecting the importance of memory in FPGA-based space-system design.

We demonstrate the importance of our framework's memory extension by investigating a case study based on an enhanced version of a hyperspectral-imaging (HSI) satellite mission.

Using the HSI case study allows us to compare our new results against our original results presented in [2], which used the same case study. We use the case study to show the method a designer might use to analyze their application and determine their application's EMB/IMC relationship. Results of the framework's memory extension show that memory resources may limit the performance of an FPGA-based space-system design and contribute significantly towards power and dependability results.

The remainder of this paper is organized as follows. Section 2 discusses the background and related work that provides the foundation of our framework's memory extension. In Section 3, we discuss our approach in modifying our framework's analysis to consider the EMB/IMC relationship and perform memory-aware analysis. Section 4 presents an overview of the HSI case study and demonstrates how a designer can analyze the three memory resources to determine an application's EMB/IMC relationship. In Section 5, we show the results of three experiments to demonstrate the use and effectiveness of the framework's memory extension. Finally, in Section 6, we discuss our conclusions and suggest a course for future research.

## 2. BACKGROUND

Our framework leverages prior work by Williams et al. [4], which established the computational density (CD) metric. CD measures a device's computational performance and is measured in operations per second (typically in giga operations per second (GOPS)). A device's CD depends on the precision and type of operations being analyzed (e.g., 8-bit add or double floating-point multiply). CD is useful for comparing performance between a wide range of processing devices including CPUs, DSPs, FPGAs, and GPUs. To compute an FPGA's CD, Williams et al. instantiated a single operation of each type on the FPGA to determine the resources consumed per operation type. A linear programming (LP) method used this data to project how to optimally use the FPGA's device resources to fit the most simultaneous operations on the FPGA. Williams et al. then calculated the CD as the maximum number of simultaneous operations that can fit on the FPGA multiplied by the operation with the lowest operating frequency.

In addition to performance-based metrics, Richardson et al. [5] established memory-based IMB and EMB device metrics. IMB measures the rate at which data can be transferred from on-chip memories to the operations. IMB is important because IMB may become a bottleneck, limiting the speed at which the data can be processed. For an FPGA, IMB represents the bandwidth between on-chip BRAM and the on-chip processing resources. EMB measures the amount of bandwidth between a processing device and off-chip external memory devices. Richardson et al. measured an FPGA's EMB by instantiating an external-memory port on the FPGA for each external memory device and measuring the external memory port's resource requirements. These results were extrapolated to predict how many external-memory ports

could simultaneously fit on the FPGA and how much EMB those external memory ports could provide.

Our original framework [2] [3] modified the LP method used by Williams et al. [4] to find the optimal device configuration based on an optimization target (either power or dependability) and then calculated both the power and dependability of that device configuration for use in the framework’s Pareto-optimal design selection. Whereas the original LP method (Williams et al.) optimized the device configuration for the maximum performance of a device, the modified LP method (MLP) (our original framework) determined device configurations that matched the required CD specified by the mission’s goals and application. Since LP can only optimize towards a single goal and the MLP method was no longer optimizing toward maximum performance, we augmented the MLP method to optimize for either power or dependability. Once the MLP method found the optimal device configuration (based on the optimization target), the MLP method computed the power and dependability of the device configuration based on the device resource usage. Therefore, for each device and FT strategy combination, the MLP method produced power and dependability results for two device configurations (one optimized for power and the other for dependability) for the framework to consider in the Pareto-optimal analysis. Not only did the MLP method enable our framework’s detailed analysis of device configurations, our refinement to the LP method served as necessary groundwork and foundation for incorporating memory-aware analysis.

Several prior works have shown the effects of various memory resources on performance and the importance of memory-aware analysis. Underwood and Hemmert [6] analyzed vector-dot product, matrix-vector multiply, and matrix multiply implementations on a Virtex-II 6000 FPGA. The authors discussed the importance of IMB for sustaining the performance of the floating point operations, and they predicted that as FPGA floating-point performance increases, the FPGA’s inherently high IMB would enable FPGAs to outperform traditional CPUs, which are more likely to be IMB-limited. Dou et al. [7] also analyzed matrix multiplication on FPGAs, focusing on a particular matrix multiplication algorithm to show that the algorithm’s required EMB was proportional to the inverse square root of the IMC usage. The authors verified this result by implementing the matrix multiplication algorithm on a Virtex-II Pro FPGA and varying the EMB and IMC usage to achieve different performance.

### 3. MEMORY-AWARE ANALYSIS

Our framework’s memory-aware analysis enhances our original framework’s analysis by including the effects of the memory resources on power and dependability. We enable our framework’s memory-aware analysis with two separate extensions: an internal-memory extension and an external-memory extension. First, we introduce a high-level view of our memory-resource concepts and analysis. Then we use

these concepts to show how we add the internal-memory extension by extending our framework’s MLP method. We then show how we add the external-memory extension by embedding the MLP method in a wrapper algorithm.

#### 3-1. Memory-Resource Concepts and Analysis

Figure 1 depicts a high-level memory model of the three memory resources’ interactions. IMC (i.e., BRAM storage) buffers data before the operators process this data or send this data to an external-memory device for storage. IMB is the on-chip data bandwidth between the IMC and the operators. EMB is the off-chip data bandwidth between the IMC and any arbitrary number of off-chip external-memory devices via an equal number of on-chip external-memory ports (Figure 1 shows an example with two external-memory devices).

In addition to representing the number and types of operators, the device configuration now represents the amount of IMC usage, IMB, EMB, and external-memory ports on the FPGA. The LP method must now find design configurations that match or exceed the limits set by mission-specific CD, IMB, and EMB requirements. Calculating the mission’s required IMB is straightforward since IMB depends only on the operators’ inputs and outputs, which the mission’s required CD describes. However, calculating the mission’s required EMB is more complicated, since EMB depends on the required CD, the IMC usage, and the mission’s application.

A cursory analysis of the memory model reveals that the required EMB must inversely correlate with IMC usage, which we can observe by varying the amount of IMC usage. With almost no IMC usage, almost every operator input would require both IMB and EMB, since the IMC could not cache the data for reuse. Conversely, with an infinite amount of IMC, the input data would only require EMB once to transfer the data to IMC, and then each data use by the operators would only require IMB, resulting in a negligibly low EMB requirement. Therefore, for realistic design configurations with IMC usage between these two extremes, the amount of required EMB must be between 0 and the required IMB. Consequently, we can only accurately predict

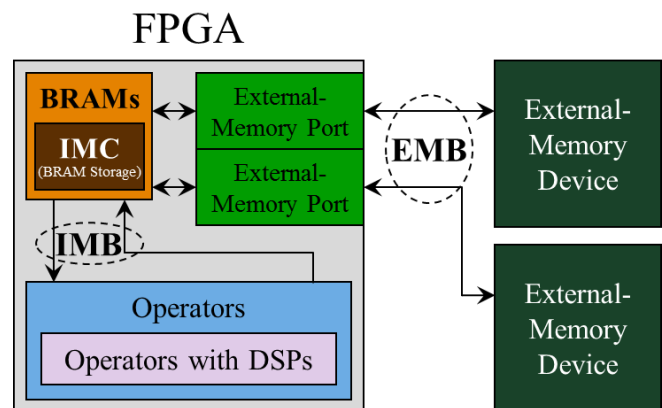


Figure 1 – High-level memory model of memory-resource interactions

the EMB/IMC relationship after we analyze the mission’s application’s memory requirements.

### 3-2. Internal-Memory Extension

The framework’s MLP method finds the optimal device configuration that satisfies the FPGA resources’ constraints. The MLP method defines the optimal device configuration by building an optimizing equation that describes the impact of various operators on the optimization target, which the framework can set to power or dependability. Before the framework can assess the impact of an operator, either the designer or the framework’s device database must specify the properties of the logic resources that compose the operators. We consider three logic resources on a standard FPGA: FFs, LUTs, and DSP units. For power, we use vendor-provided tools [8] [9] to estimate the power per MHz of each logic resource and input these power values into the framework’s device database. The framework measures dependability in terms of errors per day. The designer must specify the error rates for each resource, since the error rates will change significantly based on the mission’s environment. Predicting error rates generally involves combining radiation-test results with radiation-environment models such as CREME96 [10] or SPENVIS [11]. Using the power and error-rate properties of the logic resources, the framework can determine the power and error rate of each operator and build the optimizing equation for the MLP method. The framework must also build a resource-limit equation (RLE) for each logic resource, ensuring the MLP method ignores impossible device configurations that use more logic resources than are available. An RLE defines the resource quantity on the FPGA and the resource consumption for each operator.

Table 1 shows our estimates of the power and error rates for the three logic resources on a Virtex-4, Virtex-5, and Virtex-6, as well as four additional FPGA resources for analyzing IMB, IMC, and EMB. We obtain the power estimations from the vendor-provided Xilinx Power Estimator [8] tool. For the error rates, we use CREME96 to predict the error rate of a configuration bit for each device in the same orbit as the EO-1 satellite from our case study mission. We calculate the error rate of each resource as the number of configuration bits used to program the resource times the error rate of a single configuration bit [12]. This is a worst-case estimation technique that assumes all bits associated with a resource are able to cause an error. In reality only about 10% of the bits in a fully-utilized FPGA are able to cause an error [13], since most of the configuration bits go towards unused routing resources, but this effect is application- and design-dependent. For the BRAM memory bits, which represent a single bit of IMC and not the BRAM structure itself, we use the adjusted error rate of a single configuration bit based on vendor-provided neutron-injection results [14]. Note that we have formed the error-rate estimates purely for the purpose of demonstrating our framework’s analysis in this paper. Although these error-rate estimates are justified, these estimates may or may not reflect more accurate results obtained through beam testing.

**Table 1. Estimated power and error rates for Virtex-4, Virtex-5, and Virtex-6 FPGA resources**

Dev. Fam.	Resource	Power (nW/MHz)	Errors /day
Virtex-4	FF	204	$5.76 \times 10^{-4}$
	LUT	308	$5.76 \times 10^{-4}$
	DSP	30000	$1.76 \times 10^{-2}$
	BRAM (config. bits)	59856	$3.35 \times 10^{-2}$
	BRAM memory bit	N/A	$9.26 \times 10^{-6}$
	Pin (input mode)	712	$6.29 \times 10^{-3}$
	Pin (bidir/out mode)	33818	
Virtex-5	FF	138	$1.58 \times 10^{-4}$
	LUT	175	$1.58 \times 10^{-4}$
	DSP	18000	$4.93 \times 10^{-3}$
	BRAM (config. bits)	71572	$1.06 \times 10^{-2}$
	BRAM memory bit	N/A	$6.50 \times 10^{-6}$
	Pin (input mode)	917	$1.90 \times 10^{-3}$
	Pin (bidir/out mode)	30988	
Virtex-6	FF	111	$1.99 \times 10^{-4}$
	LUT	140	$3.97 \times 10^{-4}$
	DSP	14332	$1.24 \times 10^{-2}$
	BRAM (config. bits)	57258	$2.65 \times 10^{-2}$
	BRAM memory bit	N/A	$2.50 \times 10^{-6}$
	Pin (input mode)	917	$7.77 \times 10^{-3}$
	Pin (bidir/out mode)	33486	

The internal-memory extension modifies the MLP method in several ways to consider the effects of internal memory, producing the IMB-extended MLP (IMLP) method. First, the IMLP method ignores unsustainable device configurations, where the operators require more IMB than is available, by adding an IMB-resource limiting equation (IMB-RLE) to the set of RLEs. To define the IMB resource quantity, the framework calculates the maximum IMB/cycle as:

$$\text{IMB}_{\text{cycle}} = [\# \text{ of BRAMs}] \times \frac{\# \text{ of Ports}}{\text{BRAM}} \times \frac{\text{Usable Bytes}}{\text{Port}} \quad (1)$$

$$\frac{\text{Usable Bytes}}{\text{Port}} = \left\lfloor \frac{\text{Word Width}}{\text{BRAM Port Width}} \right\rfloor \times \frac{\text{Bytes}}{\text{Word}} \quad (2)$$

For example, on the Virtex-5 LX330, only one whole 32-bit word can fit through a BRAM port (36-bit width), so there are four bytes per port. All of the 288 BRAMs on the Virtex-5 LX330 are true-dual port, so the IMB resource quantity is 2,304 bytes/cycle. To complete the IMB-RLE, the framework defines the IMB-resource consumption as the number of bytes from IMB required by each operator per cycle. For our case study application (Section 4), each multiply operator requires two words from the IMB every cycle. Because the outputs of the multiply operators supply the inputs of the add operators, the add operators do not directly consume IMB data. Since there are an equal number of add and multiply operators, the case study application requires an average of four bytes from IMB per operator per cycle. A quick analysis of the IMB-resource quantity and average IMB-resource consumption shows that the Virtex-5 LX330 can only sustain 576 multiply and 576 add operators for the case study

application. Although the IMB-RLE restricts the IMLP method to only considering sustainable device configurations, the IMB-RLE does not enable the IMLP method to optimize for power or dependability according to IMB.

The internal-memory extension also enables the IMLP to consider IMB when optimizing a device configuration by modifying the IMLP's optimization equation. The optimization equation defines the impact of each operator in terms of power or dependability, depending on the optimization target. The internal-memory extension adjusts these impact values based on how much IMB each operator consumes. Furthermore, the calculations for power and dependability assume that using a BRAM at less than maximum capacity will decrease the BRAM's power consumption and error rate accordingly. For power, the framework uses the power per BRAM to calculate the power per byte of IMB and adds the result to each operator based on the operator's IMB consumption. For dependability, the designer must specify an error rate per BRAM (not including BRAM storage bits for IMC). As with the power values, the framework then uses the BRAM error rate to calculate the error rate per byte of IMB and adds the result to each operator based on the operator's IMB consumption. After finding the optimal device configuration based on the optimization target, the framework can calculate the final power and dependability results.

Once the IMLP method finds the optimal device configuration, the framework calculates the final power and dependability results by accumulating the resource consumptions of the operators. The device configuration also defines a single operator frequency, which the IMLP method sets as the lowest operational frequency considering all the operators used in the design configuration. The framework uses the operator frequency to scale the total operator power, which is the reason for normalizing the resource power consumptions by one MHz in the device database. However, only resource usage affects dependability, so the framework does not scale the error rates by the operator frequency. Although IMC resides in the BRAMs, the internal-memory extension cannot factor the IMC into the power and dependability analysis since the extension does not predict how much IMC is necessary. To predict the IMC usage, the framework must analyze the external memory as well.

### 3-3. External-Memory Extension

The external-memory extension enables the framework's complete memory-aware analysis by providing an EMB algorithm to wrap around the IMLP method. The external-memory extension uses a wrapper algorithm instead of directly modifying the IMLP method because the analysis of external-memory ports and external-memory devices is nonlinear for two reasons. First, the maximum number of external-memory ports that can fit on a device is relatively low, so the framework should not consider device configurations using a fractional number of ports (e.g., there is a significant difference between 2.7 ports and 2 ports).

Secondly, there is no method for encapsulating the EMB/IMC relationship in a linear equation that can be understood by the underlying LP method. Therefore, instead of directly modifying the IMLP method, the external-memory extension modifies the IMLP's inputs and outputs based on EMB and IMC usage.

The EMB algorithm tests the results of adding  $p$  external-memory ports to a device configuration by modifying the inputs and outputs of the IMLP method. First, the EMB algorithm tests that the device has enough resources (i.e., I/O pins, FFs, and LUTs) to actually fit  $p$  external-memory ports on the device. If there is enough room, the EMB algorithm subtracts the resources needed for the  $p$  external-memory ports from the resource quantities of the IMLP method's RLEs. Next, the EMB algorithm runs the IMLP method and adjusts the device configuration based on IMC usage and  $p$ . The EMB algorithm calculates the total power by summing the power result of the IMLP method with the extra power consumed by the  $p$  external-memory ports and the  $p$  external-memory devices, which the EMB algorithm calculates based on port-resource usage and the external-memory device type, respectively. Note from Table 1 that pins used for inputs require significantly less power than pins used for outputs or bidirectional signals. Also, IMC usage does not affect power because IMC represents memory storage, which uses static power, and static power remains the same independent of IMC usage. The EMB algorithm calculates the total error rate by summing the error rate result of the IMLP method with the extra error rate for each of the  $p$  external-memory ports and the extra error rate from the IMC usage (requires the designer to specify the error rate per IMC bit). By carefully choosing  $p$  and the IMC usage before running the IMLP method, the EMB algorithm can optimize a device configuration for either power or dependability.

To optimize for power, the EMB algorithm uses the maximum IMC available on the device to minimize the required EMB and determine the minimum  $p$ . Using the smallest possible  $p$  minimizes the power contribution from the external-memory ports. Furthermore, because IMC usage has no effect on power, the EMB algorithm can use the maximum amount of IMC with no negative effect on power. Therefore, with the optimization target set to power, the EMB algorithm is guaranteed to find the lowest power device configuration by running the IMLP method with the maximum IMC usage and minimum  $p$ .

Alternatively, when optimizing for dependability, the EMB algorithm tests multiple IMC usage and  $p$  combinations to find the right balance between the IMC usage and EMB. Ideally, using the minimum  $p$  and no IMC usage would produce the optimal device configuration, since both increase the overall error rate, but the EMB algorithm cannot reduce  $p$  without increasing IMC usage due to the EMB/IMC relationship. To find the optimal combination of  $p$  and IMC usage, the EMB algorithm begins by finding the minimum  $p$  using the maximum IMC usage. However, before running the IMLP method, the EMB algorithm reduces the IMC usage as

much as possible without requiring an increase in  $p$ . Minimizing IMC usage in this way guarantees that the IMLP method (with the optimization target set to dependability) will produce a device configuration with the minimum error rate for the current value of  $p$ . The EMB algorithm iterates this process, increasing  $p$  by one each step, until  $p$  becomes so large that the external-memory ports do not fit on the device or the IMLP methods fails to meet the CD requirement (due to the external-memory ports using too many FFs and LUTs). After collecting the optimal device configurations for each valid value of  $p$ , the EMB algorithm selects the optimal device configuration with the lowest error rate.

Since different external-memory devices may differ in their power consumption and EMB, the EMB algorithm optimizes for power or dependability for each type of external-memory device. From the results of these optimization tests on each external-memory device type, the EMB algorithm determines the optimal type of external-memory device and outputs this result as the final output along with the corresponding optimal device configuration.

#### 4. CASE STUDY

In order to accurately analyze an FPGA-based space-system design and predict the design’s power and dependability, our framework must correctly model inter-memory resource interactions with respect to the operations performed on the data. We demonstrate how designers specify an application and the application’s effect on the EMB/IMC relationship using the case study’s HSI-analysis application, which we approximate as a matrix multiplication (MM). First, we introduce our case study based on the Hyperion HSI sensor. Then we analyze our case study by investigating block MM for the dimensions specified in our case study.

##### 4-1. Hyperion Hyperspectral-Imaging Sensor

Our case study is based on the Hyperion HSI sensor on the EO-1 satellite mission [16]. The Hyperion HSI sensor captures an image cube (Figure 2) representing a ground scene every 2.95 seconds, which consists of 256 by 660 pixels and 220 12-bit spectral values per pixel, resulting in

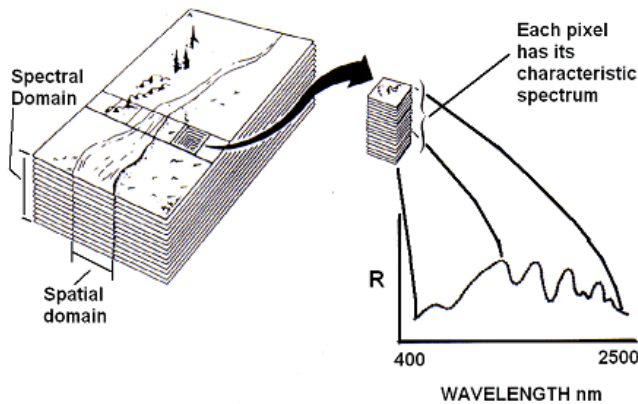


Figure 2 – HSI image cube and the spectral values of a single pixel [15]

18.9 MB/s of raw data. HSI analysis of the image cube can identify the locations of certain materials of interest within a scene, resulting in one or more two-dimensional images that lack the spectral dimension and are therefore much smaller than the original image cube. If a system could perform the HSI analysis quickly enough in situ, the EO-1 satellite would need to send only a small fraction of the original data to Earth, potentially enabling the continuous streaming of results though the limited downlink bandwidth. Our case study examines how our framework would aid in designing an FPGA-based space system to perform in situ HSI analysis for the Hyperion HSI sensor.

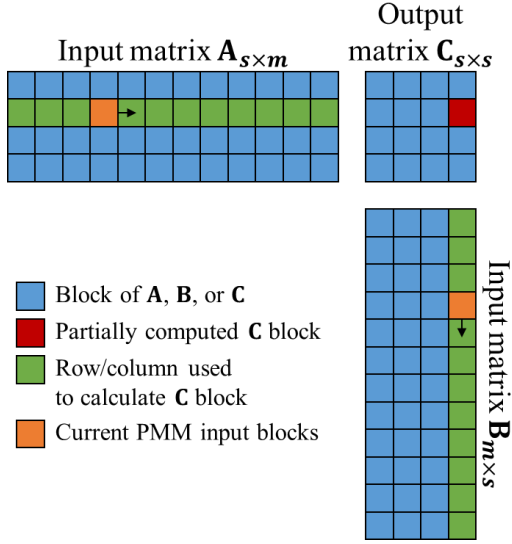
Although HSI analysis is a complex process, we can simplify our HSI computational model by looking at the HSI analysis’s largest constituent kernel. Jacobs et al. [17] profiled the HSI-analysis computations and determined that the majority of the computation was a single MM. For an image cube similar to the one produced by Hyperion, 97% of the HSI analysis computation is an MM where the first input matrix has dimensions of 220 (number of spectral bands) by 168,960 (number of pixels in a scene) and the second input matrix is a transpose of the first. Since the remaining 3% of computation is comparatively negligible, we can greatly simplify our case study analysis by modeling the HSI analysis as a single MM for these dimensions. We refer the reader to [2] for a more detailed explanation of HSI analysis and the Hyperion HSI mission.

The EO-1 satellite mission launched into space in November, 2000. Since then, the number of spectral bands and pixels that HSI sensors can gather has increased, with high-end sensors capturing up to 1,000 spectral bands and up to 1,000 pixels across [18]. Furthermore, most modern commercial FPGAs have a CD that is much higher than what would be required by the standard Hyperion HSI sensor. To compensate for the age of the Hyperion sensor and investigate a more modern mission, our case study considers an enhanced version of the Hyperion sensor that is capable of capturing an enhanced image cube that is roughly twice as large in every dimension as the original size. The other properties of the enhanced Hyperion sensor remain the same as the original sensor, resulting in the production of a  $500 \times 1300 \times 400$  image cube every 2.95 seconds.

##### 4-2. Case-Study Performance and Memory Analysis

We investigate block MM to analyze and quantify the case study’s memory requirements. For large matrices, like those in our case study, block MM is much more efficient with limited IMC than standard MM, so block MM is more representative of what a designer would actually use.

Figure 3 shows how block MM works for the case study with input matrices  $A$  and  $B$  and output matrix  $C$ . Since  $B$  is the transpose of  $A$ , we can define the dimensions  $s$  and  $m$  (for the side and middle dimensions of the MM, respectively), where  $A$  has dimensions  $s \times m$ ,  $B$  has dimensions  $m \times s$ , and  $C$  has dimensions  $s \times s$ . For our case study,  $s = 400$  for the number of spectral bands, and  $m = 650,000$  for the total



**Figure 3 – Block MM divides the input and output matrices into square blocks of size  $n \times n$  and computes one  $C$  block at a time using multiple partial MMs**

number of pixels in an image cube. The FPGA divides each matrix into a set of square blocks of size  $n \times n$  and computes one  $C$  block at a time using the corresponding row of blocks in  $A$  and column of blocks in  $B$ . The FPGA processes the  $A$  row and  $B$  column one block at a time by fetching the  $A$  and  $B$  blocks from external memory, storing these blocks in the FPGA's IMC, and performing a partial MM by multiplying the  $A$  and  $B$  blocks to calculate a partial  $C$  block. The FPGA accumulates the sum of all the partial  $C$  blocks from all the partial MMs of a row/column pair to produce the final  $C$  block, which the FPGA stores to the external memory from the IMC. Based on the dimensions of the matrices, the FPGA must perform  $m/n$  partial MMs for each of the  $(s/n)^2$   $C$  blocks in  $C$ , for a total of  $ms^2/n^3$  partial MMs per block MM. However, since  $B$  is the transpose of  $A$ ,  $C$  must be symmetric, requiring the FPGA to only calculate half of  $C$  and therefore perform only  $ms^2/(2n^3)$  partial MMs per block MM.

Each partial MM is effectively a standard MM performed on an  $A$  and  $B$  block to produce a partial  $C$  block. As Figure 4 shows, each entry in the partial  $C$  block requires an  $n$ -length dot product consisting of  $n$  multiplies and  $n$  additions. Since there are  $n^2$  entries in the partial  $C$  block, the FPGA must perform  $2n^3$  operations for each partial MM. Therefore the required CD to calculate a full block MM in  $\tau = 2.95$  seconds is:

$$CD = \frac{ms^2}{2n^3} \times 2n^3 \times \frac{1}{\tau} = \frac{ms^2}{\tau} = 35.254 \text{ GOPS} \quad (3)$$

Similarly, because each dot product requires  $n$  words from the  $A$  block and  $n$  words from the  $B$  block, and each word is 4 bytes long, the required IMB is:

$$IMB = 4 \times \frac{ms^2}{2n^3} \times 2n^3 \times \frac{1}{\tau} = \frac{4ms^2}{\tau} = 141.02 \text{ GB/s} \quad (4)$$

Since the FPGA only stores the final  $C$  blocks into external memory, and there are  $(s/n)^2$   $C$  blocks of size  $n^2$ , the required output EMB is:

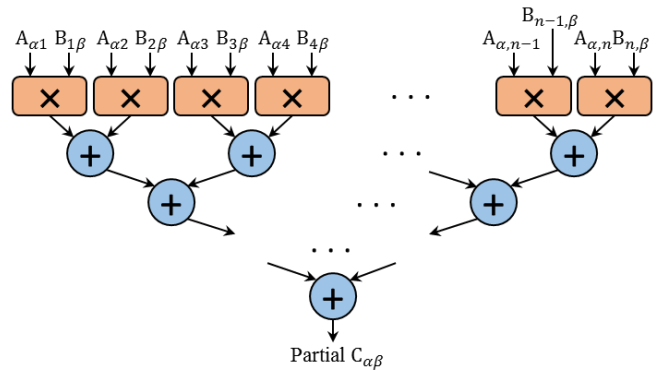
$$EMB_{out} = 4 \times n^2 \times \left(\frac{s}{n}\right)^2 \times \frac{1}{\tau} = \frac{4s^2}{\tau} = 216.95 \text{ kB/s} \quad (5)$$

Finally, for each partial MM, the FPGA must fetch an  $A$  and  $B$  block from external memory into the IMC, requiring an input EMB of:

$$EMB_{in} = 4 \times 2n^2 \times \frac{ms^2}{2n^3} \times \frac{1}{\tau} = \frac{4ms^2}{n\tau} = \frac{141.02}{n} \text{ GB/s} \quad (6)$$

Comparing (6) to (4) confirms that the lower bound for input EMB is equal to IMB, since  $n$  cannot be less than 1. Unlike the CD, IMB, and output EMB, the required input EMB is dependent on the block size, as shown by the remaining  $n$  term. Since  $n$  is directly related to the IMC usage, we can define the relationship between IMC usage and  $n$  to express the required input EMB in terms of the IMC usage.

For the FPGA to correctly perform a partial MM, the  $A$  and  $B$  blocks for the next partial MM cannot replace the current  $A$  and  $B$  blocks in IMC until the current partial MM completes. However, if the FPGA waits until the completion of one partial MM to start fetching the next  $A$  and  $B$  blocks, there will be stall between successive partial MMs when the operations cannot process data, thereby reducing performance. To resolve this issue, the FPGA stores  $A'$  and  $B'$  blocks for the next partial MM in addition to the  $A$  and  $B$  blocks. When the FPGA operates on the  $A$  and  $B$  blocks, the FPGA also begins fetching the  $A'$  and  $B'$  blocks. When the current partial MM completes, the next partial MM begins with the  $A'$  and  $B'$  blocks, which become the  $A$  and  $B$  blocks, and the FPGA can begin fetching the next partial MM's  $A'$  and  $B'$  blocks. With this pre-fetching method, the operators can continuously process data while the external-memory ports fetch new data into the IMC. Therefore, the IMC usage must be large enough to store the  $A$ ,  $B$ ,  $A'$ ,  $B'$ , and partial  $C$  blocks. Since each block has  $n^2$  4-byte entries, we calculate the IMC usage as:



**Figure 4 – Each entry in the partial  $C$  block requires  $n$  multiply and  $\sim n$  addition operations and  $n$  words from both the  $A$  and  $B$  blocks**

$$\text{IMC}_B = 4 \times 5n^2 = 20n^2 \text{ Bytes} \quad (7)$$

From (7), we can calculate  $n$  based on bytes of IMC usage:

$$n = \sqrt{\frac{\text{IMC}_B}{20}} \quad (8)$$

Combining (6) and (8) produces the required input EMB as a function of IMC usage:

$$\text{EMB}_{\text{in}} = 141.02 \left( \sqrt{\frac{\text{IMC}_B}{20}} \right)^{-1} = \frac{630.65}{\sqrt{\text{IMC}_B}} \text{ GB/s} \quad (9)$$

Comparing (6) and (5) shows that the required input EMB is greater than the required output EMB if and only if  $m > n$ . Since  $m$  is always greater than  $n$ , the required output EMB will never exceed the required input EMB. Our case study uses DDR external-memory ports, so the overall required EMB is only equal to the greater of the required input and output EMBs, and thus:

$$\text{EMB} = \text{EMB}_{\text{in}} = \frac{630.65}{\sqrt{\text{IMC}_B}} \text{ GB/s} \quad (10)$$

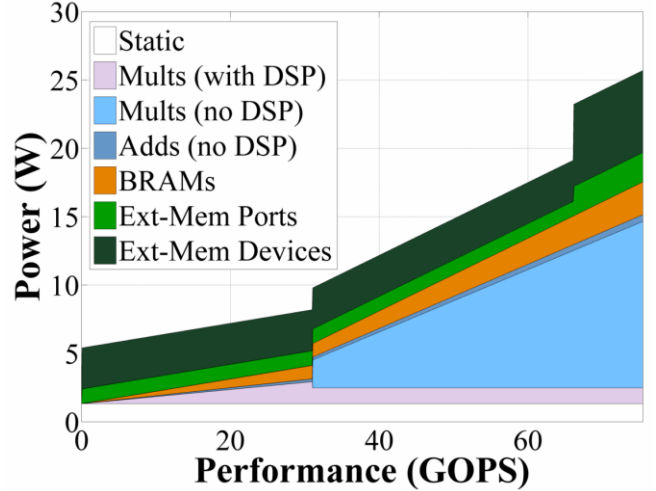
With the result from (10), the designer can completely specify the EMB/IMC relationship, enabling the framework to perform the memory-aware analysis.

## 5. RESULTS AND ANALYSIS

Using the HSI mission as our case study, we perform three experiments to demonstrate the memory-aware analysis enabled by our framework's memory extension. We evaluate all of the LX models from the Virtex-4, Virtex-5 and Virtex-6 FPGA families as well as LXT models from the Virtex-6 family (included because there is only one LX model for Virtex-6). For each model, we only consider the most capable package in terms of highest I/O pin count and fastest speed grade. To analyze the external-memory ports, we use the Xilinx CORE Generator System [19] to generate the latest generation of DDR port available for each device (DDR3 for Virtex-6 and DDR2 for Virtex-4 and Virtex-5). For the external-memory device, we assume a standard power consumption of 3 Watts and no error rate due to radiation hardening. First, we show how increasing the required performance of the case study affects the power and dependability of an FPGA-based space system. We then analyze the effects of varying device size by testing different sized FPGAs within the same family. Finally, we show how the framework finds the Pareto-optimal designs from multiple families of FPGAs.

### 5-1. Effects of Increasing Required Performance

To better understand the results from the framework's memory-aware analysis, we investigate the optimal device configurations for power and dependability for a single device as HSI analysis performance requirements increase. We choose to investigate the Virtex-5 LX155-FF1760-3 due to the device's average performance within the Virtex-5



**Figure 5 – Sources of power consumption as performance requirements of HSI analysis increase for a system using a Virtex-5 LX155-FF1760-3 device with a power-optimized device configuration**

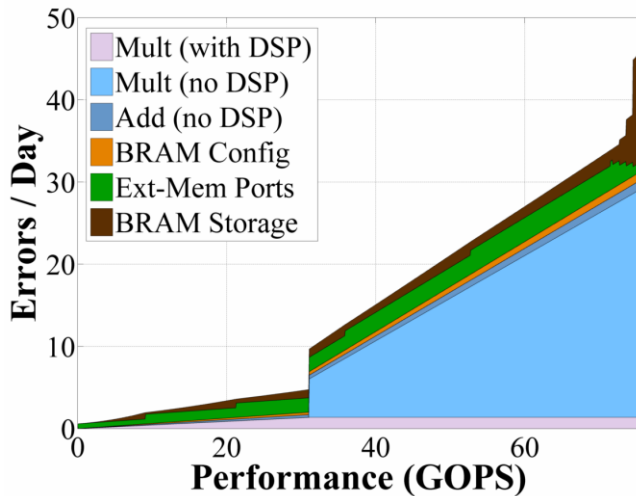
family, and the Virtex-5's average performance relative to the other FPGA families evaluated. We adjust the performance requirement by adjusting  $\tau$ , the time allowed to process an image cube. The actual value of  $\tau$  for the HSI mission is 2.95 seconds, which requires 35.254 GOPS, so the appropriate  $\tau$  for a desired required CD is:

$$\tau = \frac{3.5254 \times 10^{10}}{\text{CD}} \quad (11)$$

We analyze an FPGA-based space system with a Virtex-5 LX155-FF1760-3 FPGA device and a power-optimized device configuration. Figure 5 shows the varying power consumption of the FPGA-based space system's external-memory devices and the resources and operations of the FPGA device as the performance requirements of the HSI analysis increase. For performance below 10 GOPS, system power is dominated by the FPGA's static power (1.322 W) and the power of a single external-memory device and external-memory port on the FPGA (external-memory devices and external-memory ports do not have a low-power mode). Between 0 and 30 GOPS, power consumption rises primarily from an increasing number of multiply operations (using DSPs) and increasing BRAM activity due to increasing IMB. Although there are always an equal number of multiply and add operations on the FPGA, the power consumption of each add operation is approximately one tenth of the power consumption of a multiply operation with DSP units.

At 31 GOPS, the FPGA requires all available DSP units to sustain the required rate of 32 multiplies per cycle. When performance exceeds 31 GOPS, the FPGA must begin using the more power-hungry non-DSP multiply operations, which consist only of FFs and LUTs. The non-DSP multiply operations have a lower maximum operating frequency than the other operations, so the FPGA must reduce the overall frequency so that all operations function properly together.



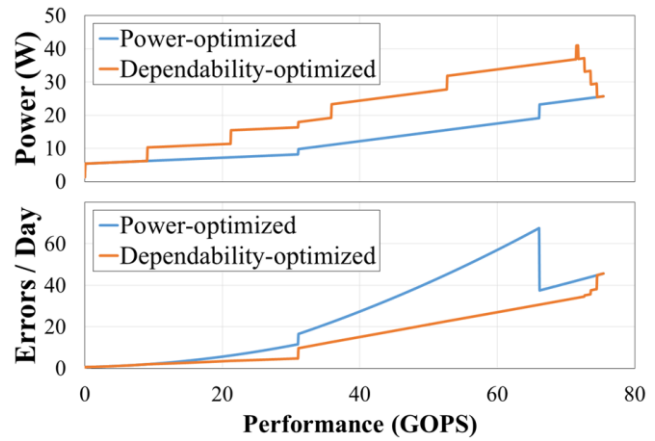


**Figure 6 – Sources of errors as performance requirements of HSI analysis increase for a system using a Virtex-5 LX155-FF1760-3 device with a dependability-optimized device configuration**

Therefore, the FPGA needs more total operations to keep up with the performance requirement at the reduced frequency. Since there are no more DSP units, the FPGA must add 13 new non-DSP multiply operations as soon as performance exceeds 31 GOPS, increasing the total power consumption by approximately 1.6 W. At 66 GOPS, the external-memory port can no longer keep up with the EMB requirement, requiring another external-memory port and external memory device, which increases the total power consumption by approximately 4 W. At 75.44 GOPS, the system reaches maximum performance, consuming 25.69 W of power and using all of the available FFs and DSP units and most of the available LUTs.

We also analyze the same FPGA-based space system using a dependability-optimized device configuration. Figure 6 shows the varying error rates for each of the FPGA’s resources and operations as the performance requirements of the HSI analysis increase. Of the three operations, non-DSP multiply operations have the highest error rate since these operations use approximately 35 times more FFs and LUTs than the add operations and do not benefit from the relatively-low error rates of the DSP units. Similar to the power trends (Figure 5), at 31 GOPS, the FPGA must incorporate several non-DSP multiply operations, resulting in a doubling of the error rate.

There is also an interesting dynamic between the external-memory ports and IMC usage. As performance increases up to 9 GOPS, the FPGA uses only one external-memory port, so the FPGA must increase IMC usage in order to use the data from the single memory port more effectively. At 9 GOPS, IMC usage becomes so large that the FPGA can add an additional external-memory port to significantly reduce the IMC usage and achieve a lower total error rate. This tradeoff occurs several more times as performance increases, reaching as high as six external-memory ports at 71.5 GOPS.



**Figure 7 – Total power consumption and error rates as performance requirements of HSI analysis increase for a system using a Virtex-5 LX155-FF1760-3 device with a device configuration optimized for either power or dependability**

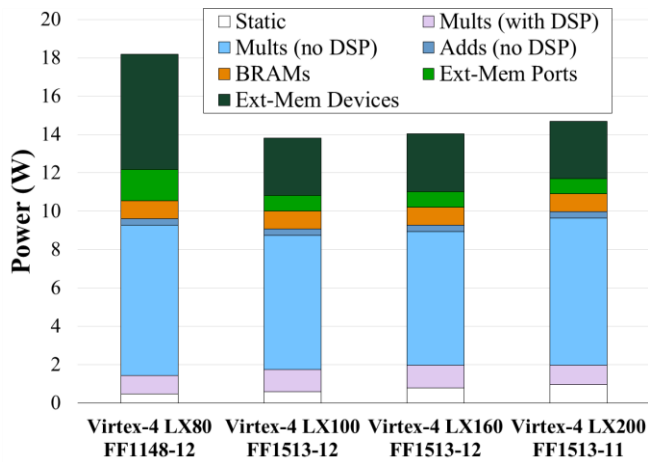
However, at 71.8 GOPS, the operations and external-memory ports use all of the available DSP units and FFs, so the FPGA must achieve further performance increases by reducing the external-memory ports in order to increase the number of operations. This tradeoff reduces the number of external-memory ports below the optimal amount and significantly increases the overall error rate. At 75.44 GOPS, the system again reaches maximum performance with a total error rate of 45.69 errors per day.

Note that Figure 5 shows an FPGA-based space system designed for optimal power, and Figure 6 shows an FPGA-based space system designed for optimal dependability, so the two figures do not represent the same system design. Figure 7 shows the tradeoff that exists between the power-optimized and dependability-optimized designs. Since the dependability-optimized design preemptively increases the number of external-memory ports to keep IMC usage low, the design requires more power due to the external-memory devices and ports as compared to the power-optimized design. Alternatively, the power-optimized design avoids increasing the number of external-memory ports to keep the power consumption low, but does so despite rapidly increasing error rates due to increasing IMC usage.

### 5-2. Effects of Varying Device Size

To analyze the power and dependability of an FPGA family for the performance requirements of the HSI case study ( $\tau = 2.95$  seconds and required CD = 35.254 GOPS), we use the Virtex-4 FPGA device family because the Virtex-4’s limited processing and memory capabilities produce the most interesting trends in power and dependability. Of the eight Virtex-4 LX models evaluated, we only consider the largest four models because the other models cannot meet the case study’s required performance.

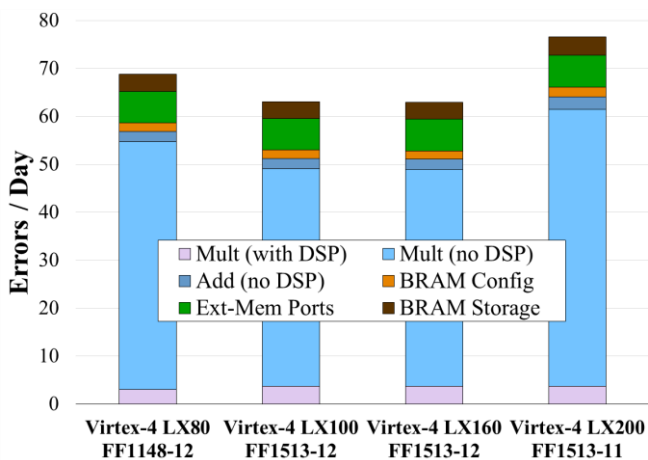
Figure 9 shows the sources of power consumption for FPGA-based space systems using a Virtex-4 LX device with a



**Figure 9 – Sources of power consumption for a system using a Virtex-4 LX device with a device configuration optimized for power**

device configuration optimized for power. The LX80 device consumes three to four Watts more power than the other three devices due to the LX80's smaller IMC, requiring two external-memory devices and ports to achieve the required performance. Between the largest three devices, static power consumption is the largest differentiator. Although the largest three devices have an equal number of DSP units, the non-DSP units consume more power in the LX200 device because of the device's lower speed grade. Since the LX200 device's DSP units operate at a slower frequency, the device must use more power-hungry, non-DSP multiply operations. The LX100 device achieves the lowest power consumption in the Virtex-4 family due to the device's low static power, high number of DSP units and BRAMs, and high speed grade.

Figure 10 shows the sources of errors for FPGA-based space systems using a Virtex-4 LX device with a device configuration optimized for dependability. The LX200 device still suffers a higher error rate due to the device's low speed grade and greater usage of non-DSP multiply operations. The LX80 device no longer uses more external-



**Figure 10 – Sources of errors for a system using a Virtex-4 LX device with a device configuration optimized for dependability**

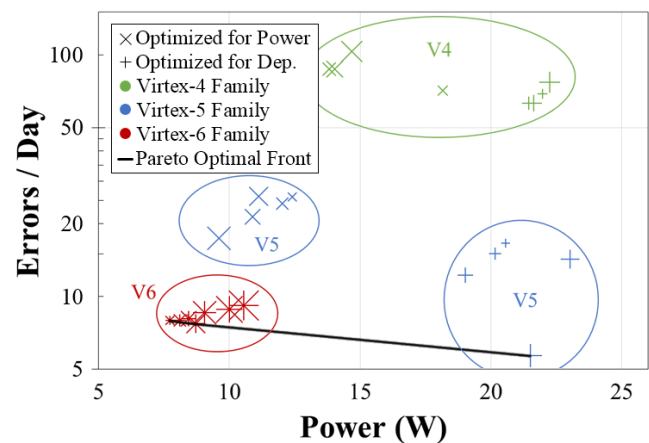
memory devices and ports than the other three FPGA devices (all four FPGA devices use three external-memory devices and ports). However, the LX80 also has fewer DSP units than the other three devices, so the LX80 device uses more non-DSP multiply operations than the LX100 and LX160 devices. The LX100 and LX160 have the lowest error rates due to the devices' high speed grades and high number of DSP units.

### 5-3. Finding Pareto-Optimal Designs

The final output of our framework is the set of Pareto-optimal designs for the mission. A Pareto-optimal design is a design that, when compared to any other design, is superior with respect to at least one of the mission goals (i.e., power or dependability). Therefore, the Pareto-optimal set presents designers with designs that specialize in power, dependability, or some combination of the two, allowing the designer to make the final decision of which mission goals are most critical.

Figure 8 shows the power and dependability results for the case study for all FPGA-based space systems using a Virtex-4, Virtex-5, or Virtex-6 FPGA device. The Pareto-optimal set shows the tradeoff space between the various Pareto-optimal designs. Each marker represents a unique system design (i.e., unique combination of FPGA device and optimization target), with larger markers representing a relatively larger model within the system's device's family. We did not include designs that cannot achieve the case study's required performance (the smallest two Virtex-5 and four Virtex-4 devices).

Aside from the Virtex-5 LX 330 designs, every Virtex-6 design performs better in power and dependability than every Virtex-4 and Virtex-5 design. Furthermore, the Virtex-6 designs show no variation based on the optimization target. This is because the device CD of every Virtex-6 FPGA device is significantly higher than the case study's required CD. As seen in Figure 7, a similar phenomenon occurs with



**Figure 8 – Graph of error rates versus power for Virtex-4, -5, and -6 FPGA families (larger markers represent larger devices within a family) showing final Pareto-optimal front traced on top of the four members of the Pareto-optimal set**

Virtex-5 devices performing less than 10 GOPS, where the choice of optimization target has no effect on the power or dependability. For Virtex-6 designs, power and dependability worsen with increasing device size (except for the dependability of the Virtex-6 LX130T and LX240T designs). However, this is not a general rule for every device family as seen with the Virtex-4 and Virtex-5 designs.

The Virtex-6 LX130T and LX240T designs stand out with slightly improved dependability due to the devices' add operations, which run slightly faster than those of the other Virtex-6 designs. Since the Virtex-6 designs do not use any non-DSP multiply operations for the required CD, the add operations are the slowest operation and therefore determine the operator frequency of the FPGA. The higher operator frequency results in a lower number of operators needed to meet the required CD, resulting in higher dependability since there are fewer operators to experience an error. Finally, the high dependability of the Virtex-5 LX330's dependability-optimized design is due to a combination of the Virtex-5's inherently low configuration memory error rate and the Virtex-5 LX330 device's high number of DSP units, allowing the device to meet the performance requirement without using non-DSP multiply operations.

**Table 2. Power consumption and error rates for the four Pareto-optimal designs**

Device	Opt. Target	Power (W)	Errors /day
Virtex-6 LX75T-FF784-3	N/A	7.751	7.962
Virtex-6 LX130T-FF784-3	N/A	8.128	7.950
Virtex-6 LX240T-FF1759-3	N/A	8.726	7.718
Virtex-5 LX330-FF1760-2	Depend.	21.52	5.672

Table 2 shows the four Pareto-optimal designs for the case study. The smallest Virtex-6 design offers the best power and reasonable dependability. The mid-range Virtex-6 LX240T design offers slightly better dependability, but at the cost of one more Watt of power consumption. The largest Virtex-5 dependability-optimized design offers the most dependability, but this comes at the cost of more than double the power consumption over the Virtex-6 designs.

## 6. CONCLUSIONS

In this paper, we have introduced our framework's memory extension, which enables memory-aware analysis by refinements to our framework's original analysis, further improving our framework's ability to evaluate and optimize increasingly complex FPGA-based space systems. The memory-aware analysis more accurately predicts an FPGA-based space system's power and dependability by modeling the internal-memory capacity (IMC), internal-memory bandwidth (IMB), and external-memory bandwidth (EMB)

memory resources. Due to the non-trivial and application-dependent nature of the EMB/IMC relationship, the designer must specify this relationship in addition to the application before the framework can begin analysis. The memory extension expands our framework's analysis to a more complete and accurate holistic design view revealing that the best systems do not always use the fewest external-memory devices or smallest, largest, or most modern FPGA devices.

To demonstrate the importance of our framework's memory extension, we investigated a case study based on an enhanced version of a hyperspectral-imaging (HSI) satellite mission. We used the case study to show the method a designer might use to analyze their application and determine their application's EMB/IMC relationship. For the case study mission, our framework evaluated 22 Virtex family FPGAs, determined power- and dependability-optimized device configurations for each device, and selected four Pareto-optimal designs ranging from very-low power to high dependability. Results show that memory resources may limit the performance of a system and contribute significantly towards power and dependability results.

Future work includes enabling the IMLP method to optimize for a combination of power and dependability, rather than one or the other. As seen with the Virtex-5 LX330 design, there can be a large variance in power and dependability between power-optimized and dependability-optimized device configurations. Determining one or more designs that are a compromise between these two extremes enables a broader range of design options that could lead to a more complete set of Pareto-optimal designs.

## ACKNOWLEDGMENTS

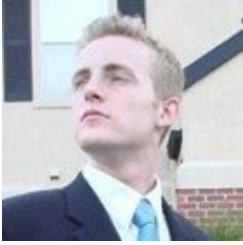
This work was supported in part by the IUCRC Program of the National Science Foundation under Grant No. EEC-0642422. The authors gratefully acknowledge vendor equipment and tools provided by Xilinx that helped make this work possible.

## REFERENCES

- [1] N. R. Mahapatra and B. Venkatrao, "The processor-memory bottleneck: Problems and solutions," *Crossroads - Computer Architecture*, vol. 5, no. 2, pp. 1-8, 1999.
- [2] N. Wulf, A. D. George and A. Gordon-Ross, "A framework to analyze, compare, and optimize high-performance, on-board processing systems," in *2012 IEEE Aerospace Conf.*, Big Sky, MT, 2012.
- [3] N. Wulf, J. Richardson and A. D. George, "Optimizing FPGA performance, power, and dependability with linear programming," in *Proc. Military and Aerospace Programmable Logic Devices Conf. (MAPLD)*, San Diego, CA, 2013.
- [4] J. Williams, A. D. George, J. Richardson, K. Gosrani,

- C. Massie and H. Lam, "Characterization of fixed and reconfigurable multi-core devices for application acceleration," *ACM Trans. Reconfigurable Technology and Syst. (TRETTS)*, vol. 3, no. 4, pp. 19:1-19:29, 2010.
- [5] J. Richardson, S. Fingulin, D. Raghunathan, C. Massie, A. D. George and H. Lam, "Comparative analysis of HPC and accelerator devices: Computation, memory, I/O, and power," in *2010 4th Int. Workshop High-Performance Reconfigurable Computing Technology and Applicat. (HPRCTA)*, New Orleans, LA, 2010.
- [6] K. D. Underwood and K. S. Hemmert, "Closing the gap: CPU and FPGA trends in sustainable floating-point BLAS performance," in *2004 12th Annu. IEEE Symp. Field-Programmable Custom Computing Machines (FCCM 2004)*, Napa, CA, 2004.
- [7] Y. Dou, S. Vassiliadis, G. K. Kuzmanov and G. N. Gaydadjiev, "64-bit floating-point FPGA matrix multiplication," in *Proc. 2005 ACM/SIGDA 13th Int. Symp. Field-Programmable Gate Arrays (FPGA 2005)*, Monterey, CA, 2005.
- [8] Xilinx, "Xilinx Power Estimator (XPE)," [Online]. Available: [http://www.xilinx.com/products/design\\_tools/logic\\_design/xpe.htm](http://www.xilinx.com/products/design_tools/logic_design/xpe.htm). [Accessed 2014].
- [9] Altera, "PowerPlay Early Power Estimator (EPE)," [Online]. Available: <http://www.altera.com/support/devices/estimator/pow-powerplay.jsp>. [Accessed 2014].
- [10] Vanderbilt University and NASA, "CREME96," [Online]. Available: <https://creme.isde.vanderbilt.edu/>. [Accessed 2014].
- [11] European Space Agency, "SPENVIS," [Online]. Available: <https://www.spennis.oma.be/>. [Accessed 2014].
- [12] A. Sari, D. Agiakatsikas and M. Psarakis, "A soft error vulnerability analysis framework for Xilinx FPGAs," in *Proc. 2014 ACM/SIGDA Int. Symp. Field-Programmable Gate Arrays (FPGA 2014)*, Monterey, CA, 2014.
- [13] Xilinx, "Considerations Surrounding Single Event Effects in FPGAs, ASICs, and Processors," 7 March 2012. [Online]. Available: [http://www.xilinx.com/support/documentation/white\\_papers/wp402\\_SEE\\_Considerations.pdf](http://www.xilinx.com/support/documentation/white_papers/wp402_SEE_Considerations.pdf). [Accessed 2014].
- [14] Xilinx, "Device Reliability Report (UG116)," 27 January 2012. [Online]. Available: <http://projects.itn.pt/adonics/XIL12.pdf>. [Accessed 2014].
- [15] G. Vane, "High spectral resolution re-mote sensing of the earth," *Sensors*, vol. 2, pp. 11-20, 1985.
- [16] NASA Goddard Space Flight Center, "Earth Observing-1," [Online]. Available: <http://eo1.gsfc.nasa.gov/>. [Accessed 2014].
- [17] A. Jacobs, C. Conger and A. D. George, "Multiparadigm space processing for hyperspectral imaging," in *2008 IEEE Aerospace Conf., Big Sky, MT*, 2008.
- [18] Spectral Cameras.com, "AISA Airborne Hyperspectral Imaging Systems," [Online]. Available: <http://www.spectralcameras.com/aisa>. [Accessed 2014].
- [19] Xilinx, "Xilinx CORE Generator System," [Online]. Available: <http://www.xilinx.com/tools/coregen.htm>. [Accessed 2014].

## BIOGRAPHY



*Nicholas Wulf is a doctoral student in ECE at the University of Florida. He is a research assistant in the advanced processing devices group in the NSF CHREC Center at Florida. His research interests include analysis and comparison of fixed*

*and reconfigurable device architectures and low-overhead fault-tolerant techniques.*



*Alan D. George is Professor of ECE at the University of Florida, where he serves as Director of the NSF Center for High-performance Reconfigurable Computing known as CHREC. He received the B.S. degree in CS and M.S. in ECE from the University of Central Florida,*

*and the Ph.D. in CS from the Florida State University. Dr. George's research interests focus upon high-performance architectures, networks, systems, services, and applications for reconfigurable, parallel, distributed, and fault-tolerant computing. He is a Fellow of the IEEE.*



*Ann Gordon-Ross (M'00) received her B.S and Ph.D. degrees in Computer Science and Engineering from the University of California, Riverside (USA) in 2000 and 2007, respectively. She is currently an Associate Professor of ECE at the University of*

*Florida and is a member of the NSF Center for High Performance Reconfigurable Computing (CHREC). Her research interests include embedded systems, computer architecture, low-power design, reconfigurable computing, dynamic optimizations, hardware design, real-time systems, and multi-core platforms.*