Prospectus Abstract:

In the semiconductor industry today, most chips are built for a specific purpose, called application-specific integrated circuits (ASICs), but field-programmable gate arrays (FPGAs) point in the other direction: while ASICs are optimized for their specific applications, FPGAs are designed to be configurable to support user-programmed functions. Unlike ASICs, FPGA designs are configurable without physical changes, and are effective for rapid verification and prototyping of VLSI designs. They are also used in products that require periodic hardware changes and short time to market. However, FPGAs incur penalties in area (17–54x), speed (2.5–6.7x), and power (5.7–62x) over standard-cell ASICs, hindering their expansion into ASIC markets. The overhead is primarily due to interconnects, which account for over 75% of area and delay.

For over 20 years, FPGAs have used 2D-mesh interconnects, where look-up tables (LUTs) are placed in configurable logic blocks (CLBs), and arrays of switch boxes are placed at interconnect crossings. Since a full array requires too much area, various heuristics are used to simplify switch-box arrays at the cost of resource utilization. Yet 80% of the 1.1B transistors on Virtex-5 are used for interconnects. So far, my work has demonstrated an FPGA with hierarchical interconnects where interconnect area is 51%, a 3–4x reduction from commercial FPGAs while preserving connectivity. An energy efficiency of 1.1 GOPS/mW is the highest among reported FPGAs, and is 22x more efficient than the most efficient commercial FPGA today. This work will be scaled to larger FPGAs to further demonstrate the benefit of hierarchical interconnects, incorporating numerous circuit-level and architecture optimizations, and even device-level improvements.
Introduction:

In modern VLSI designs, “efficiency” has become a buzzword. Following the rapid feature-size and frequency scaling in the 1990s, it became clear by the early 2000s that targeting designs purely for performance is reaching the end of the road: power, thermal, and physical constraints of the design is becoming just as important as circuit performance. This trend led to the emphasis on efficiency, for it represents design tradeoffs, balancing performance, power, and area requirements. Energy efficiency is arguably the most common efficiency metric. It quantifies work per unit energy, and is generally measured in billions of operations per milliwatt (GOPS/mW). In VLSI circuits, this translates directly to battery life, thermal limit, and reliability due to electromigration. Area efficiency is also a common criterion, quantifying work per unit of area, generally measured in billions of operations per mm$^2$ (GOPS/mm$^2$). This translates directly to die size, cost, and yield.

It is not difficult to see that energy and area efficiencies contain a mutual tradeoff: a low-power chip may utilize massive parallelism to achieve its required throughput, even at low $V_{DD}$. Such chip will have high energy efficiency, but lower area efficiency due to parallelism. In contrast, a single high-performance core running at nominal $V_{DD}$ is generally less energy efficient, but achieves high area efficiency. To have a fair comparison, it is necessary to evaluate a chip based on both criteria. This leads us to an interesting question, how efficient are today’s chips? Based on the averaged VLSI chips from ISSCC and VLSI conferences in the past decade, we observe a clear trend in Figure 1.

Figure 1: Energy and area efficiency of the ISSCC/VLSI chips from the past decade, averaged.
From Figure 1, we observe a clear 4 orders-of-magnitude difference in efficiency, between micro-processors, FPGAs, programmable DSPs, and dedicated hardware (ASICs). More interestingly, there exists an inverse relationship between efficiency and flexibility. This makes sense intuitively: whenever we take a dedicated portion of a chip and make it programmable, we need to implement additional hardware. In the extreme case of a microprocessor, the actual arithmetic-logic-unit (ALU) is only a small portion of the entire chip, leading to its low energy and area efficiencies.

Today's semiconductor industry has a strong push for efficiency, so not surprisingly, most chips today are built for a dedicated purpose. However, what if we need programmability? In the example of a smartphone, the chip is required to run an operating system, and perform different multimedia tasks on-demand, while maintaining communications through the digital-front-end. At the same time, the smartphone must have good battery life. This calls for both efficiency and programmability, but how can we have both? The current solution is to implement both of them, with microprocessors and dedicated accelerators on the same chip.

As shown in Figure 2, the NVIDIA Tegra 2 is marketed as a 1 GHz dual-core processor for Android phones. However, we see that the actual processor only occupies a small fraction of the total chip area. The rest of the chip are mostly dedicated accelerators, and the dedicated digital-front-end (DFE) is performed on another ASIC. This hybrid approach exists in almost every smartphone today, but it is not a fully scalable solution. First, this approach leads to large portions of “dark silicon”, which reduces area efficiency. As smartphones are required to do more multimedia tasks, the accelerators would need to increase even more. Second, and more importantly, the accelerators are designed for fixed standards. This means whenever a communication or multimedia standard needs to be changed or updated, even for just one block, a redesign of the entire silicon is required.

Unfortunately, standards for communication and multimedia are constantly updating, and are updating at an accelerated pace (Figure 3). As a result, the chips often need to “keep up”
with the standards, and it is common for ASIC chips to be re-designed at least once a year, even though most of the blocks on the chip are legacy designs.

**Media Standards**

![Media Standards Diagram]

**Radio Standards**

![Radio Standards Diagram]

Figure 3: Evolution of common multimedia and radio standards.

The constant redesign of ASICs are expensive, and are becoming increasingly so with every technology generation. As shown in Figure 4, the increasing design complexity, increasing man-hours per design, expensive CAD licenses, and higher fabrication costs are driving up total cost at an accelerated pace. In the very near future, we will no longer be able to afford re-designing chips on an annual basis. We would need to have reconfigurable hardware, one that can efficiently perform the tasks that dedicated hardware is now performing.

**Total Development Cost ($M)**

![Total Development Cost Graph]

Figure 4: Cost of chip design with every technology node.

Apart from micro-processors, FPGAs and programmable DSPs are possible candidates for reconfiguration. However, programmable DSPs are software controlled, and it is difficult for them to execute bit-true, cycle-true behaviors of a dedicated hardware. In addition, they generally have lower throughput unless massively-parallel (such as single-instruction-multiple-data (SIMD)) architectures are used. But in the case of a SIMD, all cores must perform the identical function, which is often not the behavior of a dedicated hardware. On the other hand,
FPGAs are designed to replace dedicated hardware in a bit-true, cycle-true manner. They have high throughput due to the implicitly parallel, independent logic blocks, just like ASICs.

Comparing ASIC design versus FPGA design (Table 1), FPGA is really appealing from a design perspective, especially for design verifications and prototypes. With supporting software, the user designs can be mapped to run on the FPGA in a matter of hours or less. Comparing to the expensive process of metallization rework or chip fabrication, the software reconfigurability of FPGAs placed it in another league. For many VLSI designers, the ability to rapidly verify design changes in silicon without any fabrication rework is an indispensable tool. In many designs that require constant changes, or for small companies that cannot afford to support an entire physical design team (followed by a million-dollar fabrication process), FPGA is becoming used in end-products as well. But for FPGA to be implemented in a mass-consumer market, especially in power-constraint environments such as smartphones, the inefficiency of FPGA operations must be corrected. We need to bridge the 100x gap in efficiency between FPGAs and ASICs, else our cellphone battery will last just a few minutes!

<table>
<thead>
<tr>
<th>Efficiency (ASIC)</th>
<th>vs.</th>
<th>Flexibility (FPGA)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Logic Design, Physical Design</td>
<td>Logic Design only</td>
<td></td>
</tr>
<tr>
<td>Licenses: Synthesis, P&amp;R, etc.</td>
<td>Fewer Licenses</td>
<td></td>
</tr>
<tr>
<td>90nm or older</td>
<td>32nm or newer</td>
<td></td>
</tr>
<tr>
<td>2 – 4 months fabrication time</td>
<td>None</td>
<td></td>
</tr>
<tr>
<td>Expensive to Design</td>
<td>Inexpensive to Design</td>
<td></td>
</tr>
<tr>
<td>Efficient Operation</td>
<td>Inefficient Operation</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: ASIC vs. FPGA – efficiency vs. flexibility.

The focus of this work is to build energy-efficient, reconfigurable hardware. We first identify the source of inefficiency from today’s FPGAs, and propose to use a hierarchical interconnect architecture to reduce the interconnect area to 51%. This 1:1 ratio of interconnect to logic area is a 3–4x reduction from commercial FPGAs. As a proof of concept, our current chip is designed with standard cells, and is tested up to 400MHz. Among the tested applications, peak area efficiency of 16.4 GOPS/mm² is achieved at 370MHz, consuming 179mW at 1V. Peak energy efficiency of 1.1 GOPS/mW is measured at 0.5V, 55MHz. A software place-and-route tool is developed to create bitstreams used to configure the FPGA. This work will be scaled to larger FPGAs to further demonstrate the benefit of hierarchical interconnects, incorporating numerous circuit-level and architecture optimizations, as well as improved software support.
The 2D-mesh interconnect architecture:

The concept of a programmable hardware started over 30 years ago, but it was regarded as prohibitively expensive because of its large overhead in area over ASICs. Transistors were expensive, and no one wanted to pay more than a 5x penalty in area just for reconfigurability. Fortunately, the semiconductor industry rapidly expanded at the pace of Moore’s law, and such large area overhead became more tolerable, finally leading to a first FPGA by Xilinx Corporation in 1985. The original FPGA, XC2000 series, had 64 or 128 look-up-tables (LUTs). Although the initial perceptions were “small, slow, expensive, and ‘different’”, the XC3000 introduced in 1987 was very successful even with very rudimentary software support. In contrast, modern FPGAs have evolved to include more than 300,000 blocks with a plethora of interface blocks, combined with seamless software integration to support the majority of ASIC functionalities today.

Although the cost-effectiveness of FPGAs is apparent, FPGAs incur penalties in area (17–54x), speed (2.5–6.7x), and power (5.7–62x) compared to ASIC designs. Such overheads are still prohibitively high for many ASIC designers to adopt an FPGA design. This is primarily caused by interconnects, which account for over 75% of area and delay. Although FPGAs sizes have grown enormously since the XC2000, the fundamental 2D-mesh architecture still remains. In 2D-mesh interconnects, LUTs are placed in configurable logic blocks (CLBs), and arrays of switch boxes are placed at interconnect crossings (Figure 5).

Figure 5: A 2D-mesh architecture with I/O connections and switch boxes.

Each switch-box contains pass-transistors programmable by the configuration memory. Since a full switch-box array at every interconnect crossing requires too much area, various
heuristics are used to simplify the arrays at the cost of interconnect connectivity, which affects logic utilization. Even with such heuristics, 80% of the 1.1 billion transistors on Virtex-5 are still used for interconnects. Although the name FPGA implies a chip of programmable logic, the interconnect area is actually 4x the logic area! (Figure 6) In addition, interconnect also accounts for the majority of the delay and power in today’s FPGA.

![Area, delay and power breakdown of a 2D-mesh FPGA.](image)

Figure 6: Area, delay and power breakdown of a 2D-mesh FPGA.

The key cause for interconnect overhead is the scalability of 2D-mesh interconnects. In the worst case, the number of switch boxes grows as $O(N^2)$ with the number of LUTs. Using Rent’s rule to model typical cases, the interconnect complexity per logic gate is $O(N^{0.75})$ for random logic, which is still $O(N^{1.75})$ for a chip of $N$ logic gates. Such complexity requires FPGA sizes to scale much faster than Moore’s Law. Scaling $N$ from 64 in XC2000 to 300,000 in modern FPGAs, it becomes clear why interconnect area is a key concern today.

In attempts to mitigate the area impact of 2D-mesh networks, a hierarchical network based on “tree of meshes” is previously proposed (Figure 7). It is a limited bisection network, where the mesh connectivity decreases for upper hierarchies, and even connectivity at local levels are limited. Additionally, a centralized routing network is required at every hierarchy, which increases routing congestion, and central switches are still based on 2D-mesh. In this work, we propose a fully hierarchical architecture using no mesh networks, and no centralized switching.

![A tree-of-meshes network with limited bisection.](image)

Figure 7: A tree-of-meshes network with limited bisection.
A hierarchical interconnect architecture:

In this work, a hierarchical interconnect architecture based on the Beneš network is proposed. To distinguish from 2D-mesh switch boxes, we name the hierarchical interconnect blocks switch matrices (SMs). As shown in Figure 8, 4 LUTs are connected via 2 stages of SM, and another 4 LUTs are to be connected with a 3rd SM stage. This effectively reduces the interconnect complexity to $O(N \cdot \log N)$.

![Figure 8: A hierarchical interconnect architecture using folded Beneš network.](image)

Each SM has 4 unidirectional connections in both the upward and downward direction. The signal will come from the LUT output, traverse up to the required hierarchy, and U-turn back down to the LUT input. Although this architecture reduces interconnect complexity, each additional SM stage results in double the routing congestion. This $O(N)$ congestion requires larger area for higher level SMs, making physical design more difficult and less area-efficient.

To alleviate congestion, routing is alternated between the x-y directions to reduce congestion to $O(N^{0.5})$ (Figure 9). Unlike tree-of-mesh interconnects, where a centralized switching network functions as a “hub”, the interconnect routing is evenly distributed across all LUTs. At every hierarchy, the LUTs near the center are connected to create shorter routes, and the LUTs near the edges have longer routes. This gives routing tools options for faster paths on timing-critical routes. In physical design, this also allows the center routes to remain at the lower metal layer without crossing over the longer routes, further avoiding congestion.
In this work, an FPGA with 2048 LUTs is realized using hierarchical interconnects. The interconnect area is reduced to 51%. This 1:1 ratio of interconnect to logic area is a 3–4x reduction from commercial FPGAs. Some details of the test chip are highlighted in the next section.

**Our recent chip, a 2048-LUT FPGA:**

The test chip has 2048 LUTs, requiring 11 levels of interconnect. Since every level translates to one SM stage, 11 levels of SMs are required, although the 11th stage is just a simple U-turn. To ensure 100% connectivity in all cases, every LUT needs to have 11 levels of SM to preserve the full Beneš network. Even then, we only guarantee it to be rearrangeably-nonblocking for point-to-point (unicast) connections. A multi-cast connection would require double Beneš network or more, which is impractical to implement.

In practical designs, however, the majority of the logic connections are local. We then apply this logic locality to implement full connectivity only for the lower levels, and prune the connections on upper hierarchies, thus creating a limited-bisection network. Full connectivity is preserved for every 128 LUTs, or 6 SM stages (Figure 10), then half-connectivity SMs are used to reduce the complexity of upper hierarchies. The half-SMs allow 2 out of 4 inputs to propagate upward, so two half-SMs reduce the top-level connectivity to 25%. The interconnect network is partitioned into three sub-networks: $N_{8:2}$, $N_{6:2}$, and $N_{6:1}$, where $N_{P:Q}$ represent a network of $P$ full-SMs and $Q$ half-SMs, respectively.
This partitioned architecture also facilitates physical design. A hierarchical physical design is used, starting from the individual CLBs: Logic, DSP, or BRAM, combined with switch matrices. The CLBs are then placed into macros of 32 CLBs, where each macro is a combination of Logic, DSP, and BRAM. The chip is divided into 16 macros of 128 LUT each (Figure 11, left): macros with N_{8:2} interconnects are placed near the center for shorter top-level routing, branching into N_{6:2} and N_{6:1}. This physical placement avoids long wires at the top level, and therefore minimizes interconnect buffers and further reduces area.

The test chip has 2048 4-input LUTs. To map a variety of designs, the resource allocation is as follows: 1024 LUTs are logic-only, 896 LUTs are configurable for logic or arithmetic functions, and 128 LUTs are used for block RAMs (BRAMs). These LUTs are grouped into configurable logic blocks (CLBs): 4 logic-only LUTs form a Logic CLB, 4 arithmetic LUTs form a DSP CLB, and 8 BRAM LUTs form a 1kb, dual-port BRAM CLB. Since some DSP and BRAM operations require many input bits, this grouping allows input-sharing between the LUTs within a CLB. The resource placement of the macros is shown in Figure 11 (right).
Four LUTs are combined with intermediate arithmetic blocks to form a Logic or DSP CLB. Each Logic CLB is composed of four 4-input look-up-tables (4-LUTs), a carry chain, and 4 parallel synchronous/asynchronous output stages (Figure 12a). Each LUT is configurable as one 4-input LUT, or two 3-input LUTs with up to 4 unique inputs. The carry chain supports 4b additions where Propagate and Generate are driven from LUTs. Since each output stage support two outputs, the Logic CLB is especially useful when two outputs per bit are required, such as in 3:2 compressors.

The DSP CLBs are a more flexible for intensive arithmetic computing demands. Each DSP CLB is composed of four 4-LUTs, a LUT combiner, a partial product generator, a configurable adder tree, and 4 parallel output stages (Figure 12b). The detailed block diagram is shown in Figure 13. Each 4-LUT is able to perform one 4-input logic, two 3-input logic by sharing two common inputs, or functions with 5/6 inputs by combining LUTs. Two 4b ripple-carry adders can perform two separate 4b, one 8b, or one 3-operand 4b addition with the support of 3:2 compressors (built from LUTs). The Wallace-tree multiplier reuses the adder cells, and uses dedicated partial-product generators. Overall, the CLB has the flexibility to support 10 operating modes, which includes (i) random logic with 3 to 6 inputs; (ii) 4 to 8b addition/subtraction for 2 to 3 operands; and (iii) 4b×4b signed/unsigned multiplication. To achieve this degree of configurability, the synthesized CLB has 50 gates on its critical path (shaded in Figure 13), amounting to a 1.1ns delay.
To control CLBs and SMs, configuration bits are distributed over all programmable elements, and their outputs must be accessible at all times during chip operation. Traditional SRAM is not suitable because all bits cannot be accessed simultaneously, and an address decoder is required. A scan-chain approach is another alternative, as adopted in Intel's 32nm DSP CLB; this was feasible for 6 CLBs, but is not scalable to larger designs. For our design, the area overhead from scan chain is very high because of the large number of configuration bits. An area-efficient configuration memory array with 6T bit-cells (BCs) is thus proposed, where the output of each BC is directly routed to SMs and CLBs for configuration.
Figure 14 shows the configuration circuits and the control mechanism. The bit-line control signals (BL+ and BL-) of each CLB are generated from a 14b scan chain for every column of BCs. Once all configuration bits are shifted into the scan chain, B_{EN} stops the shifting and activates the word-line (WL) control circuit. The writing operation has two phases: (i) write-enable (W_{EN}) shifts the WL scan chain forward for one clock cycle; (ii) write-evaluate (W_{EV}) asserts the writing signal (W_{IN}) via AND gates to program one row of BCs. The area of a custom bit-cell is 2.52μm^2, which is 5x smaller than a DFF-based scan cell from the standard-cell library. Overall, the memory area is reduced (Figure 15), and the total interconnect area is 51%, a 3–4x reduction compared to 2D-mesh for a fixed logic area.

An automated mapper is developed to map RTL onto this FPGA. A standard-cell library of LUT functions is created to enable logic synthesis using commercial tools. The LUT netlist is imported into an automated, custom place-and-route tool that generates the bitstream for FPGA programming. This tool is also used during architecture design to evaluate interconnect
connectivity by mapping Toronto20 benchmarks. The diagram mapping and verification flow is shown in Figure 16.

Figure 16: Overview of the FPGA mapping flow using IBOB/MATLAB interface.

ASIC verification is performed with the use of a custom "IBOB" (Interconnect Break-Out-Board) board with a commercial FPGA for pattern generation and data analysis. Configuration bits and test vectors are stored in block RAMs on the IBOB through the MATLAB interface. The IBOB is then initiated to stimulate the ASIC over two high-speed Z-DOK+ connectors. Either an internal or an external clock source can be used to provide flexibility and a wide range of operating frequencies. The outputs of the chip are captured into block RAMs on the IBOB for analysis. The I/O interface between the IBOB and MATLAB is built on the BEE-XPS environment developed by UC Berkeley and CASPER. The real-time operation and easy programmability of the IBOB simplifies the testing and data analysis process.

Our chip achieves 16.4 GOPS/mm² when all Logic and DSP CLBs are utilized, executing 175 16b accumulators at 370MHz. Since a 16b adder can be implemented with 2 DSP CLBs but requires 4 Logic CLBs, the DSP adders are faster, reaching 400MHz. Performance is hindered by equipment limitations due to a 0.25ns input-clock jitter at 400MHz. The energy-delay curve and the power breakdowns for minimum delay and minimum energy are shown in Figure 17.
In comparison, Intel’s work [A. Agarwal] has no interconnects and lower energy efficiency, though the full-custom CLB in 32-nm LVT is 2.5x faster. It achieves 2.6 GOPS/mW at 0.34V for 8b operations, which is 0.65 GOPS/mW for 16b (2 CLBs per operation at half the speed). With interconnects, our 65-nm chip reaches 1.1 GOPS/mW at 0.5V. Among commercial FPGAs, the highest-reported efficiency is 0.05 GOPS/mW from an Altera Stratix IV [A. George]. In comparison, our chip is 22x more energy efficient (Figure 18).

Leakage is well-controlled even without power gating. A 1.08 GOPS/mW is attainable with only 112 DSP accumulators active and most of the Logic CLBs idle (Table 2). The FIR filter achieves 274MHz due to longer routing, but interconnect delay is still under 50%. The 2×2 MIMO FFT uses 10 BRAMs to implement various delay lines. With many control signals and a critical path of 11 CLBs, the FFT achieves 83MHz.

Figure 17: Energy-delay curve and power breakdown of our FPGA.

Figure 18: Overview of the FPGA mapping flow using IBOB/MATLAB interface.
Next steps and projected work:

With our recent 2048 LUT FPGA, we have achieved more than 1 order of magnitude of improvements in energy efficiency over commercial FPGAs. With the benefit of a hierarchical interconnect architecture, we are able to achieve near-ASIC energy and area efficiencies. Moving forward, we projected work will further close the gap between reconfigurable hardware and dedicated chips (Figure 19).

![Figure 19: The efficiencies of our prior work (2048-LUT FPGA) and projected work.](image)

It is clear that the efficiencies of reconfigurable hardware can never exceed that of dedicated chips. Our current gap is less than one order of magnitude away. To close in on the final gap, we propose various changes in the architecture, circuit and (potentially) device level of our FPGA.

There are many applications that require > 1 GOPS/mW of energy efficiency, just to name a few: the communication DSP in cellular phones, the multimedia DSP in many portable devices,
and even the neural DSP in implanted neural recordings. In these applications, high data throughput and long battery life must co-exist (one day for smartphone, but neural recordings may last days or months). Even though these designs are evolving, there are many core blocks that always remains. For example, a communication DSP usually requires a Fast-Fourier Transform and complex multiplier, a multimedia DSP usually requires discrete-cosine-transform and wavelet transform, and a neural DSP usually uses wavelet transforms and FIR/IIR filters.

Our FPGA is far more efficient than commercial FPGA today, but a FPGA can only maintain its efficiency when it is mapping designs that effectively utilize its architecture. Based on our experience in mapping designs onto our FPGA, computation-intensive designs such as finite-impulse-response (FIR) filters and arithmetic blocks such as parallel accumulators are highly efficient when mapped onto our FPGA. However, control-heavy designs such as Fast-Fourier Transform (FFT) cannot be mapped very efficiently. As shown in Table 2, the measured energy efficiency of a FFT is up to 16x lower than the maximum efficiency of our FPGA.

Fortunately, because many of these inefficient blocks are core blocks, we can implement some of these designs as coarse-grain accelerator cores on our FPGA (instead of using the fine-grained DSP CLB). These accelerators need to be programmable, because parameters such as FFT radix (8-2048 point), datapath wordlength (8-16 bits) and pipeline stages (throughput-dependent) are usually different amongst different designs. Even though programmable accelerators are still not as efficient as fully dedicated chips, embedding them on the FPGA enables us to push the efficiency gap much closer.

Another change in the CLB level is to merge the Logic and DSP CLBs. From our previous experience, there were two issues to be addressed: 1) the critical-path of an FPGA is usually dominated by LUTs, for there can be many LUT logic on the critical path, but it is rare to have back-to-back DSP logic on the critical path; 2) in high-density designs, sometimes the tool is unable to route to the desired CLB, and that creates an issue: if a LUT is to be mapped, then using a DSP CLB for LUT will be slower, but if a DSP is to be mapped, then a LUT CLB cannot be used. By merging the Logic and DSP CLB together, we can avoid this problem altogether. The proposed CLB architecture is shown in Figure 20. The LUT outputs (lutA, B, C, and D) can be bypassed to the final output mux, enabling the shortest datapath (Figure 20, orange). This is because the critical path of a signal can have multiple combinatorial blocks in series, thus requiring multiple LUT stages (let us assume \(N\) stages). Because only the final stage of the datapath will drive a flip-flop, the remaining \(N-1\) LUTs can benefit from the faster bypass mux. In addition, the CLB will support two clock domains, and allow clock gating when no flip-flop outputs are required (Figure 20, red). The LUT portion and the DSP portion of each CLB may be power-gated independently to reduce leakage.
Although CLB improvements are essential, the most important aspects of our FPGA is the interconnect network. In the previous 2048-LUT FPGA chip (Figure 10), the architecture was optimized manually, and two types of SMs are utilized. Now, we have developed the software tool to map designs onto our architecture, which allows us to explore the optimal interconnect architecture. Since there is no theoretical method to calculate the necessary connectivity at every level of the hierarchy, we can only perform the exploration by mapping benchmark designs and commonly-used designs for each application. Figure 21 shows an optimized architecture with 5 types of SMs. This highly heterogeneous SM architecture allows sufficient connectivity for all of our mapped designs.
For interconnect optimizations, we are adopting a top-down approach: an optimized interconnect architecture can make drastic differences in performance and efficiency. Afterwards, we implement circuit-level changes, and if possible, new devices can be utilized.

Switch matrices (SMs) are the building block of the interconnect network, not only are the SMs replicated throughout the chip, it is common for a signal to pass through 10 or even 20 SMs between CLBs. Therefore a small increase SM delay will quickly manifest itself into a large increase in delay, directly affecting performance. For physical design, we plan to implement a static mux to replace traditional muxes. As shown in Figure 22a), a traditional 4-input mux requires a concatenation of two stages, each controlled by one bitcell (BC). This requires two stages of pass-transistors, for each mux requires one pass-transistor. Since the mux switches of SMs arrive from BCs, they are static throughout the operation of the chip. We can therefore replace 2 BCs with 4 BCs, each driving just one pass-transistor (Figure 22b). This reduces the on-resistance of the mux by 2x, at the cost of more BCs. But since each BC is just a 5T SRAM cell, the area impact is small in comparison to the performance gains. The migration from a 6T bitcell to a 5T obtains a significant savings in area (Figure 23). We were also able to free up more routing tracks on the M2 layer (the new cell occupies 1 M2 track instead of 4).

Figure 22: A 4-input mux implemented in a) standard cells, and b) static mux.

Figure 23: The migration from 6T to 5T bitcell (BC) for area savings.
In high-speed pass-transistor design, 1 inverter can usually drive 2 pass-transistors before re-buffering, we therefore propose to interleave buffered SM with unbuffered SM to achieve this 1:2 ratio. The buffered SMs will have a CMOS inverter at its output, while the unbuffered SM will be purely a pass-transistor mux.

Extending from the concept of static mux, we observed that only one pass-transistor may remain on, and that is determined by the BC values. Since each BC is essentially a SRAM cell, keeping the BC VDD at a higher voltage will lower the on-resistance of the pass-transistors. This is especially useful when VDDL, the logic VDD, is scaled down. In this scenario, total energy will decrease as $C \cdot (VDDL)^2$, just like normal circuits, but system performance will not degrade as much, because the on-resistance of the pass-transistors remains constant. Because the voltage of the BCs ($VDD_H$) remains at 1.2V, high-voltage-threshold (HVT) transistors (Figure 24, red) are required to minimize BC leakage power.

![Figure 24: A bitcell (BC) and a custom 4-input mux for dual-VDD operation.](image)

As a final part of our design effort, we plan to incorporate micro-electro-mechanical relays as switches for the interconnect pass-transistors (Figure 25a). A single relay is able to function as a “NMOS” and a “PMOS” depending on its body voltage. Although they are considered slow due to their long mechanical delay (to physically open and close the gate), such delay is irrelevant for our static mux. We can, however, greatly benefit from its low on resistance ($< 1k\Omega$), which remains constant throughout our regions of operation.

![Figure 25: Diagram of a) relay-based mux and b) its physical integration with CMOS.](image)
Another benefit of MEM relays is its simplicity of fabrication: no silicon substrate is required, and fabrication can be done with just 2 metal layers. Therefore, relays can even be “stacked” on top of CMOS designs. Ideally, the CLB and core logic can be implemented in CMOS, and the interconnect can be stacked on the relays layer, effectively cutting chip area by 2x! We have already designed relays-based FPGA interconnects (currently in fabrication), and this seemingly far-fetched idea may not be that far into the future!

**Conclusion:**

In modern VLSI designs, it is clear that an efficient, reconfigurable hardware will bridge the large gap between silicon cost, design time, and efficiency. To target the high cost of ASIC design and the inefficiency of FPGAs, a hierarchical interconnect architecture for efficient reconfigurable hardware is proposed. Based on our prior work, we have demonstrate a 2048-LUT FPGA that is 22x more energy efficient than the most efficient commercial FPGAs today. Based on our demonstrated results, we propose to incorporate reprogrammable DSP cores, a homogeneous DSP CLB, a highly-optimized interconnect architecture, and custom-designed circuits to further bridge the efficiency gap between reconfigurable and dedicated hardware. Reconfigurable hardware will never be as efficient as ASICs, but given a small enough penalty in efficiency and performance, we are convinced that the reduced time and cost of employing a reconfigurable hardware will make a strong candidate for ASIC replacement.