# A Technical Report on Design Space Exploration and CLB Customization for Application-Specific FPGAs

Mark Hammerquist, Roman Lysecky
Department of Electrical and Computer Engineering
University of Arizona, Tucson, AZ
{hansolo, rlysecky}@ece.arizona.edu

## Abstract

*The inclusion of field programmable gate arrays (FPGAs) within a system-on-a-chip (SOC) design offers programmability, flexibility, and reconfigurability not possible with application-specific integrated circuits (ASIC) or full-custom implementations. However, these benefits come at the expense of significant area, performance, and power consumption overheads compared to ASIC or full-custom circuits. As a typical SOC design will require fabrication of the final integrated circuit, rather than rely on a generic FPGA architecture, an FPGA integrated within an SOC design can be optimized for the specific intended application by tailoring an FPGA's architectural features for a specific hardware circuit to improve the area, delay, or energy consumption compared to traditional FPGA designs. Such an application-specific FPGA (ASFPGA) would have reduced overheads compared to ASIC and full custom implementations. We present two methodologies for creating ASFPGAs including a design space exploration framework for customizing FPGA architectural elements and a configurable logic block (CLB) customization algorithm intended to reduce area requirements. The resulting ASFPGA generation methods can be utilized to create various ASFPGAs ranging from a customized, yet flexible, FPGA architecture to a fully customized FPGA architecture with significant area savings over traditional FPGAs.*

## Keywords

Application-specific FPGAs, reconfigurable computing, design space exploration.

## 1. INTRODUCTION

The inclusion of field programmable gate arrays (FPGAs) within a system-on-a-chip (SOC) design offers many advantages over purely application-specific integrated circuits (ASIC) or full-custom implementations. Figure 1 presents a basic SOC architecture incorporating ASIC and FPGA alternatives for implementing custom hardware circuits. FPGAs can implement any hardware circuit simply by downloading bits for that hardware circuit, much in the same way that microprocessors can execute any software program simply by downloading an application binary. In this manner, FPGAs extend the flexibility of software design to hardware circuits, allowing hardware modifications, corrections, upgrades, etc. throughout the development cycle and even after device fabrication. For example, often costly hardware design errors that would require a respin in a traditional ASIC implementation can be fixed in hardware within the FPGA by downloading the corrected hardware circuit. Similarly, FPGAs enable rapid development efforts in which the hardware implementation does not need to be finalized before fabrication. Instead, the physical SOC hardware incorporating an FPGA can be fabricated earlier in the design process, as designers need not finalize the hardware design for those elements implemented within the FPGA before manufacturing.

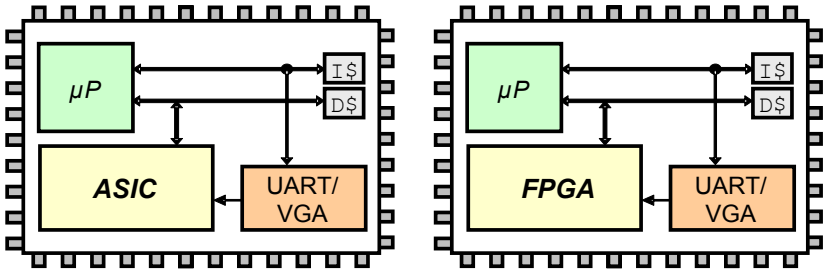
An FPGA can also be reconfigured at runtime to implement multiple hardware circuits throughout an application's execution using the same physical resources. Hardware/software codesign approaches targeting FPGAs can also provide significant performance benefits and/or power savings compared to software only implementations.

However, the reconfigurability and flexibility of FPGAs comes at the costs of significant area, performance, and power consumption overheads compared to ASIC circuits. Research has demonstrated that FPGAs require 10-40X greater area, 5-12X greater power consumption, and 3-4X greater critical path delays compared to their equivalent ASIC counterparts [Kuon and Rose 2005]. While rapid increases in IC capacities may alleviate the area concern, the power and performance overheads may present significant hurdles for allowing FPGAs to be integrated within hardware designs.

While many platform-based SOC design options exist, typical SOC designs require the fabrication of an ASIC or full-custom design, and provide an opportunity to tailor the design of the FPGA itself to the specific target application. Rather than rely on a generic FPGA architecture – typically optimized to perform well for a large variety of applications – an FPGA can be optimized for the intended application, thereby requiring less area with greater performance and lower energy consumption.

In this paper, we present a two new methods for creating such application-specific FPGAs (ASFPGAs). An ASFPGA can be created for the specific hardware circuit to be implemented within an SOC. The ASFPGA generation process outputs an architectural description of the FPGA needed to fabricate the SOC as well as a bitstream to program the resulting ASFPGA after fabrication. We present both a design space exploration framework – originally presented in [Hammerquist and Lysecky 2008] – along with a configurable logic bock (CLB) customization algorithm for generating an ASFPGA by tailoring several FPGA architectural features for a specific hardware circuit to improve the area, delay, or energy consumption compared to traditional FPGA designs. Section 2 provides an overview of related work in creating customized FPGA architectures and complementary techniques for creating a physical layout from an FPGA architectural description. In Section 3, we present a detailed description of a generic FPGA architecture and specifically highlights the configurable attributes of an FPGA that can be customized. In Section 4, we present our design space exploration framework for ASFPGA generation with experimental results highlighting the area, delay, and power benefits of ASFPGA compared to traditional FPGA architectures. In Section 5, we present an algorithm for customizing the architecture of the CLBs within an FPGA specifically intended to further reduce area requirements compared to our design pace exploration methodology. Finally, in Section 8 we conclude and discusses several directions for future work.

**Fig. 2:** Basic System-on-a-chip (SOC) architecture incorporating (a) ASIC and (b) FPGA alternatives for custom hardware circuits.




(a) (b)

## 2. RELATED WORK

Significant research has focused on providing automated methods for creating reconfigurable devices as either standard cell technologies or custom physical layouts.

Several research and development efforts have focused on efficiently implementing FPGAs – or similar reconfigurable devices – using standard cell technologies. eASIC has developed a technology which consists of coarse-grained reconfigurable logic cells that create a structured reconfigurable array [Levinthal and Herveille 2005]. The structured array has the ease of use and prototype cost of an FPGA, with the speed, density, power consumption, and production cost similar to a standard cell design. The reconfigurable logic cells consist of two 3-input LUTs connected to a flip-flop through a multiplexer. A two-input NAND gate drives one input of each of the LUTs, allowing the LUT to perform a subset of four-input functions in addition to being able to perform any three-input function. Custom interconnect is used inside the logic cells and fixed metal routing is used for cell-to-cell connections. While such fixed routing channels provide significant area advantages over the flexible routing resources in traditional FPGAs, the lack of configurable routing can have impacts on the ability to implement alternative hardware designs within the reconfigurable array.

In [Buch 2005], eASIC structured reconfigurable array can be utilized to create flexible SOC designs that incorporate IP blocks including microprocessors, peripherals, and DSP functions with embedded blocks for SRAMs, traditional FPGA fabrics, high-speed I/O, and clock management. By incorporating both eASIC's reconfigurable logic cells with traditional FPGA fabric, varying degrees of flexibility can be achieved. For example, IP cores that require high-performance and are intended to be utilized across many designs can be implemented within the eASIC's reconfigurable logic cells, whereas IP cores only required for a single can be implemented within the traditional FPGA. This methodology provides designers with more options for supporting flexible design elements.

In [Aken'ova et al. 2005], FPGA specific standard cells are proposed to enable more efficient implementation of FPGAs within standard cell technologies by imposing structure specific to the FPGA. However, these tailored FPGAs may not have sufficient logic resources to implement additional design elements. A designer must thus choose a larger tailored FPGA if future modifications or additions are anticipated.

In [Padalia et al. 2003; Kuon et al. 2005], researchers present a method for automatically generating a transistor-level implementation of an FPGA starting from an architectural description of the FPGA. The input to this system is similar to the FPGA architectural description file input of VPR [Betz et al. 1999]. The resulting FPGA layouts were verified through the successful fabrication of the resulting FPGA implementation and were shown to incur a 36% area overhead compared to similar commercial FPGAs. We anticipate that these automated FPGA layout generation can be utilized to fabricate the resulting ASFPGA architectures proposed within our approach.

In [Holland and Hauck 2005; Holland and Hauck 2006], an automated tool flow is presented for creating domain-specific PLAs, PALs, and CPLDs in order to reduce design time. Within this proposed approach, a domain can be loosely defined as a set of similar applications, such as sequential, combinational, floating point, arithmetic or encryptions domains. By analyzing the netlists of a set of applications within the target domain, a simulated annealing approach is used to optimize the reconfigurable device in terms of inputs, product terms, and outputs. Optimizing these parameters allowed for the reduction of programmable connections needed in the crossbar interconnect. Through the utilization of automatic layout generation tools, the final output is a physical layout for the domain-specific reconfigurable device. Using area-delay product as the performance metric, the resulting domain-specific CPLDs outperformed fixed-architecture CPLDs even when the fixed-architecture was handpicked for the domain.

Further research has been conducted in developing domain-specific CPLDs [Holland and Hauck 2006] that replace the full crossbar interconnect with a sparse crossbar along with incorporating additional CPLD resources needed to provide support for future unknown circuits that exist in the target domain. Additional, early work on extending this approach to support domain-specific FPGAs was presented in [Phillips 2004]. Whereas a domain-specific reconfigurable device is optimized for a particular domain, our proposed application-specific FPGA will be optimized for one specific hardware application, which provides an opportunity to further optimize a FPGA architecture beyond a set of applications that define a domain.

The availability of tools for automatically creating a standard cell or full-custom FPGA implementation is essential for enabling the proposed integration of ASFPGAs within a hardware design. As such, our proposed approach focuses on the optimization of the FPGA architecture for the intended hardware application and not on the physical generation of the resulting ASFPGA.

A typical FPGA routing architecture uses about 70-90% of the total transistors on an FPGA [Dehon 1996], thus using most of the area, delay and power. Therefore, in [Sivaswamy et al. 2005], the authors propose forming hardwired junctions between horizontal and vertical wire segments inside switch boxes. The junctions are in the shapes of T's, L's, +'s and their rotated versions. As a result of hardwiring connections, some programmable switches are eliminated, decreasing delay, area and power dissipation.

However, the reduction in programmable switches could severely affect the routing flexibility. Therefore a careful analysis of routing profiles was conducted using a number of circuits placed and routed on a traditional FPGA architecture. The resulting routes of each circuits are analyzed to extract the frequency of hardwired patterns. This analysis is used to create a new architecture with a mix of traditional and hardwired switches. By hardwiring some of the junctions in the routing switches, a 24% reduction in delay, 34% reduction in energy, and a 7% reduction in area were achieved. While this research focuses on improving the routing, it does so trying to maintain routability for any hardware circuit, which leaves room for further optimization such that routability can traded off with further reduction in area, delay, and energy consumption.

Although not directly related to the proposed ASFPGAs, it is interesting to mention Xilinx's EasyPath FPGA design option [Krishnan 2005]. In contrast to a standard FPGA that is guaranteed to work for all hardware designs, an EasyPath FPGA is a low cost design option that is only guaranteed to work a few hardware designs. In [Campregher et al. 2006], an analysis of the yield advantages of using an EasyPath FPGA found that the improvement in yield from such devices is due to the ability to avoid routing paths that have defects significant enough to cause faults.

# 3. FPGA CONFIGURABILITY

In general, an FPGA consists of an array of combinational logic blocks (CLBs) organized into rows ands columns, routing channels running vertically and horizontally between the CLBs, connection blocks connecting the inputs and outputs of CLBs to routing channels, and switch matrices, located at the intersections of vertical and horizontal routing channels, for connecting routing channels together.

Each CLB consists of multiple lookup tables (LUTs), flip-flops, and multiplexors for internal routing among the LUTs and flip-flops within the CLB. The basic configurable logic element within an FPGA is an M-input LUT capable of implementing any combinational logic function with M inputs. To support sequential logic, LUT table outputs can optionally be connected to a flip-flop within the CLB. Thus, the basic configurability of a CLB is controlled by the size of the LUTs and the number of LUTs within the CLB.

While a CLB may have multiple LUTs, the inputs/outputs of a CLB typically do not provide direct access to every input/output of each LUT. Instead, internal routing in the form of multiplexors is provided to allow a smaller number of inputs to connect to the LUTs, as well as connecting the outputs from one LUT to the input of another. Thus, the number of inputs and outputs to and from a CLB can also be configured. For example, a CLB with two 3-input LUTs may have only four inputs – instead of the six necessary to have a unique input for each LUT input.

While CLBs provide the basic configurable resources for implementing combinational and sequential logic, the routing resources of the FPGA are needed to connect the CLBs together to achieve the final circuit implementation. Routing resources include routing channels, connection blocks, and switch matrices. Routing channels are physical wires within the FPGA that run vertically and horizontally between the columns and rows of CLBs. The number of routing channels, often referred to as the channel width, is a configurable parameter that specifies the total number of wires that will run in parallel within the routing channels. In addition, the lengths of the wires – or routing segments – within the routing channels can also be configured. For example, an FPGA may include multiple different routing segments ranging from segments spanning a single CLB to segments spanning an entire row or column. The lengths and frequency at which these routing segments are provided can also be configured.

Connection blocks are utilized to connect the inputs and outputs of CLBs to the routing channels, which can be configured to specify the percentage of routing channels to which each input/output can connect. For example, whereas a connectivity of 100% would allow an input/output of a CLB to connect to any routing channel, a connectivity of 50% would only allow that input/output to connect to half of the available routing channels, where different inputs/outputs within a connection block can connect to different subsets of the available routing channels.

Finally, switch matrices provide the mechanism for connecting the various routing segments within the routing channels together and are incorporated at the intersection of horizontal and vertical routing channels. In addition to connecting routing segments within a single channel, switch matrices also connect routing segments between the horizontal and vertical routing channels. The connectivity of a switch matrix can be configured to determine how many and what type of connections are allowed. For example, a full crossbar switch matrix would allow any routing segment to connect to any other routing segment.

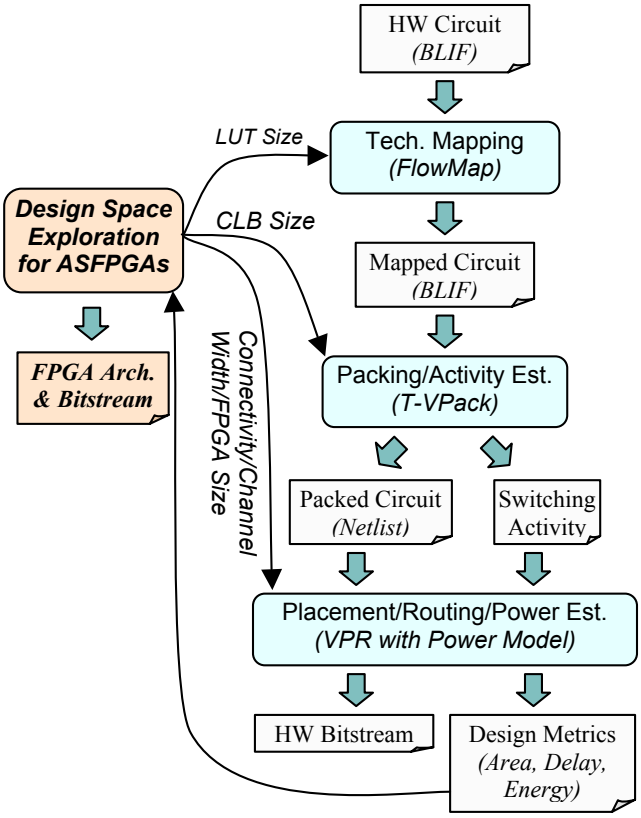
In designing an ASFPGA that will be integrated within a hardware design, the vast configurability of an FPGA design provides an excellent opportunity to tune the configurable architectural features to achieve a smaller, faster, or more energy efficient FPGA architecture.

# 4. DESIGN SPACE EXPLORATION FOR APPLICATION-SPECIFIC FPGA GENERATION

Figure 2 presents our design space exploration environment for application-specific FPGAs originally presented in [Hammerquist and Lysecky 2008]. The input to the design space exploration process is a hardware circuit – specified using the BLIF format – and a set of configurable FPGA architecture features to be considered during the design space exploration. The final output is an optimized ASFPGA architecture and hardware bitstream for the specific hardware circuit. In addition, the resulting ASPFGA can be optimized for area, delay, or energy consumption as specified by the designer. The current ASFPGA design space exploration environment provides an automated exploration framework leveraging existing FPGA CAD tools and supports the following configurable FPGA architectural options:

- *LUT Size*: 3-input, 4-input, or 5-input LUTs
- *CLB Size*: 2 or 4 LUTs per CLB

**Fig. 3:** Design space exploration framework for application-specific FPGAs (ASFPGAs)




- *Connection Block Connectivity*: CLB to routing channel connectivity of 100%, 90%, 80%, 70%, or 60%
- *FPGA Size*: NxN fixed aspect array of CLBs
- *Channel Width*: 100% to 130% of the minimum channel width needed to route input hardware circuit

The ASFPGA design space exploration framework will evaluate the area, delay, and energy consumption of each FPGA architecture alternative to determine the best design given the designers specified optimization criteria. First, for each LUT size, FlowMap [Cong and Ding 2004] is utilized to map the input hardware circuit into the corresponding LUTs. Then we take the mapped hardware and a correction script to add a clock input so that circuits with latches will run through T-VPack properly [Betz et al. 1999]. Next, for each CLB size, T-VPack is used to pack the LUTs into CLBs within the current FPGA architecture configuration. Additionally, Power Model [Poon et al. 2005] is utilized to estimate the switching activity of the mapped and packed hardware circuit needed to estimate the overall energy consumption of the FPGA architecture and hardware circuit. Power Model is an FPGA power estimation model that utilizes activity estimation and transistor level power estimation to estimate the static, logic, routing, and clock power of an FPGA. Next, for each connection block connectivity, each channel width, and each FPGA size, VPR [3] and Power Model are utilized to place and route the hardware circuit onto each FPGA architecture and provide an estimate of the area, delay, and energy consumption of the hardware circuit implemented within the specific FPGA architecture. After evaluating all FPGA architecture alternatives, the final ASFPGA architecture and hardware bitstream are created.

## 4.1 Experimental Results

To evaluate the benefits of ASFPGAs, we consider ten MCNC [Yang 1991] hardware benchmark circuits, including *alu4*, *apex6*, *bigkey*, *cordic*, *des*, *dsip*, *misex1*, *mult32a*, *s1423*, and *s298*. For each hardware circuit, we utilized the ASFPGA design space exploration framework to select the three best ASFPGA architectures for each design criteria, namely area, delay, and energy consumption.

The total area is estimated as the sum of the routing area and combinational logic area for the given FPGA architecture and is reported in minimum sized transistors. While the routing area is directly provided by VPR, VPR does not provide a method for accurately estimating the number of transistors needed to implement the CLBs. Thus, we developed a transistor-level model for CLBs that determines the number of transistors needed to implement LUTs, flip-flops, multiplexers, and configuration SRAM needed within the CLB given the CLB size, LUT size, and inputs/outputs per CLB. The total CLB area is calculated as:

$$Area_{CLB} = Area_{MUXi} + Area_{LUT} + Area_{MUXo} + Area_{FF},$$

where $Area_{MUXi}$ is the area of an input multiplexer, $Area_{LUT}$ is the area of all the LUTs in a CLB, $Area_{MUXo}$ is the area of the output multiplexer, and $Area_{FF}$ is the area of a flip-flop.
The area for an input multiplexor can be estimated as:

$$Area_{MUXi} = \left(20 * N_{LUT} * N_{Cins} * \lceil \log_2 \left(Size_{MUXi}\right) \rceil\right) + \left(6 * N_{LUT} * N_{Cins} * \left(Size_{MUXi} - 1\right)\right),$$

where $N_{LUT}$ is the number of LUTs in the CLB, $N_{Cins}$ is the number of inputs to the CLB, and $Size_{MUXi}$ is the number of inputs to the input multiplexer.

The area for a LUT, is estimated as:

$$Area_{LUT} = N_{LUT} * \left(20 * \left(2^{Size_{LUT}}\right) + 6 * \left(2^{Size_{LUT}} - 1\right)\right),$$

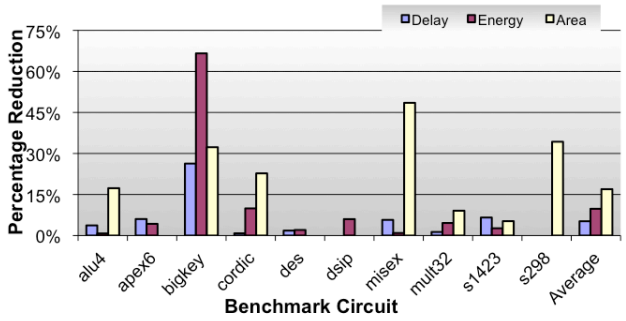
where $Area_{LUT}$ is the area of all the LUTs in a CLB, $N_{LUT}$ is the number of LUTs in the CLB, and $Size_{LUT}$ is the number of inputs to each LUT.

Circuit delay for a hardware circuit implemented within a specific FPGA architecture is the critical path as reported by VPR.

Finally, energy consumption is estimated as the total power consumption reported by Power Model – including static, routing, logic, and clock power consumption – for the hardware circuit implemented within the specific FPGA architecture multiplied by the critical path. We evaluate the FPGA architectural alternatives in terms of energy consumption because the reported power consumption from Power Model is estimated for a hardware circuit executing at the maximum achievable operating frequency, even though the maximum operating frequency may not be needed. Instead, the energy consumption reports the average total energy needed during each clock cycle, which combines the interrelated effects of delay and power consumption.

Application-Specific versus Area/Delay/Energy-Optimized FPGA: While ASFPGAs are optimized for one particular hardware application, commercially available FPGAs must perform well across a broad set of possible hardware circuits. At the same time, many FPGA vendors offer several alternative FPGA devices targeted for specific design criteria, including logic density, speed, and low power consumption. Thus, we compared the area, delay, and energy benefits of an ASFPGA compared to a general area-optimized, delay-optimized, and energy-optimized FPGA architecture. For each hardware circuit, the ASFPGA design space exploration determined the three best FPGA

**Fig. 4:** Comparison of percentage reduction in area, delay, and energy consumption of ASFPGAs versus area-optimized, delay-optimized, and energy-optimized FPGA architectures for several MCNC hardware benchmark circuits.




architectures in terms of area, delay, and energy consumption. In contrast, the general area-optimized, delay-optimized, and energy-optimized FPGA architectures represent the FPGA architecture with the best average area, delay, and energy consumption across all hardware circuits considered. The area-optimized FPGA architecture has 3-input LUTs, 2 LUTs per CLB, and a connection block connectivity of 90%. Both the delay-optimized and energy-optimized FPGA architectures have 5-input LUTs, 4 LUTs per CLB, and a connection block connectivity of 80%.

Figure 3 presents the percentage reduction in area, delay, and energy consumption of ASFPGAs compared to the area-optimized, delay-optimized, and energy-optimized FPGA architectures for several MCNC hardware benchmark circuits. On average, an ASFPGA is 5% faster than the delay-optimized FPGA architecture, 17% smaller than the area-optimized FPGA architecture, or 10% more energy efficient than the energy-optimized FPGA architecture. However, for some applications, the benefits of ASFPGA are much greater. For the hardware circuit *bigkey*, the area, delay, and energy consumption of an ASFPGA is 32%, 26%, and 67% better compared to the area-, delay-, and energy-optimized FPGA architectures, respectively. In addition, the reduction in area of ASFPGAs is greater than 30% for the circuits *misex1* and *s298*. As expected, an ASFPGA performs better that a general FPGA architecture – even when the general FPGA architecture is optimized for the same design metric.

Application-Specific versus Balance-Optimized FPGA: Alternatively, many FPGAs are designed to provide a good balance between area, delay, and energy consumption. Thus, we further compare the three ASFPGA architectures for each hardware circuit to a general balance-optimized FPGA architecture. To determine the general balance-optimized FPGA architecture, we calculated the average area/delay/energy (ADE) cost for each FPGA architecture alternative as the average of the area cost, critical path cost, and energy cost. The ADE cost for each hardware circuit implemented within a specific FPGA architecture is calculated as the ADE for that FPGA architecture divided by the maximum ADE for any FPGA architecture for the current hardware circuit. Thus, the final balance-optimized FPGA architecture is the FPGA with the best average ADE cost across all hardware circuits considered. The resulting balance-optimized FPGA architecture has 5-input LUTs, 2 LUTs per CLB, and a connection block connectivity of 60%. We note that the balanced-optimized FPGA architecture is partially tailored to each hardware

**Fig. 5:** Comparison of percentage reduction in area, delay, and energy consumption ASFPGAs versus a balance-optimized FPGA architecture for several MCNC hardware benchmark circuits.

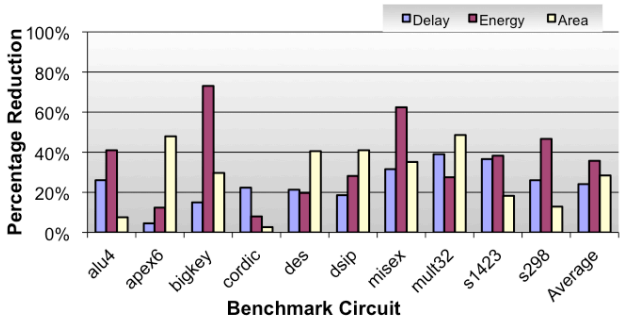
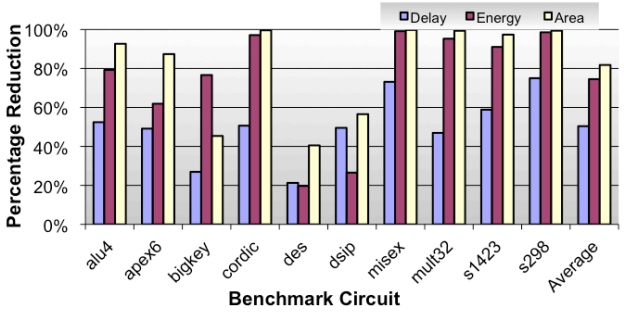



**Fig. 6:** Comparison of percentage reduction in area, delay, and energy consumption of ASFPGAs versus a fixed-size balance-optimized FPGA architecture for several MCNC hardware benchmark circuits.




circuit, as the resulting FPGA design is the smallest FPGA needed to implement each circuit.

Figure 4 presents the percentage reduction in area, delay, and energy consumption of ASFPGA compared to the balance-optimized FPGA architecture for several MCNC hardware benchmark circuits. On average, the ASFPGAs are 25% faster, 28% smaller, or 36% more energy efficient compared to the balance-optimized FPGA architecture, with a maximum reduction of 75%, 99%, and 99%, respectively. The largest improvements provided by an ASFPGA are a 73% and 62% reduction in energy consumption for the circuits *bigkey* and *misex1*, respectively. Furthermore, for eight of the ten hardware circuits, an ASFPGA provides an improvement greater than 40% for at least one design metric.

Application-Specific versus Fixed-Size Balance-Optimized FPGA: As off-the-shelf FPGAs are only available with limited fixed sizes, we further consider a fixed-size balance optimized FPGA architecture. Using the balance-optimized FPGA architecture, the fixed FPGA size was determined as the minimum size needed to support all hardware benchmarks circuit considered. For the ten MCNC hardware circuits, a 63x63 fixed-sized balance-optimized FPGA is needed.

Figure 5 presents the percentage reduction in area, delay, and energy consumption of an ASFPGA compared to the fixed-size balance-optimized FPGA architecture for several MCNC hardware benchmark circuits. Overall, an ASFPGA is on average 50% faster, 82% smaller, or 75% more energy efficient compared to fixed-size balance-optimized FPGA architecture. Compared to

**Fig. 7:** Pseudocode for simulated annealing based application-specific CLB customization.

***CLBCustomization*** *( cluster_count, clusters[], luts[] ):*

```
1.  while( temp > temp_min ) {
2.     if( temp < temp_i*0.25 ) near_end = true;
3.     prev_cluster_count = cluster_count;
4.     for( i in 0  to MovesPerIterations ) {
5.        RandomizedLUTMove( cluster_count, clusters[],
                             luts[], near_end );
6.     }
7.     if( cluster_count <= prev_cluster_count && ValidClustering() ) {
8.          AcceptNewClustering();
9.     }
10.    else if( cluster_count > prev_cluster_count &&
                ValidClustering() ) {
11.       if ( Rand() < temp ) AcceptNewClustering();
12.    }
13.    else {
14.       RejectNewClustering()
15.    }
16.    UpdateClustering()
17.    UpdateTemp()
18. }
```

a fixed-size FPGA, greater savings in area requirements are achieved for those circuits much smaller than the FPGA. At the same time, as the balance-optimized FPGA must be able to balance all design criteria, the ASFPGAs provide area savings of more than 40% for all hardware circuits. In addition, ASFPGAs improve the delay, area, or energy consumption by at least 21%, 27%, and 40%, respectively.

# 5. APPLICATION-SPECIFIC CONFIGURABLE LOGIC BLOCK CUSTOMIZATION

While design space exploration for ASFPGAs provides substantial area, power, and performance benefits, additional opportunities exist for optimizing an FPGA for a specific application. To accommodate an architecture that would further reduce area and enhance the previous design space exploration, we present a configurable logic block customization methodology for ASFPGAs that tailors each CLB within an FPGA to the logic resources of the target application while preserving the traditional routing architecture.

Within a customizable FPGA architecture, a designer can specify the number of LUTs per CLB, the number of CLB inputs, and the connectivity of those inputs to the routing channels. The number of outputs from the CLB is typically equal to the number of LUTs because each CLB can support several independent logic functions implemented within a LUT. With this structure, the number of LUTs per CLB is the defining parameter that directly affects the inputs and outputs to and from a CLB. Thus, a traditional prefabricated FPGA consists of a predefined number of LUTs per CLB. While existing CLB clustering algorithms are able to efficiently pack LUTs into these predefined CLBs, many CLBs will remain unused or underutilized.

The structure of each CLB within an FPGA can also be customized to the required logic components of the target hardware circuit. We present a methodology for generating a highly customized CLB structure in which the number of LUTs per CLB is defined by the application and is only limited by the number of inputs and outputs to and from the CLB. This allows for – and even encourages – grouping together LUTs with shared inputs and grouping together dependent LUTs, in which the output from one LUT is the input to another LUT in the same CLB. The resulting application-specific CLB customization produces a customized FPGA architecture in which unnecessary LUTs and/or CLBs are eliminated. At the same time, the resulting FPGA architecture still adheres to an array style layout that can utilize existing FPGA routing architectures and existing placement and routing algorithms for those architectures. While performance and power benefits may be gained from this methodology, we currently focus on reducing area requirements of the FPGA architecture.

Figure 6 presents the pseudocode of the proposed simulated annealing based *CLBCustomization* algorithm for ASFPGAs. The inputs to the CLB customization algorithm are the number of CLBs in the original clustering, an array of CLBs, and an array of all LUTs. Before executing the CLB customization, an initial valid clustering is created. A valid clustering is a clustering in which the number of inputs to all CLBs is less than the maximum number of inputs allowed and the number of outputs from all CLBs is less than the maximum number of outputs allowed.

Simulated annealing is an optimization scheme that can be used to solve many different types of problems [Fleischer 1995]. The general approach is to start with an initial problem and then transition from one solution state to another, trying to reach a global optimum. Since simulated annealing is analogous to the concept of annealing in thermal dynamics, a temperature parameter is used to decide how long to anneal and how to accept new solution states. An initial temperature value is chosen and then that value is lowered to decrease the probability of transitioning to a new state. Many such *cooling schedules* exist, including linear, quadratic and exponential versions. The idea is to freely allow changes to the solution state in the beginning even if the new state results in an inferior solution so as to avoid getting caught in a local minimum.

Our simulated annealing approach starts with an initial temperature that is lowered exponentially using a cooling schedule defined as:

$$T = T_{Init}\left(\frac{T_{Min.}}{T_{Init}}\right)^{\frac{Iter_i}{Iter_{Total}}}$$

where $T$ is the current temperature, $T_{Init}$ is the initial temperature, $T_{Min}$ is the minimum temperature at which the algorithm terminates, $Iter_i$ is the current simulated annealing iteration, and $Iter_{Total}$ is the total number of iterations we want to update the temperature before terminating.

As our CLB customization algorithm is focused on creating CLB customized for the logic components of the input hardware circuit with the only restriction of the number of inputs and output to and from a CLB, during each simulated annealing iteration a number of randomized moves are utilized to move LUTs between CLBs, specifically including moving a LUT forward to another CLB, moving a LUT backwards to another CLB, and creating a new CLB into which the randomly selected LUT is moved. Figure 7 presents the pseudocode for the *RandomizedLUTMove* operation.

The input to the *RandomizedLUTMove* operation are the number of CLBs in the current clustering, the current array of CLBs, an array of all LUTs, and a flag indicating whether or not the algorithm is nearing the end of execution. One of three different move operations, *ForwardMove*, *BackwardMove*, or *MoveToNewCLB*, is randomly selected, where each move has a specific probability of being executed. First, a random number is utilized to select between three different possible moves, where each move has a different probability of being selected. For all operations, a single randomly selected LUT will be chosen to move.

For the *ForwardMove* operation, the selected LUT's inputs and outputs are utilized to find a CLB farther along in the clustering array that contains a LUT that it is dependent on or that has a LUT dependent on the LUT being moved. The randomly selected LUT will be moved to the first CLB that contains a dependent LUT. For the *BackwardMove*, a similar procedure is followed except that the algorithm looks backwards in the clustering array. The *ForwardMove* and *BackwardMove* operation focus on an arbitrary number of LUTs within CLBs as long as some dependency between the LUTs within a CLB is maintained such that either a decrease in CLB inputs and output can be achieved or any potential increase in the number of inputs and outputs is minimized.

If the *MoveToNewCLB* operation is selected, the randomly selected LUT will be moved to that CLB. The *MoveToNewCLB* operation helps to ensure that the simulated annealing approach does not get caught within local minimum. By allowing new CLBs to be created, the algorithm ensures that the simulated annealing continues to explore new clusterings. However, creating new CLBs is a less desirable operation compared to moving LUTs between existing CLBs – as it results in increased area resources. As such, the probability of creating a new CLB is much lower than the probability of the other LUT moves. The probability of executing the *ForwardMove* and *BackwardMove* operations are equal, whereas the *MoveToNewCLB* operation is 20 times less likely to be executed. Furthermore, to avoid creating additional CLBs near the end of the simulated annealing process, the near end flag is used to indicate the end of the annealing process and further reduces the probability of executing the *MoveToNewCLB* operation by 20%.

During the simulated annealing process, if the new clustering has fewer CLBs than the previous iteration and all CLBs are valid, the new clustering is automatically accepted. Additionally, if the new clustering has the same number of CLBs compared to the previous iteration and all CLBs are valid, the new clustering is again accepted. While keeping the original clustering is also an option, or randomly selecting among the two alternative, we found that always selecting the new clustering achieved better overall results, as the algorithm is allowed to continue exploring new clustering alternatives that may help to avoid a local minimum. If the number of CLBs is greater than the previous iteration and the clustering is still valid, the new clustering will be selected with a probability proportional to the temperature, which decreases exponentially. If the new clustering contains any invalid CLBs – i.e. a CLB with that requires too many inputs or outputs – the new clustering is always rejected.

**Fig. 8:** Pseudocode for Randomized LUT Move operation employed within the simulated annealing based CLB customization algorithm.

***RandomizedLUTMove*** *( cluster_count, clusters[], luts[], near_end ):*

```
1.  move = Rand();
2.  lut = ChooseRandomLUT();
3.  if( move == ForwardMove ){
4.     clb = FindCLBFromLut(lut);
5.     MoveLUT(lut, clb);
6.  }
7.  else if( move == BackwardMove) {
8.     clb = FindCLBToLut(lut);
9.     MoveLUT(lut, clb);
10. }
11. else ( move == MoveToNewCLB ) {
12.    clb = CreateCLB();
13.    MoveLUT(lut, clb);
14. }
```

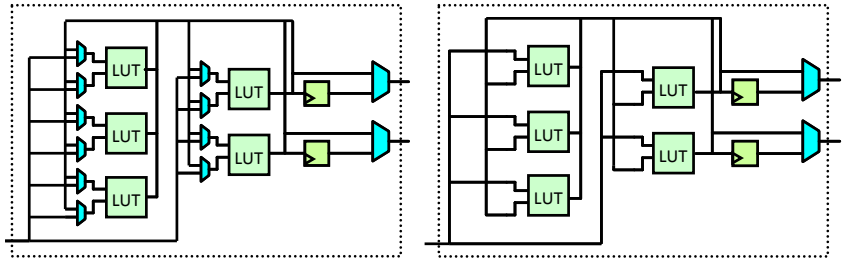
## 5.1 Experimental Results

To evaluate the benefits of CLB customization for ASFPGAs, we consider five MCNC [Yang 1991] hardware benchmark circuits, including *alu4*, *apex6*, *cordic*, *des* and *misex*. For all hardware circuits, our CLB customization algorithm was executed for 250,000 iterations. We note that for the hardware circuits *cordic*, *des*, and *misex1* no further improvements were achieved after the first 10,000 iterations.

While the base LUT structure for which an application is mapped can be tuned within our design space exploration framework for ASFPGAs, we currently assume 2-input LUTs provide the basic configurable component within the target FPGA architecture. FlowMap is utilized to map each hardware benchmark circuits to the corresponding 2-input LUT implementation that is provided as the input to our CLB customization algorithm. After creating an initial, valid clustering using T-VPack, our simulated annealing based CLB customization algorithm was utilized to create a customized CLB architecture for each CLB within the resulting FPGA architecture.

We consider both a *flexible-optimized* and *fully-optimized* CLB customization. As highlighted in Figure 8 (a), a *flexible-optimized* CLB customization produces a CLB structure in which the traditional input multiplexers, flip-flops, and output multiplexers are included. Alternatively, by eliminating all components that are not needed to implement a specific hardware circuit, a *fully-optimized* CLB structure as highlighted in Figure 8 (b) can be further tailored by eliminating these components and using dedicated wires for all connection within the CLB. These two alternatives present the worst case and best case scenarios,

**Fig. 9:** (a) Flexible-optimized CLB architecture customization and (b) fully-optimized CLB architecture customization.




(a) (b)

respectively, for application-specific CLB customization. Note the absence of input multiplexers in the fully-optimized CLB structure. While the fully-optimized CLB customization can greatly reduce area requirements, the flexibility of the resulting ASFPGA to implement any changes in the hardware design is significantly reduced.

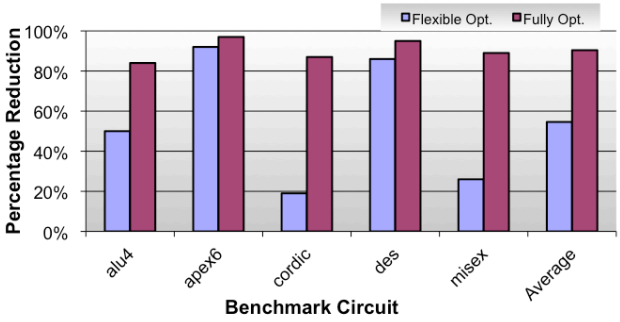
Figure 9 compares the percentage area savings of application-specific CLB customization compared to an area-optimized ASFPGA incorporating CLBs with four 3-input LUT for both a flexible-optimized and fully-optimized CLB customization with a CLB constrained to six inputs and fours outputs. Thus, the number of inputs and outputs for each CLB is equivalent for both the CLB customized ASFPGA architecture and the area-optimized ASFPGA architecture. For the hardware circuits considered, the flexible-optimized application-specific CLB customization achieves an average area savings of 55% over an area-optimized ASFPGA. On the other hand, the flexible-optimized CLB customization achieves a maximum reduction of 94% for the hardware circuit *apex6*.

Because the application-specific CLB customization is not required to follow the regular structure imposed by traditional FPGA architectures, we can eliminate CLBs that are not being used to implement the application as well as eliminate unused inputs and outputs from each CLB. Furthermore, instead of requiring a flip-flop and output multiplexer for each LUT, the resulting ASFPGA only requires one flip-flop and output multiplexer for each output. For the circuit *cordic*, while the resulting CLB customized ASFPGA has more CLBs compared to the area-optimized ASFPGA, the total CLB area is 20% smaller, which is the smallest savings in area achieved by the CLB customization algorithm.

Alternatively, by eliminating all unnecessary configurable elements within all CLBs throughout the resulting FPGA fabric, *fully-optimized application*-specific CLB customization achieves an average area savings of 91% with a maximum reduction in area of 98% for the hardware circuit *apex6*. The largest difference between a flexible-optimized and fully-optimized CLB customization is exhibited for the hardware circuit *cordic*, in which the fully-optimized CLB architecture achieved an additional area savings of 67% over the flexible-optimized CLB architecture.

Overall, our application-specific CLB customization produces a smaller FPGA compared to the design space exploration framework. However, we note that design space exploration and application-specific CLB customization are complementary approaches that can be combined to further optimize the ASFPGA architecture.

**Fig. 10:** Percentage reduction in area of flexible-optimized *(Flexible Opt.)* and fully-optimized *(Fully Opt.)* CLB customized ASFPGA versus an area-optimized ASFPGA for several MCNC hardware benchmark circuits.




## 6. CONCLUSIONS AND FUTURE WORK

By tailoring the architecture of an FPGA to be integrated within a specific hardware application, significant improvements in area, delay, or energy consumption can be achieved. The proposed design space exploration environment for ASFPGAs provides an initial implementation and analysis of these benefits compared to traditional FPGA design alternatives. On average, an ASFPGA optimized for a particular design metric provides a 70% improvement compared to a fixed-size balance-optimized FPGA architecture, with a minimum and maximum improvement of 20% and 99% for specific hardware circuits.

Further reductions in area can be made by customizing the CLBs within an FPGA to the specific needs of the target hardware circuit. The presented application-specific CLB customization achieves further improvements in reducing area requirements, yielding an ASFPGA that is 19% to 94% smaller for a flexible-optimized CLB architecture and 85% to 98% smaller for a fully-optimized CLB architecture.

While this data shows great improvement over standard FPGAs, much future work remains in further exploring the possibilities of application-specific FPGAs. While some automated layout generation tools are available, efficient and robust automated tools are needed to create the physical implementation for the ASFPGA architecture, such that it can be integrated within the target design. In addition, our current approach does not consider fixed functional units, such as adders, multipliers, etc., that can provide significant performance, area, and energy benefits compared to purely logic based FPGA architectures. Additional future work includes exploring the effects of adjusting the number of inputs and outputs for our custom CLB architecture. This exploration would consider a

greatly expanded design space, potentially allowing for greater improvements in area, delay, and energy. Furthermore, as the number of configurable FPGA architectural options continues to grow, the need for efficient design space exploration heuristics becomes increasingly imperative.

# 7. REFERENCES


[1] AKEN'OVA, V., G. LEMIEUX, AND R. SALEH. 2005. An Improved "Soft" eFPGA Design and Implementation Strategy. Proceedings of the IEEE Custom Integrated Circuits Conference, pp. 179-182.

[2] BETZ, V., ROSE, J., AND MARQUARDT, A. Architecture and CAD for Deep-Submicron FPGAs. Kluwer Academic Publishers.

[3] BUCH, K. 2005. Application Specific Programmable Platform Using eASICore. http://www.easic.com.

[4] CAMPREGHER, N., CHEUNG, P., CONSTANTINIDES, G., AND VASILKO, M. 2006. Yield Enhancements of Design-Specific FPGAs. Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA), pp. 93-100.

[5] CONG, J., AND DING, Y. 1994. FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Designs. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems (TCAD), 31(1), pp. 1-12.

[6] DEHON, A. 1996. Reconfigurable Architectures for General-Purpose Computing. Ph.D. Dissertation, Massachusetts Institute of Technology.

[7] FLEISCHER, M. 1995. Simulated Annealing: Past, Present, and Future. Proceedings of Winter Simulation Conference, pp. 155-161.

[8] HAMMERQUIST, M. AND LYSECKY, R. 2008. Design Space Exploration for Application-Specific FPGAs in System-on-a-Chip Designs. Proceedings of the IEEE International SOC Conference (SOCC), pp. 279-282.

[9] HOLLAND, M. AND HAUCK, S. 2004. Automatic Creation of Reconfigurable PALs/PLAs for SoC. Field-Programmable Logic and Applications (FPL), pp. 536-545.

[10] HOLLAND, M., AND HAUCK, S. 2005. Automatic Creation of Domain-Specific Reconfigurable CPLDs for SOC. Proceedings of International Conference on Field Programmable Logic and Applications (FPL), pp. 95-100.

[11] HOLLAND, M., AND HAUCK, S. 2006. Improving Performance and Robustness of Domain-Specific CPLDs. Proceedings of the International Symposium on Field Programmable Gate Arrays, (FPGA), pp. 50-59.

[12] KRISHNAN, G. 2005. Flexibility with EasyPath FPGAs. XCell Journal, Fourth Quarter 2005, pp. 96-98.

[13] KUON, I., EGIER, A., ROSE, J. 2005. Design, Layout and Verification of an FPGA using Automated Tools. Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA), pp. 215-226.

[14] KUON, I., AND ROSE, J. 2006. Measuring the Gap between FPGAs and ASICs. Proceedings of the International Symposium on Field Programmable Gate Arrays (FPGA), pp. 21-30.

[15] LEVINTHAL, A., AND HERVEILLE, R. 2005. FlexASIC Structured Array: A Solution to the DSM Challenge. DesignCon.

[16] PADALIA, K., FUNG, R., BOURGEAULT, M, EGIER, A., AND ROSE, J. 2003. Automatic Transistor and Physical Design of FPGA Tiles from an Architectural Specification. Proceedings of International Symposium on Field Programmable Gate Arrays (FPGA), pp. 164-172.

[17] PHILLIPS, S. 2004. Automating Layout of Reconfigurable Subsystems for Systems-on-a-Chip. Ph.D. Thesis, University of Washington, 2004.

[18] POON, K., WILTON, S., AND YAN, A. 2005. A Detailed Power model for Field-Programmable Gate Arrays. ACM Transactions on Design Automation of Electronic Systems (TODAES), 10 (2), pp. 279-302.

[19] SIVASWAMY, S., WANG, G., ABABEI, C., BAZARGAN, K., KASTNER, R., BOZORGZADEH, E. 2005. HARP: Hard-Wired Routing Pattern FPGAs. Proceedings of the International Symposium on Field-Programmable Gate Arrays (FPGA), pp. 21-29.

[20] S. YANG. 1991. Logic Synthesis and Optimization Benchmarks, Version 3.0. Technical Report, Microelectronics Centre of North Carolina.