Architecture-Specific Packing for Virtex-5 FPGAs

Taneem Ahmed[‡], Paul D. Kundarewich[‡], Jason H. Anderson[‡], Brad L. Taylor[†], and Rajat Aggarwal[†]

[‡]Xilinx, Inc., Toronto, ON, Canada
[†]Xilinx, Inc., San Jose, CA, USA
{taneema | paulku | janders | blt | rajata}@xilinx.com

ABSTRACT

We consider packing in the commercial FPGA context and examine the speed, performance and power trade-offs associated with packing in a state-of-the art FPGA – the Xilinx® Virtex™-5 FPGA. Two aspects of packing are discussed: 1) packing for general logic blocks, and 2) packing for large IP blocks. Virtex-5 logic blocks contain dual-output 6-input look-up-tables (LUTs). Such LUTs can implement any single logic function requiring no more than 6 inputs, or any two logic functions requiring no more than 5 distinct inputs. The second LUT output is associated with slower speed, and therefore, must be used judiciously. We present placement-based techniques for dual-output LUT packing that lead to improved area-efficiency and power, with minimal performance degradation. We then move on to address packing for large IP blocks, specifically, block RAMs and DSPs. We present a packing optimization that is widely applicable in DSP designs that leads to significantly improved design performance.

Categories and Subject Descriptors

B.7 [**Integrated Circuits**]: Design Aids

General Terms

Design, Algorithms

Keywords

Field-programmable gate arrays, FPGAs, optimization, packing, placement, power, performance

1. INTRODUCTION

Modern field-programmable gate arrays (FPGAs) consist of a regular fabric of generic logic blocks and specialized hard IP blocks. Clusters of look-up tables (LUTs) and flip-flops form the basis for logic blocks in today's FPGAs. For example, a SLICE in the Xilinx Virtex-5 FPGA contains four 6-input LUTs and four flip-flops, as well as arithmetic and other circuitry [2]. The programmability of FPGAs makes them both slower and less area-efficient than custom ASICs for implementing a given logic circuit [11]. One way PLD vendors have addressed such weaknesses is to incorporate ASIC-like hard IP blocks directly into the FPGA fabric. The hard IP blocks in state-of-the-art FPGAs are quite varied, and include block RAMs, digital signal processing (DSP) blocks, analog-to-digital converters, specialized high-speed I/Os and even entire microprocessors.

Packing (also known as clustering) is a crucial step of the FPGA CAD flow that traditionally falls between technology mapping and placement, but that in modern flows, is tightly integrated with placement, technology mapping or synthesis. In essence, packing is the step wherein the elements of the technology mapped circuit are packed into the available FPGA hardware resources. Most commonly, we think of the packing step as packing a design's LUTs and flip-flops together to form logic blocks. However, in commercial FPGAs, the definition of packing is considerably broader, and includes packing design elements into the hard IP blocks.

Packing is well-studied in the literature for the "academic" FPGA model, where logic blocks consist solely of tightly connected clusters of LUTs and flip-flops [4]. To our knowledge, however, no prior work has been published on packing for commercial FPGAs, which have significantly more complex hardware structures.

Commercial FPGAs present unique challenges and optimization opportunities for packing and this paper addresses two aspects of packing for the 65nm Virtex-5 FPGA family. First, we describe how placement-based packing can be used to leverage the dual-output 6-input LUT available in Virtex-5 logic blocks, with the aim of improving both area-efficiency and power. LUTs in Virtex-5 FPGAs can implement any single function with at most 6 inputs, or any two functions that together need at most 5 inputs. Our results show that using both LUT outputs improves logic density and may shorten wirelengths, thereby reducing capacitance. However, as the second LUT output has lower speed, careful handling is needed to maintain overall design performance. We also consider the packing problem for the large IP blocks in Virtex-5 FPGAs: block RAMs and DSPs. We describe a simple block RAM packing technique that considerably improves design performance for a frequently-occurring design configuration, pervasive in many DSP-oriented circuits, namely, FIR filters.

With this work, we illustrate the benefit of taking advantage of certain architectural features present in Virtex-5

FPGA'08, February 24-26, 2008, Monterey, California, USA.

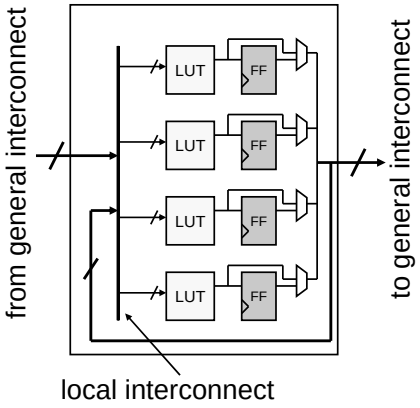

Figure 1: FPGA logic block.

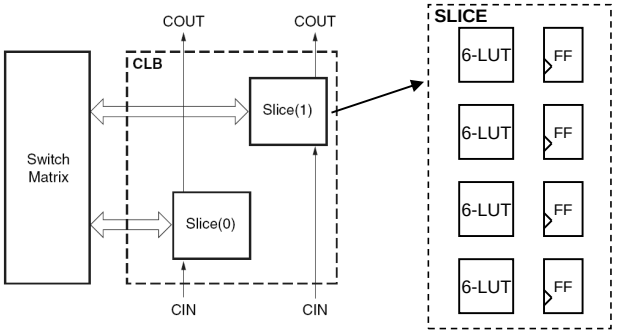

Figure 2: Virtex-5 CLB and SLICE.

FPGAs. We also hope to encourage the academic community to investigate packing algorithms targeting the hardware structures present in modern FPGAs. The remainder of the paper is organized as follows. Section 2 reviews prior work on packing and provides relevant details on the Virtex-5 FPGA architecture. Placement-based logic block packing techniques for area-efficiency and power are offered in Section 3. Section 4 discusses packing for block RAMs in DSP designs. Conclusions and suggestions for future work are given in Section 5.

2. BACKGROUND

2.1 Packing

Fig. 1 shows the logic block model assumed by prior work on FPGA packing. It comprises a cluster of LUTs and flip-flops, where each flip-flop can be optionally bypassed for implementing combinational logic. Inputs to the logic block come from the FPGA's general interconnect: the horizontal and vertical channels of FPGA routing. Local interconnect inside the logic block is available for realizing fast paths within the logic block. Observe that each LUT/FF pair drives both local interconnect, as well as general interconnect.

Packing techniques for the model of Fig. 1 have been extensively studied. An early work by Betz and Rose proposed an area-driven packing algorithm and showed that, because of inherent locality in circuits, the number of inputs to a logic block can be much smaller than the total number of LUT inputs within a cluster [4]. Marquardt extended the work to perform timing-driven packing and demonstrated the impact of packing on critical path delay [13].

An approach for power-driven packing was presented in [16]. Rent's rule was used to establish a preference for how many logic block inputs should be used during packing, leading to lower overall interconnect usage, capacitance and power. Packing for dual-V_{DD} FPGAs was presented in [5], where the FPGA model permits logic blocks to optionally operate at reduced supply voltage (slower). The aim of packing, therefore, is to pack LUTs based on their timing-criticality, placing non-critical LUTs together into logic blocks that will be operated at low-V_{DD}. [9] dealt with packing for a low-power FPGA having logic blocks that, when idle, can be placed into a low leakage sleep state.

Recent work on performance-driven packing includes [6] which also uses a Rent's rule-based algorithm, and prevents loosely connected LUTs from being packed together. [14] looked at packing in the context of logic replication for performance; a subset of LUTs are deliberately left empty by the packer to accomodate later LUT replications during placement. An interesting recent work by Lin et. al. considered technology mapping and packing in concert and showed that higher speed can be attained by a unified algorithm for concurrent packing and technology mapping [12].

All of the papers noted above target the logic block model of Fig. 1, which is representative of early Altera FPGAs [1], however differs from the logic blocks in modern Xilinx or Altera FPGAs. Modern logic blocks present a more complex packing problem, new optimization opportunities and impose different constraints, as we will illustrate in the sections below.

2.2 Virtex-5 FPGA Architecture

We now review the Virtex-5 blocks that are the target of our packing optimizations.

2.2.1 Logic Blocks

Fig. 2 depicts a Virtex-5 logic block, which is referred to as a configurable logic block (CLB), comprising two SLICEs and a switch matrix. The switch matrix allows for connections from a SLICE back to the same SLICE, between the two SLICEs, as well as into rows and columns of general interconnect. As shown, each SLICE contains four 6-input LUTs and 4 flip-flops. The LUTs in Virtex-5 are implemented as "true" 6-LUTs, rather than being constructed using smaller LUTs that can be optionally combined together via multiplexers. The true 6-LUTs provide excellent performance for implementing large 6-input logic functions. Notice that SLICEs receive and produce carry signals from neighboring SLICEs for implementing fast arithmetic functions.

Fig. 3 provides additional details on the LUT/flip-flop pair within a SLICE. Dashed lines in the figure represent connections whose details are omitted for clarity, but will be described in a subsequent section. Traditionally, a LUT has a single output; however, LUTs in Virtex-5 FPGAs have two outputs, $O5$ and $O6$. The LUT can implement a single logic function using up to 6 inputs, with the output signal appearing on $O6$. Alternately, the LUT can implement two functions that together use up to 5 inputs. In this case, both the $O5$ and $O6$ outputs are used, and the LUT input $A6$ must be tied to logic-1. Notice that when two LUT outputs are

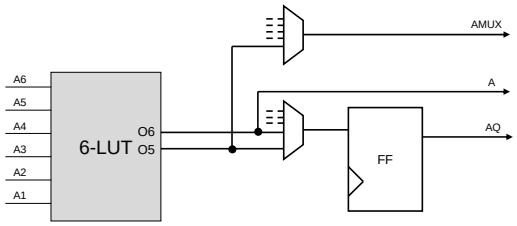

Figure 3: Dual-output 6-input LUT.

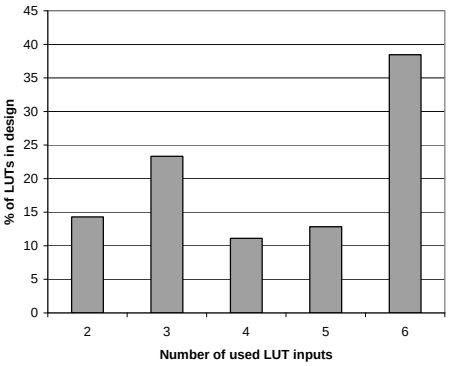

Figure 4: LUT input usage for sample design.

used, only one of them can be registered. A key observation is that $O6$ directly drives a SLICE output, A, whereas $O5$ must pass through an additional multiplexer before reaching SLICE output, $AMUX$. This makes output $O5$ *slower* than output $O6$. Consequently, for high performance, $O5$ should not be used in timing-critical combinational paths.

Although many LUTs in circuits utilize all 6 inputs, the dual-output LUT is useful for circuits that contain many small logic functions, for example, with 2 or 3 inputs. We have observed that industrial FPGA circuits do exhibit this property. For example, Fig 4 shows a histogram (similar to that in [10]) of LUT sizes from a circuit collected from a Xilinx customer. Observe that 37% of LUTs use 2 or 3 inputs, and therefore, could be paired together in a dual-output 6-LUT. This bodes well for the usability of the second LUT output. Section 3 describes a packing approach for using both LUT outputs to produce higher logic density and reduced circuit power, while minimizing the performance impact of the slower $O5$ output.

2.2.2 Block RAMs and DSPs

Fig. 5(a) shows a Virtex-5 block RAM, which can store up to 36 Kbits of data. A block RAM can operate as a single 36 Kb RAM, or as two independent 18 Kb RAMs. The RAMs have configurable "aspect ratio", e.g., an 18 Kb RAM can be configured as 16K x 1, 8K x 2, 2K x 9, or 1K x 18. As shown in the figure, a single 36 Kb RAM can be viewed as having two 18 Kb slots inside it. Reading and writing from the block RAM is a synchronous operation; each block RAM has independent write and read ports available. Multiple block RAMs are organized into columns on the Virtex-5 device and two adjacent block RAMs can be combined to implement deeper memories.

Fig. 5(b) depicts a Virtex-5 DSP block, called the DSP48E. In the main, the DSP48E comprises a 25 x 18 multiplier

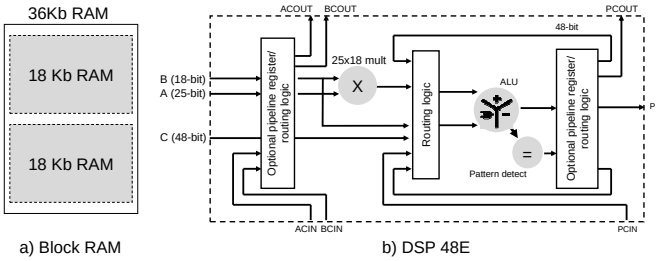

Figure 5: Virtex-5 BRAM and DSP blocks.

which receives inputs A and B. The multiplication result is fed into an ALU that also may receive input C. The ALU can implement addition, subtraction, and varied logic functions. A multiply-accumulate operation, pervasive in DSP circuits, can be realized in a single DSP48E. Multiple DSP48Es can be chained together to form more complex functions through the $PCIN$ and $PCOUT$ ports shown in the figure. Configurable registers (for pipelining) and routing circuitry is present at several places in the block. The interested reader is referred to [3] for the complete details of the DSP48E.

The Virtex-5 device layout contains columns of block RAMs and columns of DSP48Es, integrated into the regular fabric of logic blocks and routing. Section 4 describes a use case combining block RAMs and DSPs that occurs frequently in DSP circuits, and gives for it a simple packing block RAM packing optimization, yielding improved performance.

3. PACKING FOR LOGIC BLOCKS

As mentioned earlier, in the traditional FPGA CAD flow logic elements are clustered together during the packing stage and the placer solves the placement problem for these clusters. For the Virtex-5 6-LUT, it is certainly possible to do the dual-output packing during the packing step of the flow. However, due to the extra delay on the $O5$ output, any timing critical LUT should not be packed to use it. As the placer has better estimates on connection delay, we do this LUT packing during the placement stage.

3.1 Placer Flow

The input to the Virtex-5 placer is a netlist composed of LUTs, registers, large IP blocks, and some pre-packed SLICEs, which may or may not be able to accomodate additional LUTs and/or registers. The core algorithm in the placer uses analytical techniques, as in [7, 17]. After initial IO placement, the analytical placer begins with an overlapped placement of a design and then uses a force abstraction to move logic blocks away from highly congested regions, ultimately producing a feasible, overlap-free placement. Once analytical placement is finished, swap-based local optimization is run on the placed design to further refine the placement. Fig. 6 shows the overall placer flow.

The traditional cost function used in the placer considers wirelength and timing as follows:

$$Cost = a \cdot W + b \cdot T \quad (1)$$

where W and T are the wirelength and timing costs, respectively, and a and b are scalar weights. The values of a and b can be set according to the relative priority of timing

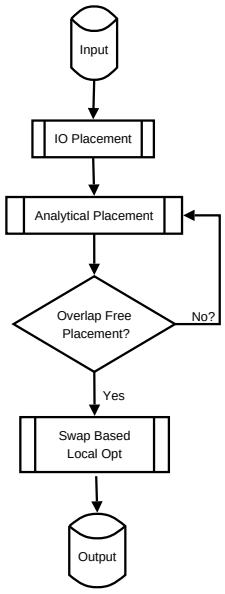

Figure 6: Placer flow

versus wirelength. As actual routes are not available during placement, the wirelength cost depends on wirelength estimation. Likewise, the timing cost depends on estimates of connection delays and user-supplied constraints.

3.2 Packing During Placement

3.2.1 Utilizing Both LUT Outputs

The first step in realizing the dual-output LUT packing was to "open up" the usage of the second LUT output, $O5$, thereby permitting two LUTs to be packed together. Naturally, this involved implementing legality checking on whether two LUTs can feasibly pack together. Two primary legality checks are made: First, we check that the total input signal count of the two LUTs does not exceed 5 unique signals. Second, we check whether the two LUTs drive registers, and if so, we disqualify them from packing together. The reason for this second restriction is that there is only a single register per dual-output 6-LUT. If two LUTs feeding registers were packed into the 6-LUT, one of the LUTs would not attain a fast connect to its fanout register, damaging performance considerably. Other detailed packing restrictions that we found to be useful are described below.

Analytical placement algorithms compute a floating point-valued placement for each design object and then "snap" the objects into the discrete slots available in the FPGA grid. Opening up the second LUT slot in each 6-LUT permits a better fitting of the design following analytical placement. Specifically, the additional slots permit design objects to be snapped into slots that are closer to their floating point placements, reducing the difference between the analytical placement results and the gridded placement, which should translate into improved placed wirelength.

To encourage packing of the second LUT, we extended the cost function of the analytical placement and local optimization by adding a new component. The modified cost function is as follows:

$$Cost = a \cdot W + b \cdot T + c \cdot L \tag{2}$$

where L is computed based on the occupancy of the 6-LUTs and c is a scalar weight. To calculate L, assume $LUTS = \{LUT_1 \ \ldots \ LUT_n\}$ represents the set of all dual-output 6-LUT slots that are used by a given placement solution. Here, we are only interested in what occupies each 6-LUT slot, LUT_i, and not where the slot LUT_i is located on the device. Now,

$$L = \sum_{i=0}^{n} l \cdot L_i \tag{3}$$

where l is a scalar weight and L_i is the occupancy cost of LUT_i, calculated as:

$$L_i = \begin{array}{ll} 1 & \text{if both O5 and O6 outputs are used} \\ 2 & \text{if only O6 output is used} \\ 3 & \text{if only O5 output is used} \end{array}$$

The above cost is minimized if two LUTs are packed together. If a single LUT is placed in a 6-LUT slot, preference is given to the scenario where the LUT uses the $O6$ output (the fast output). The delay estimator was also updated to properly model the increased delay incurred by the $O5$ output signal routing through the additional multiplexer – a change necessary for maintaining high performance.

3.2.2 Improving SLICE Utilization

The extended cost function in (2) aims to reduce the total number of 6-LUTs being used, and consequently reduces the total number of SLICEs being used. A further extension to the cost function was incorporated to reduce SLICE count beyond that achievable by (2) alone, as follows:

$$Cost = a \cdot W + b \cdot T + c \cdot L + d \cdot S \tag{4}$$

where d is a scalar weight. To compute S, assume that $SL = \{SLICE_1 \ \ldots \ SLICE_m\}$ represents the set of all SLICEs in the device. Then, $S = \sum_{SLICE_i \ \epsilon \ SL} S_i$, where S_i represents the LUT utilization of $SLICE_i$. Let N_i represent the number of used LUTs in $SLICE_i$ in the current placement. N_i is at most 8, counting the two sub-LUTs of a 6-LUT separately. S_i is then given by:

$$S_i = \begin{array}{l} 0, \text{ if } N_i \text{ is zero, else} \\ (1 - N_i \ / \ 8), \text{ if } N_i > 0 \end{array}$$

S_i is zero if either none or all of the LUTs are utilized in $SLICE_i$. Otherwise, the cost is proportional to the number of vacant LUTs in the SLICE.

3.3 Recovering Performance

As mentioned above, a Virtex-5 SLICE contains four 6-LUTs, four flip-flops, wide-function multiplexers, and carry logic. Fig. 7 expands on Fig. 3 to show a detailed view of one quarter of a Virtex-5 SLICE. As shown in the figure, the $O6$ LUT output can exit the SLICE through the A output, drive the select line of the carry-logic, feed the XOR gate or the flip-flop, or go to the wide-function multiplexer, $F7$. Likewise, $O5$ can exit the SLICE through the $AMUX$, feed the carry-logic chain or the flip-flop.

The registers in a SLICE are normally used in edge-triggered mode, however, they can also be configured as a transparent "buffer" latch. There are certain cases where using the $O5$ output of the 6-LUT necessitates configuring the flip-flop

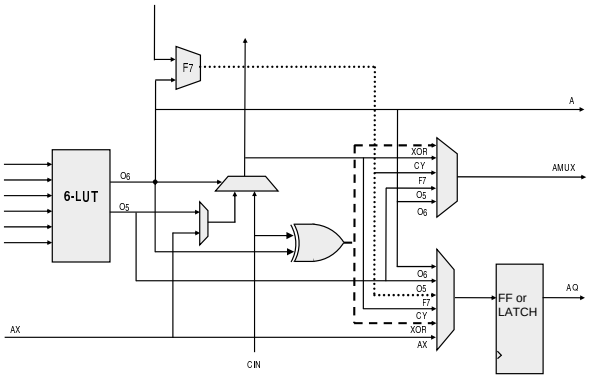

Figure 7: Details of portion of SLICE.

as a buffer latch[1], which adds additional delay, worsening overall performance. To reduce the impact of the packing on performance, we restrict such cases. Here, we highlight two such cases.

3.3.1 Wide-Function Multiplexer

If the wide-function multiplexer is used unregistered, then its output, marked as the dotted line from $F7$ in Fig. 7, uses the $AMUX$ to exit the SLICE. In this case, the $O6$ output is driving the $F7$. If a second LUT is placed to use the $O5$ output, then either the $O5$ or the $F7$ output can use the $AMUX$, the other output can only exit the SLICE through the flip-flop. The flip-flop therefore needs to be configured as a buffer latch through. We prohibit the LUT packing in such a case.

3.3.2 Dedicated XOR Gate

When the dedicated XOR gate is used unregistered, then its output, marked as the bold dashed line in Fig. 7, uses the $AMUX$ to exit the SLICE. Similar to the above senario, placing a second LUT to use the $O5$ output would cause contention between the two signals, resulting in one of them needing to use the flip-flop as a latch through.

3.4 Results

3.4.1 Area and Speed Performance

We evaluate area and speed performance of the dual-output LUT packing using 39 large designs collected from Xilinx customers. Each design contains at least 4000 LUTs. Each design was placed and routed with a very difficult, yet achievable speed performance (clock period) constraint. Specifically, for each circuit we began with an easy-to-meet performance constraint and iteratively increased the constraint to find the highest meet-able constraint for each circuit. The constraint generation process was repeated twice for each circuit: with and without the dual-output LUT packing described above. Note that the technology mapper used was the same in all cases; that is, we did not alter the mapper to favor producing LUTs with fewer inputs for the case of dual-output LUT packing.

[1]A buffer latch is a level-sensitive latch with the clock input tied to logic-1, thereby making the latch transparent at all times.

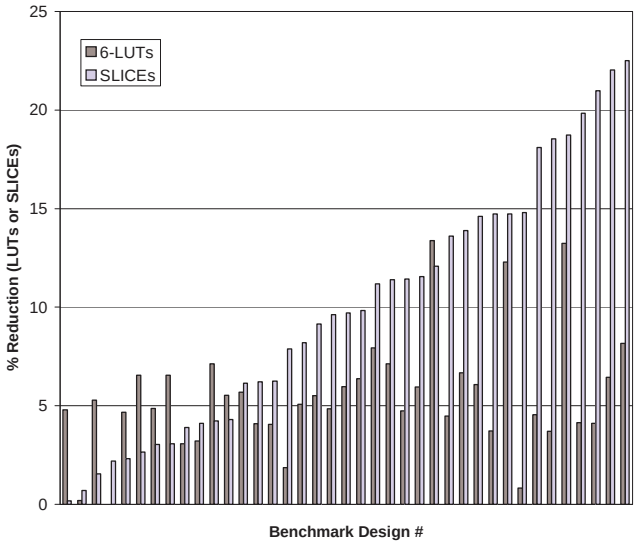

Figure 8: Area reduction results for LUTs and SLICEs.

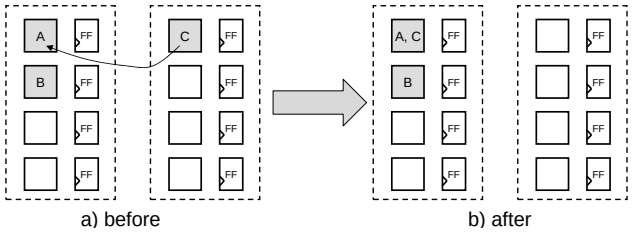

Figure 9: Example of LUT and SLICE reduction.

Fig. 8 presents the area reduction results. Two data points are shown for each circuit. The first is the percentage reduction in LUT-6s due to the dual-output LUT packing; the second data point is the reduction in Virtex-5 SLICEs. On average, across all the circuits, the LUT-6 count is reduced by 5.5% and the SLICE count is reduced by 10.5%. Importantly, it is the number of used SLICEs that determines which particular device family member is selected by a customer. Consequently, the SLICE reduction results here are encouraging, as they may allow a customer to move to a smaller part in the Virtex-5 family, thereby reducing cost.

We originally considered the large SLICE count reduction (in comparison with LUT count reduction) somewhat counter-intuitive. However, Fig. 9 illustrates how SLICE reduction can exceed LUT reduction. Fig 9(a) shows 2 SLICEs with 3 occupied LUTs, prior to a packing. In Fig 9(b), LUT C is packed together with LUT A, yielding a single occupied SLICE containing two occupied LUTs. In this case therefore, LUT count is reduced from 3 to 2: a 33% reduction; SLICE count is reduced from 2 to 1: a 50% reduction.

Fig. 10 presents the speed performance results. One data point is presented for each circuit. The vertical axis represents the percentage reduction in speed performance for each circuit, and the horizontal axis represents the percentage reduction in SLICE count. Observe that some circuits

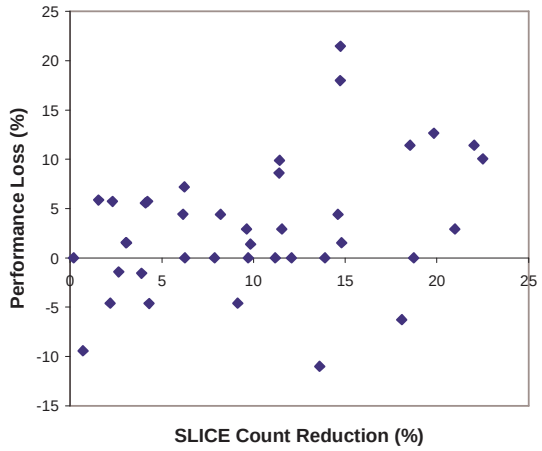

Figure 10: Performance reduction (%) versus SLICEs reduction (%).

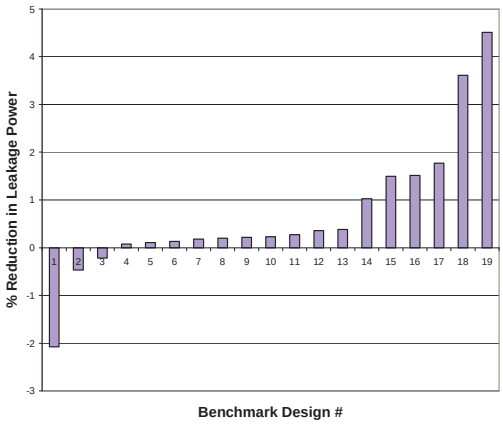

Figure 11: Leakage power reduction results.

improved in speed performance (negative values) and some circuits degraded (positive values). On average, speed performance degraded by 3.3% across all circuits. Notice that there appears to be no clear correlation between a circuit's SLICE reduction and its performance change. One way to interpret the area/speed results is that dual-output LUT packing affords a 10.5% SLICE reduction, at a cost of about 3.3% speed performance, on average. We believe this is a trade-off that will be desirable to many customers.

Regarding the "noisy" speed performance results, we believe several factors are at play. First, higher packing density produces a more compact placement, having short wirelengths and higher performance, as is apparent for some circuits. On the other hand, packing increases pin density in the layout, worsening routing congestion and performance for some circuits. And further, as explained above, the $O5$ LUT output is inherently a slower output than $O6$, increasing the delay of some combinational paths. Likewise, packing two LUTs together means that neither function can use the fast LUT input, $A6$, which must be tied to logic-1 when both outputs are used. Lastly, some of the performance swings may be due the "instability" inherent in any FPGA layout system based on heuristic algorithms. Certainly, the performance consequences of the logic block packing are complex and warrant further investigation and tuning.

For comparison, we also created a very aggressive version of dual-output LUT packing that permits connected LUT/FF pairs to be separated, and does not include the performance recovery techniques mentioned in Section 3. This more aggressive packing reduced SLICE count by about the same amount, and reduced LUT count by twice the amount, on average, however, the speed performance loss was over 10%, which we believe would not be acceptable to most customers.

3.4.2 Power

Power in the core part of an FPGA is consumed in both its logic and interconnect [15]. The packing techniques described above lead to lower logic usage, potentially reducing both static and dynamic power. Furthermore, the reduced and LUT and SLICE counts may permit the placer to achieve a "tighter" layout with shorter wirelengths, thereby reducing interconnect usage and power.

We investigated power using 19 industrial designs collected from Xilinx customers targeted to Virtex-5. The designs were augmented with built-in automatic input vector generation by attaching a linear feedback shift register-based (LFSR-based) pseudo-random vector generator to the primary inputs. This permitted us to perform board-level measurements of power without requiring a large number of externally applied waveforms. The results presented here are based on measurements of current drawn from the V_{ccint} supply during circuit operation. Dynamic power was separated from static power by taking each design and clocking it at multiple frequencies, making a current measurement at each frequency, and then using the property that dynamic power scales linearly with frequency to break out dynamic from static current.

Fig. 11 shows the percentage improvement in leakage power afforded by the packing optimization. Negative values in the figure represent power increases. Observe that for 3 of the 19 designs, leakage power was actually made worse by the packing optimization. However, for the majority of designs (16 of 19), power improved. Across all designs, the average leakage power improvement is a modest 0.7%. We also measured total power (including static and dynamic) and found the total power improvement to be similar, 0.7%, on average across the designs. Thus, we conclude that LUT packing is generally beneficial to power, and the approach can be combined with the techniques in the Xilinx power-aware flow [8].

4. PACKING FOR BLOCK RAMS

Memory density is an important FPGA metric because customer designs are commonly limited by block RAM availability. One way around this is to combine small block RAMs together to increase the memory density of a design. However, the packing and placement of these combined block RAMs can have a large impact on the performance.

We describe two block RAM packing algorithms that contribute to increased memory density and performance. The first algorithm is a block RAM and DSP packing optimization. The second algorithm packs chains of block RAMs together to increase memory density which allows for a tighter placement. We will describe each algorithm in turn.

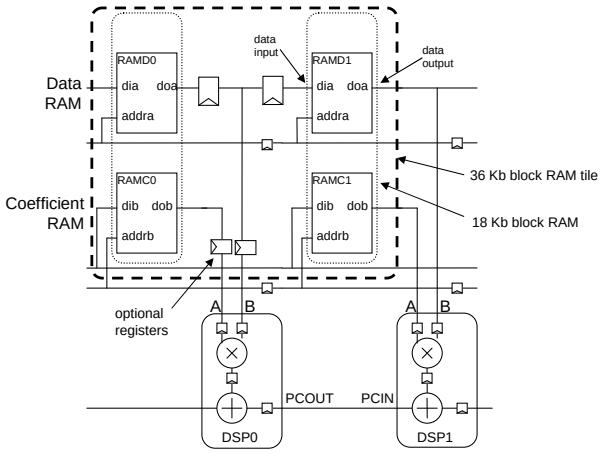

Figure 12: 2-tap FIR implementation.

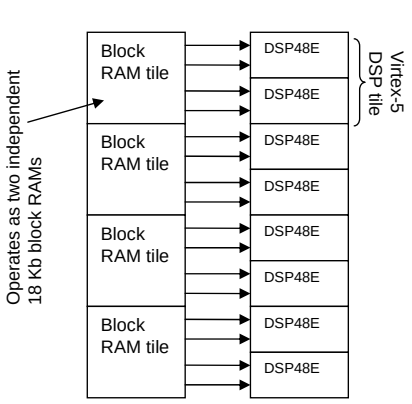

Figure 13: Placement of FIR implementation.

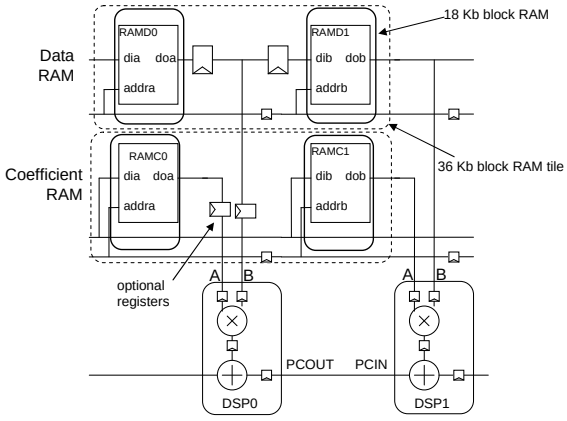

Figure 14: Alternate 2-tap FIR implementation.

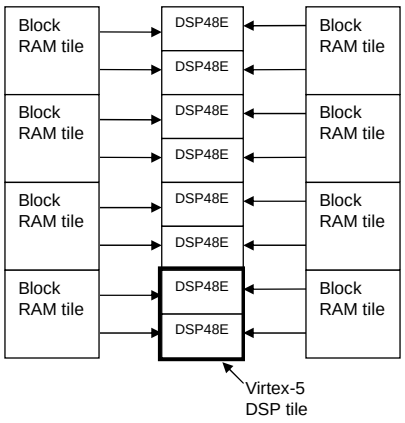

Figure 15: Placement of alternate implementation.

4.1 Block RAMs and DSPs

DSP circuits commonly use both block RAM and DSP blocks together. Perhaps the most common DSP application is a Finite Impulse Response (FIR) filter. FIR filters are a type of DSP circuit that contain multiple instances of a sub-circuit referred to a "tap". Fig. 12 illustrates a 2-tap FIR filter. In essence, each filter tap multiplies a coefficient word by a data word and accumulates the result. In Virtex-5 FPGAs, each 18 Kb RAM and the DSP it drives can implement a single tap of an FIR filter. The registers in the diagram represent registers that the designer may or may not have added. The block RAMs on the top of the figure are "data" block RAMs. The block RAMs on the bottom are the "coefficient" block RAMs.

If the data and coefficient RAMs are 9 Kb RAMs or less, the block RAMs for two taps can be mapped into two 18 Kb block RAMs which can then be packed into a single Virtex-5 block RAM tile. Thus, a single 36 Kb block RAM tile can hold the data and coefficients for two FIR taps. This creates a natural alignment, as the pitch of the Virtex-5 block RAM tile matches that of the DSP tile (comprising two DSP48Es). This is illustrated in Fig. 13, which shows an 8-tap FIR filter.

The second FIR filter use case we target is shown in Fig. 14, where the block RAMs driving the DSP48Es are too large to fit into one 36 Kb block RAM tile. This scenario occurs if the FIR block RAMs require simple dual port mode (which can have larger widths than the true dual port block RAMs in Fig. 12), if their depth is greater than 512-deep (which is the case for the single-port RAMs in Fig. 14), or in the case of higher required data precision. In this case, the data block RAMs and the coefficient block RAMs are packed together into two 36 Kb block RAM tiles. As can be seen in Fig. 15, this also permits a natural alignment, where two columns of block RAM are used to drive the DSP column.

The algorithm for finding the block RAMs and packing them together, applicable to both use cases noted above, is as follows:

1. Iterate over the netlist and find DSP48E chains, where a DSP48E chain consists of DSP48E blocks connected to one another through PCOUT-to-PCIN connections.

2. For each DSP48E chain, pack the block RAMs driving the chain as follows:

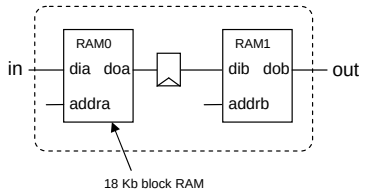

Figure 16: Block RAM chaining example.

```
i = 0;
while (i < size of the DSP chain - 1)
  DSP1 = dspChain[i];
  i++;
  DSP2 = dspChain[i];
  i++;

  // get the block RAMs driving the DSP on a
  // specific input port

  bram1c = getRAMDrivingDSP(DSP1, port "A")
  bram2c = getRAMDrivingDSP(DSP2, port "A")
  bram1d = getRAMDrivingDSP(DSP1, port "B")
  bram2d = getRAMDrivingDSP(DSP2, port "B");

  Pack together bram1c and bram2c, if possible
  Pack together bram1d and bram2d, if possible
```

The rationale behind using a "while" loop instead of using a "for" loop is to allow the packing pattern to commence at either an even or odd DSP block in the chain. In the implementation of the function "getRAMDrivingDSP", we also recognize indirect connectivity between a DSP and a block RAM that traverses up to two intermediate registers.

4.2 Block RAM Chains

Occasionally, a series of block RAMs can be chained together where each block RAM in the chain is connected to the next block RAM by a data-out to data-in connection. This can happen, for example, when the block RAM is used to implement a FIFO. An example of two 18 Kb block RAMs that should be packed together into a single Virtex-5 block RAM tile is shown in Fig. 16. Our block RAM packing traverses the netlist to identify such chains and packs them accordingly. As above, indirect connections between block RAMs are also handled, for example, when two block RAMs connect serially through a register file.

4.3 Results

We benchmarked the algorithms against five internally generated benchmark circuits that contain the specific FIR filters in question and/or block RAM chains. The designs have between 30-64 block RAMs, and up to 58 DSP48E units. We used the same iterative clock tightening methodology described in Section 3.4 for finding the highest achievable performance constraint for each circuit.

The results can be seen in Table 1, where performance is measured in MHz. Across the five designs, the average performance increase was 11%, which is quite encouraging, given the simplicity of the RAM packing optimizations.

Table 1: Block RAM packing results.

Circuit	Performance RAM Packing	Performance Baseline	Percent Improvement
Circuit1	500	400	25%
Circuit2	450	365	23%
Circuit3	500	470	6%
Circuit4	425	435	-2%
Circuit5	215	200	8%
Geomean	400	359	11%

5. CONCLUSIONS

Modern FPGAs contain composite logic blocks with multiple LUTs, flip-flops, multipliexers and other circuitry. The latest FPGAs from Xilinx and Altera contain multi-output LUTs, capable of implementing one large logic function, or two smaller logic functions. Packing design elements into the available logic blocks is a complex problem, with broad impact on the metrics of area, speed and power. Likewise, the IP blocks integrated into today's FPGAs present unique packing problems. In this paper, we described approaches for logic block and block RAM packing for Virtex-5 FPGAs. Our dual-output LUT packing lowers the number of used SLICEs by 10.5%, on average, with a 3.3% performance hit. The block RAM packing is targeted towards DSP circuits and improves performance by 11%, on average. Both optimizations are available in the Xilinx ISE^{TM}10.1i software. The LUT packing optimization can be invoked using the "`-lc auto`" switch on the `map` command line.

This work represents a first step in architecture-specific packing for Virtex-5 FPGAs. One direction for future work is to more closely integrate the dual-output LUT packing with technology mapping. It is conceivable that mapping can be altered to produce more 2 and 3-input LUTs that can be combined together in a single 6-LUT during placement, yielding greater SLICE count reductions and higher logic density. It may also be possible to target certain physical synthesis optimizations towards improved usage of the dual-output LUT. For the block RAM packing, we plan to extend the work to cover additional FIR filter implementations and other types of DSP circuits, such as FFTs.

6. ACKNOWLEDGMENTS

The authors thank Kamal Chaudhary for his helpful suggestions on this work. The authors also thank Billy Chan, Stephen Jang, and Humberto Rico Romo for providing the aggressive packing results mentioned in Section 3.4.

7. REFERENCES

[1] *FLEX 10K Programmable Logic Device Datasheet.* Altera, Corp., San Jose, CA, 2003.

[2] *Virtex-5 FPGA Family Datasheet.* Xilinx, Inc., San Jose, CA, 2007.

[3] *Virtex-5 XtremeDSP Design Considerations User Guide.* Xilinx, Inc., San Jose, CA, 2007.

[4] V. Betz and J. Rose. Cluster-based logic blocks for FPGAs: Area-efficiency vs. input sharing and size. In *IEEE Custom Integrated Circuits Conference*, pages 551–554, Santa Clara, CA, 1997.

[5] D. Chen and J. Cong. Delay optimal low-power circuit clustering for FPGAs with dual supply voltages. In

ACM/IEEE International Symposium on Low-Power Electronics and Design, pages 70–73, Newport Beach, CA, 2004.

[6] M. Dehkordi and S. Brown. The effect of cluster packing and node duplication control in delay driven clustering. In *IEEE International Conference on Field-Programmable Technology*, pages 227–233, Hong Kong, 2002.

[7] H. Eisenmann and F. Johannes. Generic global placement and floorplanning. In *ACM/IEEE Design Automation Conference*, pages 269–274, San Francisco, CA, 1998.

[8] S. Gupta, J. Anderson, L. Farragher, and Q. Wang. CAD techniques for power optimization in Virtex-5 FPGAs. In *IEEE Custom Integrated Circuits Conference*, pages 85–88, San Jose, CA, 2007.

[9] H. Hassan, M. Anis, and M. Elmasry. Lap: A logic activity packing methodology for leakage power-tolerant FPGAs. In *ACM International Symposium on Low Power Electronics and Design*, pages 257–262, San Diego, CA, 2005.

[10] M. Hutton, J. Schleicher, D. Lewis, B. Pedersen, R. Yuan, S. Kaptanoglu, G. Baeckler, B. Ratchev, K. Padalia, and et. el. Improving FPGA performance and area using an adaptive logic module. In *International Conference on Field-Programmable Logic and Applications*, pages 135–144, Antwerp, Belgium, 2004.

[11] I. Kuon and J. Rose. Measuring the gap between FPGAs and ASICs. *IEEE Transactions On Computer-Aided Design of Integrated Circuits and Systems*, 26(2):203–215, Feb. 2007.

[12] J. Lin, D. Chen, and J. Cong. Optimal simultaneous mapping and clustering for FPGA delay optimization. In *ACM/IEEE Design Automation Conference*, pages 472–477, San Francisco, CA, 2006.

[13] A. Marquardt, V. Betz, and J. Rose. Using cluster based logic blocks and timing-driven packing to improve FPGA speed and density. In *International Symposium on Field-Programmable Gate Arrays*, pages 37–46, Monterey, CA, 1999.

[14] K. Schabas and S. Brown. Using logic duplication to improve performance in FPGAs. In *ACM International Symposium on Field-Programmable Gate Arrays*, pages 136–142, Monterey, CA, 2003.

[15] L. Shang, A. Kaviani, and K. Bathala. Dynamic power consumption of the Virtex-2 FPGA family. In *ACM International Symposium on Field-Programmable Gate Arrays*, pages 157–164, Monterey, CA, 2002.

[16] A. Singh and M. Marek-Sadowska. Efficient circuit clustering for area and power reduction in FPGAs. In *ACM International Symposium on Field-Programmable Gate Arrays*, pages 59–66, Monterey, CA, 2002.

[17] N. Viswanathan, G.-J. Nam, C. Alpert, P. Villarrubia, H. Ren, and C. Chu. RQL: Global placement via relaxed quadratic spreading and linearization. In *ACM/IEEE Design Automation Conference*, pages 453–458, San Diego, CA, 2007.