Packing Techniques for Virtex-5 FPGAs

TANEEM AHMED, PAUL D. KUNDAREWICH, and JASON H. ANDERSON
Xilinx, Inc.

18

Packing is a key step in the FPGA tool flow that straddles the boundaries between synthesis, technology mapping and placement. Packing strongly influences circuit speed, density, and power, and in this article, we consider packing in the commercial FPGA context and examine the area and performance trade-offs associated with packing in a state-of-the-art FPGA—the Xilinx® Virtex™-5 FPGA. In addition to look-up-table (LUT)-based logic blocks, modern FPGAs also contain large IP blocks. We discuss packing techniques for both types of blocks. Virtex-5 logic blocks contain dual-output 6-input LUTs. Such LUTs can implement any single logic function of up to 6 inputs, or any two logic functions requiring no more than 5 distinct inputs. The second LUT output has reduced speed, and therefore, must be used judiciously. We present techniques for dual-output LUT packing that lead to improved area-efficiency, with minimal performance degradation. We then describe packing techniques for large IP blocks, namely, block RAMs and DSPs. We pack circuits into the large blocks in a way that leverages the unique block RAM and DSP layout/architecture in Virtex-5, achieving significantly improved design performance.

Categories and Subject Descriptors: B.7.2 [**Integrated Circuits**]: Design Aids

General Terms: Design, Algorithms

Additional Key Words and Phrases: Field-programmable gate arrays, FPGAs, optimization, packing, placement, technology mapping, performance, power, logic density

ACM Reference Format:
Ahmed, T., Kundarewich, P. D., and Anderson J. H. 2009. Packing techniques for Virtex-5 FPGAs. ACM Trans. Reconfig. Techn. Syst. 2, 3, Article 18 (September 2009), 24 pages.
DOI = 10.1145/1575774.1575777. http://doi.acm.org/10.1145/1575774.1575777.

1. INTRODUCTION

Since their birth in the mid-1980s, field-programmable gate arrays (FPGAs) have experienced dramatic growth and have become an increasingly popular choice for digital circuit implementation. The earliest FPGAs were uniform arrays of look-up-table (LUT)-based logic blocks, surrounded by a ring of I/Os. Each subsequent generation of FPGAs has become more heterogeneous and incorporated different types of specialized ASIC-like hard IP blocks. The drive

Authors' addresses: T. Ahmed, P. D. Kundarewich, and J. H. Anderson, Xilinx, Inc., 505-30 St. Clair Avenue West, Toronto, ON M4V 3A1, Canada; email: {taneema, paulku}@xilinx.com, janders@eecg.toronto.edu.

towards including more and varied hard IP in FPGAs is rooted in the goal of mitigating the overhead associated with the programmability of FPGAs—an overhead that makes FPGAs both slower and less area-efficient than custom ASICs for implementing a given logic circuit [Kuon and Rose 2007]. The hard IP blocks in state-of-the-art FPGAs include block RAMs, digital signal processing (DSP) blocks, analog-to-digital converters, specialized high-speed I/Os and even entire microprocessors.

Packing is the step of the FPGA computer-aided design (CAD) flow that traditionally falls between technology mapping and placement, but that recently has become more tightly integrated with placement, technology mapping, and synthesis. Packing, also known as *clustering* in the FPGA CAD literature, is the step wherein the elements of the technology mapped circuit are packed into the available FPGA hardware resources. Clusters of LUTs and flip-flops form the basis for logic blocks in today's FPGAs, with fast local interconnect available for intralogic block connectivity. Most commonly, the packing step is considered to be that which combines a design's LUTs and flip-flops together to form logic 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 [Betz and Rose 1997]. To our knowledge, no prior work has been published on packing for commercial FPGAs, which have more complex hardware structures. Notably, in commercial FPGAs the definition of packing is considerably broader than LUTs and flip-flops, and includes packing design elements into the hard IP blocks.

This article describes packing techniques for the logic and hard IP blocks present in the Xilinx Virtex-5 FPGA. As opposed to traditional packing, based solely on circuit connectivity, our proposed techniques are directly incorporated into the placer, allowing the placer to base decisions on physical information. Our first packing technique targets the dual-output 6-input LUT available in Virtex-5 logic blocks, with the aim of improving area-efficiency. LUTs in Virtex-5 FPGAs can implement any single logic function of 6 variables or less, or can implement two functions that together need 5 variables or less. In dual-output mode, however, the two outputs are not identical—one has considerably slower speed than the other, and this must be recognized during the packing step if speed performance is not to be compromised. Our second technique targets two types of large IP blocks in Virtex-5 FPGAs: block RAMs and DSPs. FPGA designs frequently use the block RAMs and DSPs in a limited number of predictable ways, and such "patterns" can be detected and packed in a manner well-suited to the underlying hardware. We describe a packing technique that improves design performance for DSP-oriented circuits.

Both of the packing techniques described in this paper are architecture-specific, in the sense that they leverage hardware features in Virtex-5. However, the techniques presented are general enough to be easily adapted to other commercial families or FPGAs from other vendors. One of our objectives with this article is to bridge the gap between the academic and vendor perspectives of the packing problem, and to encourage the academic community to research packing algorithms that can function with the constraints and hardware structures present in modern commercial FPGAs.

A preliminary version of this work appeared in Ahmed et al. [2008]. In this extended journal version, we give a more thorough treatment of the proposed techniques. More significantly, we have retuned our algorithms to reflect an area and performance trade-off that we believe will be more desirable to customers. As such, for this article, we have regenerated all of our experimental results, and we also present new experimental work on the area/performance trade-off space of the proposed packing techniques.

The balance of the article is organized as follows: Section 2 reviews related work on packing and introduces the Virtex-5 FPGA architecture. Packing techniques that target logic blocks are offered in Section 3. Packing techniques for large IP blocks appear in Section 4. Conclusions and suggestions for future work are given in Section 5.

2. BACKGROUND

2.1 Packing

Figure 1 depicts the classic FPGA logic block model used in the vast majority of academic research. It consists of a cluster of LUTs and flip-flops, where each flip-flop can be optionally bypassed for implementing combinational logic. Static RAM cells, not shown in the figure, control the logic function of each LUT, and also control the multiplexer and interconnect connectivity. Inputs to the logic block come from the FPGA's general interconnect: 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. Most prior packing work assumes the local interconnect to be a full crossbar switch matrix—every input can be programmably connected to any output. In this logic block model, connections *within* the logic block are fast, and connections *between* logic blocks, routed through the general interconnect, are slow in comparison. The packing step decides which LUTs to put together into a single logic block, and therefore, packing has a significant impact on circuit speed. The model of Figure 1 is representative of early Altera FLEX FPGAs [Altera 2003], however, it has become out-of-step with the logic blocks present in modern Xilinx or Altera FPGAs. Modern logic blocks present a more complex packing problem and new optimization opportunities and impose different constraints, as we will illustrate in the sections below.

Much research has been published on packing for the logic block in Figure 1. Perhaps the most cited work is that of Betz and Rose, who proposed an area-driven packing algorithm and showed that, due 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 [Betz and Rose 1997]. In particular, for a logic block with N 4-input LUTs, Betz and Rose [1997] showed that only $2N+2$ inputs to the cluster are needed—a number much smaller than $4N$. Marquardt extended the work to perform timing-driven packing and demonstrated the impact of packing on critical path delay [Marquardt et al. 1999].

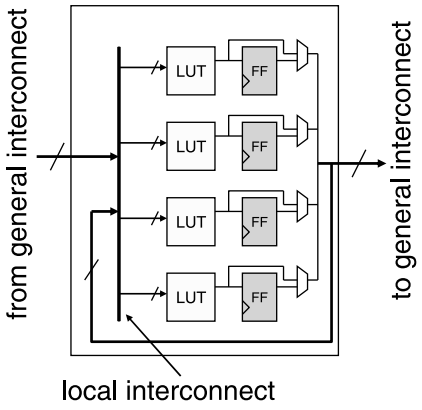

Fig. 1. Classic FPGA logic block targeted by most academic research.

Packing affects power consumption as intralogic block connections will have lower capacitance than interlogic block connections. A natural approach therefore is to attempt to keep nets with high switching activity contained within logic blocks, as was proposed in Lamoureux and Wilton [2003]. An entirely different approach for power-driven packing was shown in Singh and Marek-Sadowska [2002], where 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. Although not yet available commercially, dual-V_{DD} FPGAs have been proposed by academia, where the idea is to programmably allow logic blocks to operate at reduced supply voltage (slower but lower power). Researchers at UCLA developed a complete CAD flow for a proposed dual-V_{DD} FPGA, including new packing techniques [Chen and Cong 2004]. The aim of packing in this context is to pack LUTs based on their timing-criticality, placing noncritical LUTs together into logic blocks that will be operated at low-V_{DD}. Hassan et al. [2005] dealt with packing for a low-power FPGA having logic blocks that when idle, can be placed into a low leakage sleep state.

On the speed axis, more recent work includes Dehkordi and Brown [2002] which also uses a Rent's rule-based algorithm, and prevents loosely connected LUTs from being packed together; that is, it prevents the packing of unrelated LUTs. Other papers tie together packing with other phases of the FPGA CAD flow. For example, Schabas and Brown [2003] 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. [2006] brought together packing and technology mapping and showed that higher speed can be attained using a unified algorithm for concurrent packing and technology mapping.

2.2 Virtex-5 FPGA Architecture

We now describe the Virtex-5 architecture, focusing on the blocks that are the target of our packing techniques.

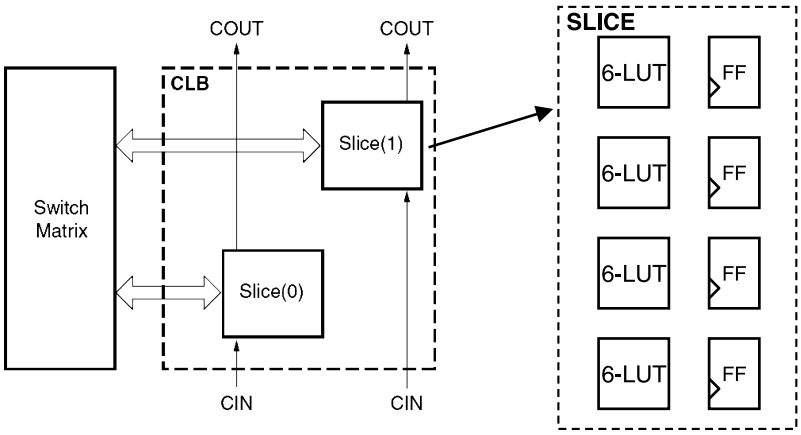

Fig. 2. Virtex-5 CLB and SLICE.

2.2.1 *Logic Blocks.* Logic blocks in Virtex-5 are called configurable logic blocks (CLBs) and comprise two SLICEs and a switch matrix, as shown in Figure 2. The switch matrix serves the purpose of providing intra *and* inter-CLB connectivity. As shown in the figure, each SLICE contains four 6-input LUTs and 4 flip-flops. SLICEs receive and produce carry signals from neighboring SLICEs for implementing fast ripple-carry addition.

Figure 3 zooms in on the details of portion of a SLICE—the LUT/flip-flop pair. Dashed lines in the figure represent connections whose details are omitted for clarity, but will be described below. Traditional LUTs in FPGAs have a single output and can implement a single logic function. However, LUTs in Virtex-5 FPGAs have two outputs, $O5$ and $O6$, and can implement two logic functions. Specifically, the LUT can implement a single logic function using up to 6 inputs, with the output signal appearing on $O6$. The LUT can also implement two functions that together use up to 5 inputs, in which case, both the $O5$ and $O6$ outputs are used, and the LUT input $A6$ must be tied to logic-1. The architecture necessitates that when two LUT outputs are used, only one of them can be registered.

An important property of the LUT/FF structure in Figure 3 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. The relative slowness of $O5$ is something that must be considered when packing LUTs into Virtex-5 SLICEs, if high speed performance is to be maintained.

Figure 4 shows the dual-output LUT hardware implementation. LUTs in FPGAs are implemented with trees of multiplexers, and the figure shows a portion of a 6-input LUT. Values in SRAMs cells (on the left of the figure) are set to hold the truth table for the logic function(s) implemented in the LUT. The fastest LUT input, $A6$, is held constant at logic-1 when two functions are mapped to the LUT, and therefore the fast input is unavailable for either of the

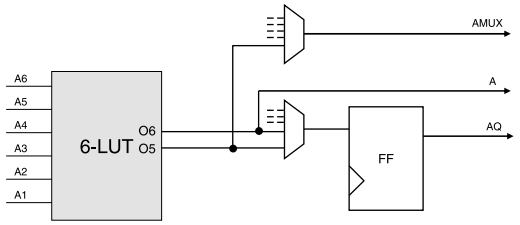

Fig. 3. Dual-output 6-input LUT.

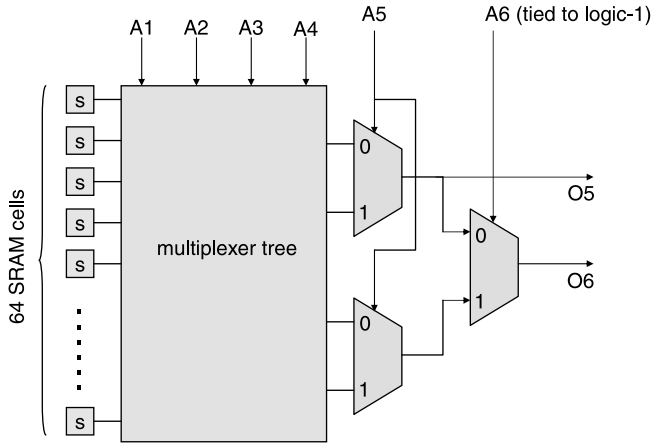

Fig. 4. Hardware implementation of dual-output LUT.

two functions. Thus, *both* logic functions suffer from increased delay in dual-output mode, relative to the case of implementing a single function in a LUT, where the $A6$ can be used. Observe that the $O5$ output is fed by an internal node of the multiplexer tree.

Circuits implemented in an FPGA need not use all 6 LUT inputs, and in fact, circuits frequently contain many small logic functions that need only 2 or 3 LUT inputs. Figure 5 shows a histogram (similar to that in Hutton et al. [2004]) of LUT sizes from a circuit collected from a Xilinx customer. In this case, 37% of LUTs use 2 or 3 inputs, and such small LUTs could be paired together and packed into one dual-output 6-LUT. The frequency of small LUTs in circuits 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, while at the same time minimizing the performance hit of using of the slower $O5$ output. The higher density is achieved as a result of the pervasiveness of small logic functions in circuits that can be packed together in a single dual-output 6-LUT.

2.2.2 *Block RAMs and DSPs.* In early FPGAs, any data used by the on-chip application had to be stored in off-chip RAMs. Accessing memory was slow and power hungry. Modern FPGAs such as Virtex-5 include blocks of

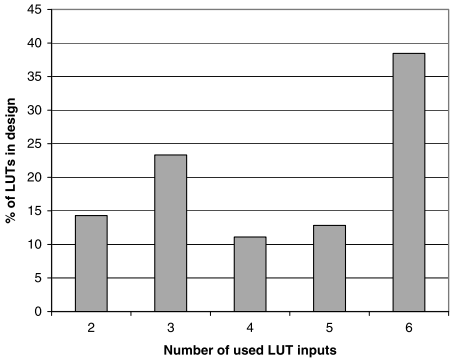

Fig. 5. LUT input usage for sample design.

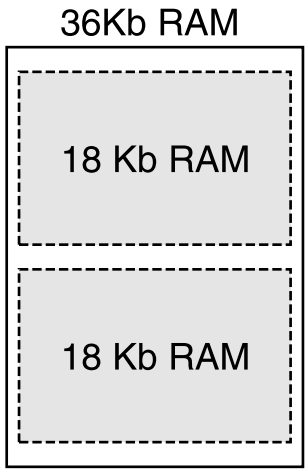

Fig. 6. Virtex-5 block RAM tile.

static RAM on-chip. Figure 6 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 RAM's read and write operations are synchronous. Multiple block RAMs are organized into columns on the Virtex-5 device and two adjacent block RAMs in a column can be combined to implement deeper memories.

When an 18 Kb RAM is used in dual-port mode, there are two possibilities:

(1) True dual-port mode: the 18 Kb block RAM has two independent access ports, A and B. Data can be written to either or both ports, and can be read from either or both ports. Figure 7(a) illustrates this case. In the figure, the two 18 Kb block RAMs in the block RAM tile are referred to as the *upper* and *lower* block RAMs; the two access ports on each 18 Kb block RAM are called the A and B access ports. DI, DO and ADDR refer to data-in, data-out, and address ports, respectively.

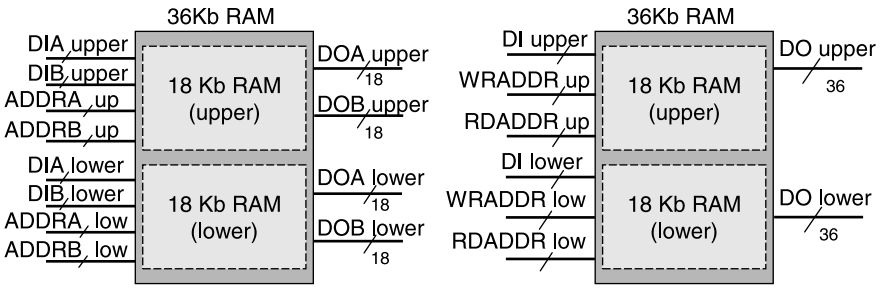

Fig. 7. Dual-port block RAMs: true and simple.

(2) Simple dual-port RAM mode: In this case, each port cannot be used for *both* reads and writes; rather, the 18 Kb block RAM has one port for read operations and an independent port for write operations. Figure 7(b) illustrates this case. In the figure, WRADDR and RDADDR refer to the write-address and read-address ports, respectively.

The maximum allowable word width of a RAM in simple dual-port mode is 36 bits, which is double that allowed for true dual-port mode. The RAMs have configurable "aspect ratio," meaning that an 18 Kb RAM can be configured as 16K × 1, 8K × 2, 2K × 9, or 1K × 18 or for simple dual-port RAMs, 512 × 36.

DSP circuits frequently contain multiply-accumulate operations, and in early FPGAs, such functionality was implemented using the generic LUT fabric. In modern FPGAs, however, such functionality can be realized with improved speed and power using hard IP blocks. Figure 8 depicts a Virtex-5 DSP block, called the DSP48E. The DSP48E comprises a 25 × 18 multiplier which receives inputs A and B as the operands to the multiply block. The product is fed into an ALU that also may receive input C. The ALU can implement addition, subtraction, and varied logic functions. Multiple DSP48Es can chain together to form more complex functions through the $PCIN$ and $PCOUT$ ports shown in the figure. The structure in Figure 8 is not entirely combinational and actually has programmable pipeline depth, via configurable registers present at several places in the block. A comprehensive description of the DSP48E can be found in Xilinx [2007].

The Virtex-5 layout scheme has columns of block RAMs and columns of DSP48Es, integrated into the regular fabric of logic blocks and routing. Section 4 describes packing techniques for block RAMs and DSPs that yield improved performance for circuits containing commonly-occurring patterns of block RAM/DSP instances.

3. PACKING FOR LOGIC BLOCKS

In the traditional FPGA CAD flow, packing is done at the preplacement stage. However, for the case of packing into the dual-output 6-LUTs in Virtex-5, one of the two LUT outputs has slower speed. Packing into 6-LUTs must be done with consideration of the circuit's critical path, if high speed is to be maintained.

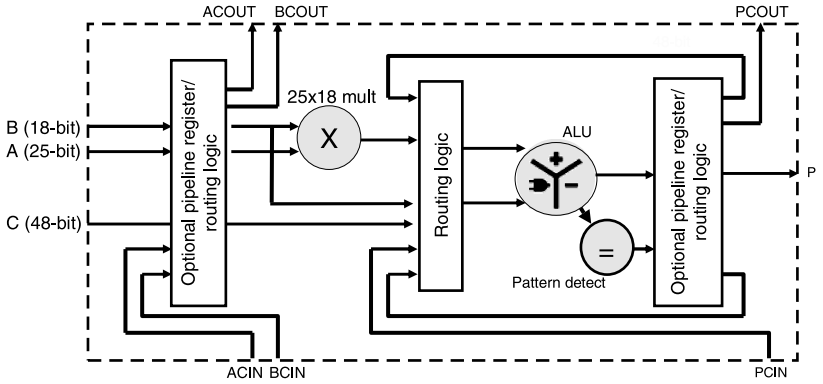

Fig. 8. Virtex-5 DSP48E block.

Because roughly 50% of the path delay in FPGAs is interconnect delay, little is known regarding path criticality at the preplacement stage. Therefore, we choose to do dual-output LUT packing during placement, as we have access to better estimates of connection delay.

3.1 Placer Flow

Figure 9 shows the flow of the Xilinx placer. The input to the flow is a netlist composed of LUTs, flip-flops, large IP blocks, and some prepacked SLICEs, which may or may not be able to accomodate additional LUTs and/or registers.[1] Core logic blocks are placed using analytical techniques, similar to those published in Eisenmann and Johannes [1998] and Viswanathan et al. [2007]. After initial IO placement, the placement problem is represented mathematically, as a system of equations to be solved. An initial overlapped placement of the design is computed by solving the equations. Based on the intermediate placement, the mathematical formulation is modified to move logic blocks away from highly congested regions. The solving and reformulation of the mathematical system continues iteratively until ultimately, a feasible overlap-free placement is produced. Once analytical placement is finished, swap-based local optimization is run on the placed design to further refine the placement. The cost function used in the placer considers wirelength and timing:

$$Cost = a \cdot W + b \cdot T, \tag{1}$$

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

[1]The Xilinx tools permit a user to manually pack the SLICEs in their design, thereby limiting the potential for additional automatic packing into dual-output LUTs.

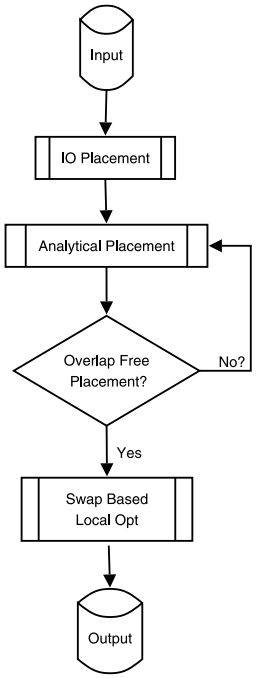

Fig. 9. Placer flow.

3.2 Packing during Placement

3.2.1 *Leveraging Two LUT Outputs.* The first task in creating a dual-output LUT packing algorithm was to "open up" the usage of the second LUT output, $O5$, and permit two LUTs to be packed together. This involved implementing legality checking on whether two LUTs can feasibly pack together. Two principal legality checks are made: First we check that, combined, the two LUTs use no more than 5 distinct variables, and if so, they are disqualified from packing together. Second, we ensure that at most one of the two LUTs drives a register. The reason for this second restriction is that there exists only one register per dual-output 6-LUT. If two LUTs feeding registers were packed into the 6-LUT, one of the LUTs would not be able to attain a fast connect to its fanout register, damaging performance considerably. In fact, if two LUTs needing registers are packed together into a single 6-LUT, one of the two registers must be placed in a nonadjacent register slot, perhaps even in another CLB. Other detailed packing restrictions that we found useful are described in the following paragraphs.

While simulated annealing-based placers keep design objects snapped onto the FPGA grid at all times, analytical placement algorithms operate in the floating point domain. At the end of spreading, when the placement is close to being feasible (overlap-free), the design objects are "fit" or "snapped" onto the FPGA grid. Actually, opening up the second LUT slot in each 6-LUT permits a

better fitting of the design following analytical placement. Consider two LUTs that are close to one another in the floating point placement. If the LUTs use no more than 5 variables, they may be able to pack together in the same 6-LUT, whereas, without the dual-output 6-LUT, they may be forced apart from one another. 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.

Beyond opening up the second LUT output, we alter the cost function to encourage dual-output LUT packing, and increase logic density, reducing the total number of used 6-LUTs. The cost function of the analytical placement and local optimization incorporates a new component as follows:

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

where L is computed based on the utilization of the 6-LUTs and c is a scalar weight. To calculate L, let $LUTS = \{LUT_1 \ \ldots \ LUT_n\}$ represent the set of all dual-output 6-LUT slots that are *used* by a given placement solution. That is, for now, 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,

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

where L_i is the occupancy cost of LUT_i. For L_i there are three different cases to consider:

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

Observe that L_i is the smallest cost for the case of two LUTs being packed together in a single dual-output 6-LUT. The next-smallest cost is assigned to the case of a single LUT placed in a 6-LUT slot and the LUT uses the $O6$ output; recall, $O6$ is the faster LUT output. Finally, the highest cost is associated with the scenario of a single LUT in a 6-LUT slot using slow $O5$ LUT 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 purpose of the cost function in (2) is to favor a reduction in the total number of 6-LUTs being used, and that goal in itself will help to reduce the total number of SLICEs being used. We also extended the cost function to more directly target SLICE count reduction, 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 and S is a new subfunction that targets SLICE count. To compute S, let $SL = \{SLICE_1 \ \ldots \ SLICE_m\}$ represent the set of all SLICEs in the device, including *both* used and unused SLICEs. Then,

$$S = \sum_{SLICE_i \ \epsilon \ SL} S_i, \tag{5}$$

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:

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

S_i is zero if either none or all of the LUTs are utilized in $SLICE_i$. Otherwise, the cost S_i is linearly proportional to the number of vacant LUTs in the SLICE. The highest cost value of $S_i = 1 - 1/8 = 7/8$ occurs when only a single LUT function is present in $SLICE_i$, leaving 7 unused LUT outputs.

3.3 Recovering Performance

To recap, recall that a Virtex-5 SLICE contains four 6-LUTs, four registers, wide-function multiplexers, and carry logic. Figure 10 is a more detailed view of Figure 3, depicting one quarter of a Virtex-5 SLICE. Tracing the path from the $O6$ LUT output, we see that the $O6$ 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.

Flip-flops in sequential circuits are normally edge-triggered. However, the flip-flops in the Virtex-5 SLICE can be configured as level-sensitive latches, where they act as buffers when the clock input is logic-1 and act as memory elements when the clock input is logic-0. If the clock input of a level-sensitive latch is pulled high to a constant 1 at all times, then the latch ceases to be a memory element; it is essentially a buffer and we call this a buffer-latch. It is generally not desirable to use a flip-flop as a buffer-latch, as it is an inefficient use of the latch, and worsens circuit delay. Also, since all flip-flops in a SLICE share the same clock, if the clock is tied to logic-1 for one flip-flop, it must be tied to logic-1 for all flip-flops in the same SLICE. Key to the packing problem is the realization is that there are certain cases where using the $O5$ output of the 6-LUT necessitates configuring the flip-flop as a buffer-latch, which adds additional delay, hurting overall performance. To reduce the impact of the packing on performance, we prevent such cases from happening during dual-output LUT packing. Here, we describe two such cases.

3.3.1 *Use of the Wide-Function Multiplexer.* The $F7$ multiplexer in Figure 10 is called the wide-function multiplexer, as it can be used to combine the outputs of two 6-LUTs to implement a wider 7-input logic function. If the wide-function multiplexer is used unregistered, then its output, marked as the dotted line from $F7$ in Figure 10, uses the $AMUX$ to exit the SLICE. In this scenario, the $O6$ output is driving the wide-function multiplexer, $F7$.

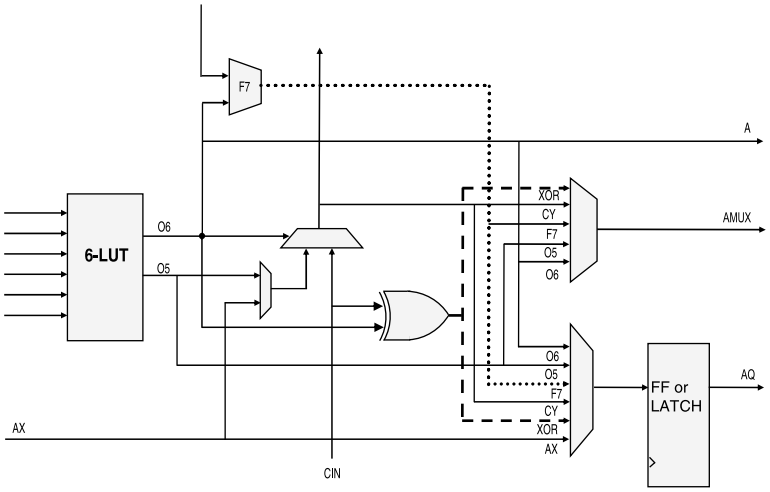

Fig. 10. Details of portion of SLICE.

If a second LUT is placed using the $O5$ output, then either the $O5$ or the $F7$ output can use the $AMUX$, but not both. The other output can only exit the SLICE through the flip-flop. The flip-flop therefore needs to be configured as a buffer-latch. We detect such scenarios before they happen, and prohibit the LUT packing in such cases.

3.3.2 *Use of the Dedicated XOR Gate.* When the dedicated XOR gate is used unregistered, then its output, marked as the bold dashed line in Figure 10, uses the $AMUX$ to exit the SLICE. Similar to the scenario of the wide-function multiplexer, 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 buffer-latch. As above, we prevent such cases from occurring during packing.

3.4 Results

3.4.1 *Area and Speed Performance.* Industrial designs collected from Xilinx customers are used to evaluate the area and speed performance of the dual-output LUT packing. Results for 35 large designs, each containing at least 4000 LUTs, are shown in this article. Each design was placed and routed with a 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. Through this methodology, we can determine whether dual-output LUT packing permits us to reach a higher performance constraint or whether, with dual-output packing, we can no longer meet as high of a constraint. Note that the technology mapper used was the same in all cases; that

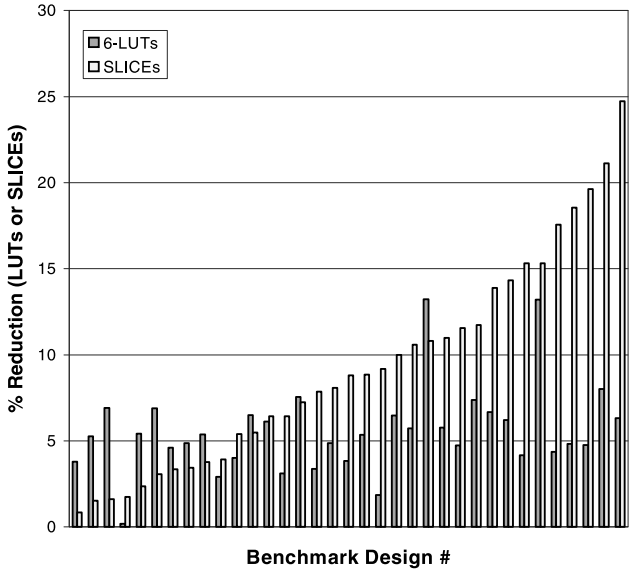

Fig. 11. Area reduction results for LUTs and SLICEs.

is, we did not alter the mapper to favor producing LUTs with fewer inputs for the case of dual-output LUT packing.

We look first at results for logic density. Figure 11 shows the area reduction results for dual-output LUT packing. 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 circuits, the LUT-6 count is reduced by 5.6% and the SLICE count is reduced by 9.5%. It is normally 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, reducing cost.

We were originally surprised by the large SLICE count reduction in comparison to the LUT count reduction, which we found counter-intuitive. However, Figure 12 illustrates how SLICE reduction can exceed LUT reduction and helps to explain the results in Figure 11. Figure 12(a) shows 2 SLICEs with 3 occupied LUTs, prior to a packing. In Figure 12(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.

We now turn to the speed performance results, shown in Figure 13. One data point is presented for each circuit. The horizontal axis represents the percentage *reduction* in speed performance for each circuit, and the vertical axis represents the percentage reduction in SLICE count. Observe that some circuits improved in speed performance (negative values) and some circuits degraded (positive values). On average, speed performance degraded by 1.6%

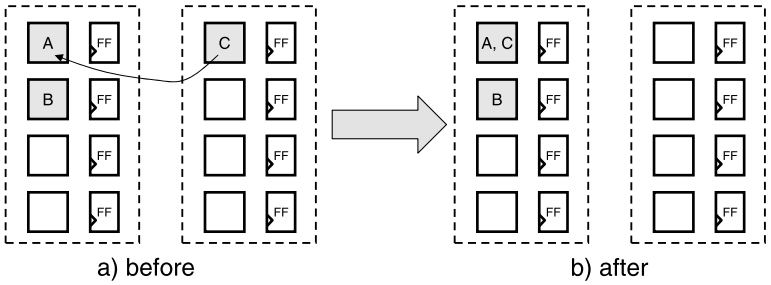

Fig. 12. Example of LUT and SLICE reduction.

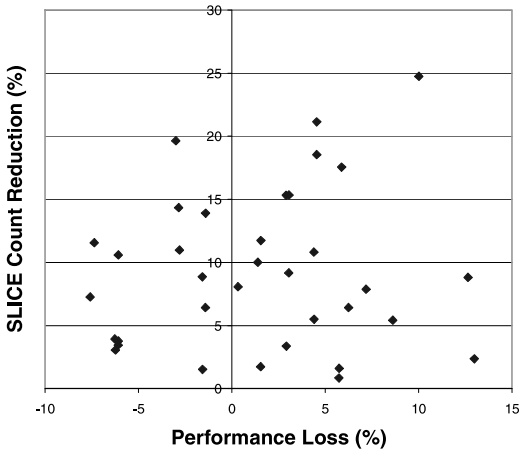

Fig. 13. SLICEs reduction (%) versus performance reduction (%).

across all circuits. In the figure, 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 9.5% SLICE reduction, at a cost of about 1.6% speed performance, on average. As we demonstrate below, other algorithm parameter tunings produce different area/performance results, however, we believe the tuning shown here is a trade-off that will be desirable to customers.

Regarding an explanation for the "noisy" speed performance results, we believe several factors are at play. First, the use of two LUT outputs contributes to higher packing density, producing a more compact placement, having shorter wirelengths and higher performance, as is apparent for some circuits. On the other hand, packing increases pin density in the layout, and higher pin density worsens routing congestion and performance for some circuits, since more signals must compete for low-delay paths in the routing fabric. Furthermore, 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.

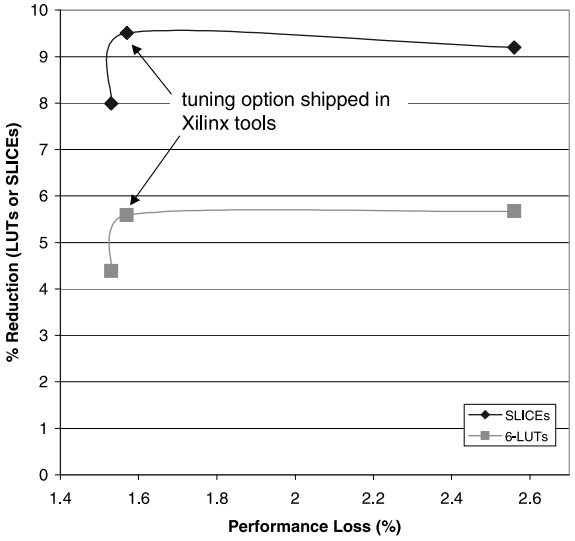

Fig. 14. SLICEs and 6-LUTs reduction (%) versus performance reduction for three possible tunings.

Lastly, some of the performance swings are certainly due the "instability" inherent in any FPGA layout system based on heuristic algorithms, which must make arbitrary decisions in some cases, for example to break ties in sorted lists. The performance consequences of the logic block packing are complex and warrant further investigation.

In addition to the area/performance results presented above, we considered alternative tunings where, essentially, we changed the weights c and d in (4), changing the importance given to area reduction relative to wirelength and timing. We made two additional runs across all circuits: one run with c and d reduced, and a second run with c and d increased. The results are shown in Figure 14, where average performance loss (across all designs) is shown on the horizontal axis, and two curves represent the average percentage reduction in LUTs and SLICEs. The middle points (at a performance loss of 1.6%) represent the tuning results presented above and shipped to Xilinx customers. The two alternate tunings illustrate that the weight parameters in (4) are quite effective in exploring the area-performance trade-off space. Decreasing the weights c and d in (4) produces the leftmost data points in the figure, representing compromised area and only slightly better performance. Increasing c and d yields the rightmost data points in the figure, reflecting higher performance loss with minimal additional benefit to area reduction.

Finally, 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. Such aggressive packing uses no placement information to decide on suitable LUT packing candidates, and aims for area reduction above all else, even permitting a connected LUT/FF pair to be placed apart from each other in

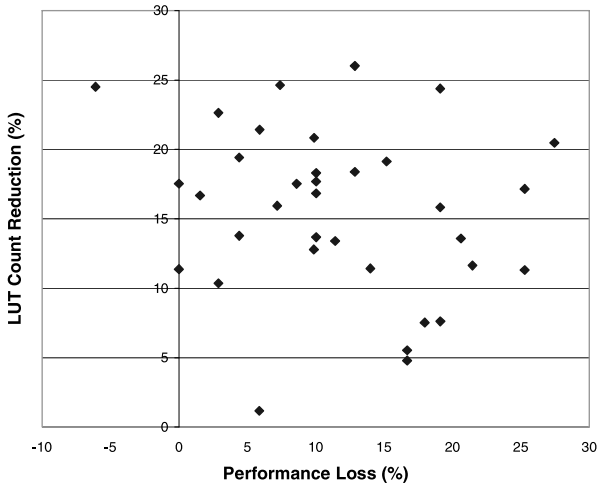

Fig. 15. 6-LUTs reduction (%) versus performance reduction (%) for aggressive packing.

different SLICEs. Figure 15 shows the results for this more aggressive packing. As shown, LUT count is reduced considerably beyond that achieved by the approaches described above. In particular, LUT count is reduced by 15.8% across the suite of circuits. However, the increased packing density comes at a significant performance cost of about 12%, on average. A 12% performance decrease corresponds to one speed-grade—a speed hit 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 [Shang et al. 2002]. The packing techniques described above lead to lower logic usage, potentially reducing both static and dynamic power. Also, the reduced LUT and SLICE counts may permit the placer to achieve a "tighter" layout with shorter wirelengths, reducing interconnect usage and power.

We investigated power using 19 industrial designs collected from Xilinx customers and 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.

Figure 16 shows the percentage improvement in leakage power owing to 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),

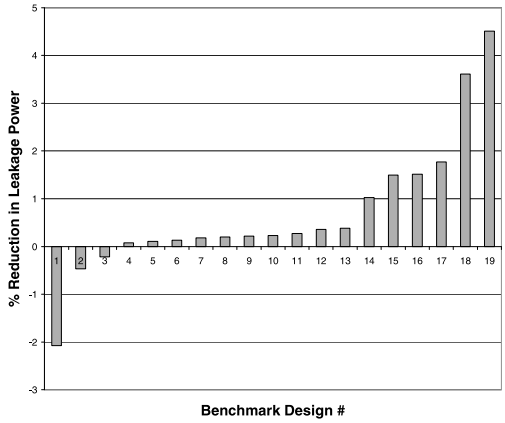

Fig. 16. Leakage power reduction results.

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 [Gupta et al. 2007].

4. PACKING FOR BLOCK RAMS

We now turn from packing for logic blocks to the problem of packing for large IP blocks. We describe two IP block packing algorithms that contribute to increased performance and density. The first algorithm is a block RAM and DSP packing optimization that takes advantage of the property that FPGA designs use block RAMs and DSPs in regular and predictable patterns and also takes advantage of the block RAM and DSP tile layout on the FPGA. The second packing algorithm packs chains of block RAMs together to increase memory density. It is worth mentioning that memory density is an important FPGA metric because customer designs are commonly limited by block RAM availability.

4.1 Block RAMs and DSPs

DSP circuits commonly use both block RAM and DSP blocks together in a number of predictable patterns. 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 subcircuit referred to a "tap."

Each filter tap multiplies a coefficient word by a data word and accumulates the result. The multiply-accumulate function of a tap can be implemented in a single DSP block, and the storage of coefficients and data can be implemented using block RAMs. In Virtex-5 FPGAs, each 18 Kb RAM and the DSP it drives can implement a single tap of an FIR filter.

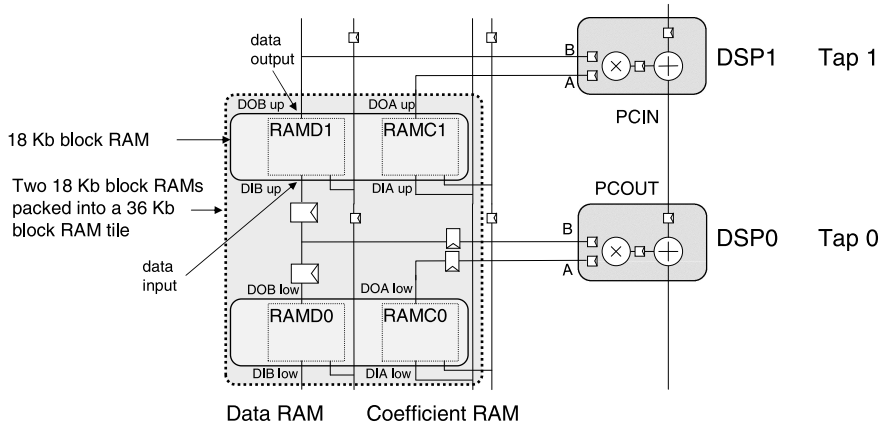

Fig. 17. 2-tap FIR implementation.

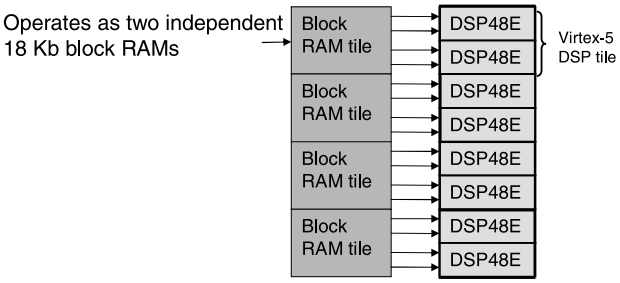

Fig. 18. Placement of FIR implementation.

Figure 17 illustrates a 2-tap FIR filter. The data and coefficient RAMs in this figure are 9 Kb RAMs or less and have been mapped into two 18 Kb block RAMs which can then be packed into a single 36 Kb Virtex-5 block RAM tile. The FIR filter data is mapped to port A of the block RAMs; the FIR filter coefficients use port B of the block RAMs. In this case, the two 18 Kb block RAMs are in true dual-port mode. The registers in the diagram represent registers that the designer may or may not have added.

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 Figure 18, which shows an 8-tap FIR filter. This is the alignment our algorithm produces. Without the algorithm, the 18 Kb block RAMs that ideally should share a 36 Kb block RAM tile might be packed into different 36 KB block RAM tiles, creating an unnatural alignment, as illustrated in Figure 19, and leading to worse performance results.

The second FIR filter use case we target is shown in Figure 20, 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

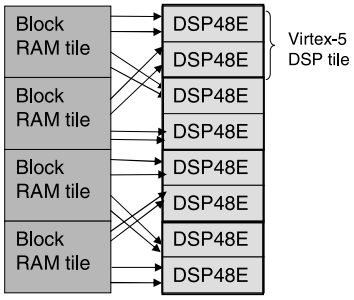

Fig. 19. Placement of FIR implementation with poor placement alignment.

Figure 17 and hence more precision) or if one of the data and coefficient block RAMs for a tap is greater than 9 Kb and hence cannot be mapped in a single 18 Kb block RAM (which is the case for the RAMs in Figure 20). 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 Figure 21, 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 shown below. The input to the algorithm is a list of 18 Kb block RAMs and a list of DSP blocks. The output of the algorithm is a list of 18 Kb block RAMs that have been packed together into 36 Kb block RAM tiles.

(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:

```
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
```

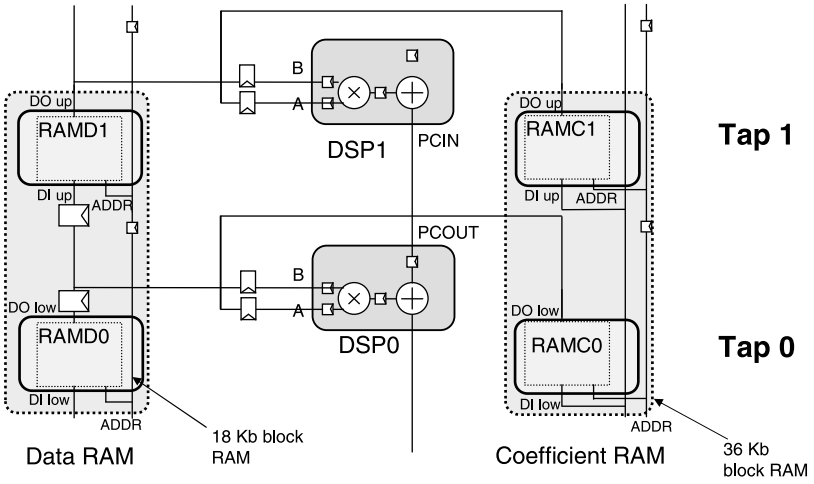

Fig. 20. Alternate 2-tap FIR implementation.

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 Figure 22.

Our block RAM packing traverses the netlist to identify such chains and packs them accordingly. The input to the algorithm is a list of 18 Kb block RAMs and the output is a a list of 18 Kb block RAMs that have been packed together into a single 36 Kb block RAM tile. 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 mentioned above, and/or block RAM chains. The designs have between 30 and 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.

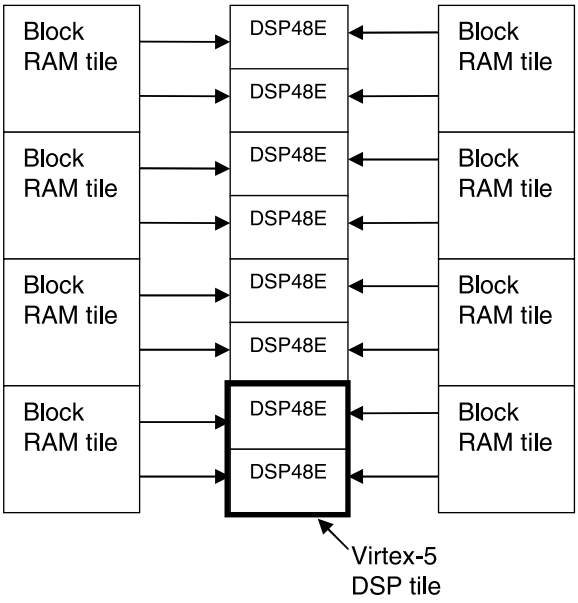

Fig. 21. Placement of alternate implementation.

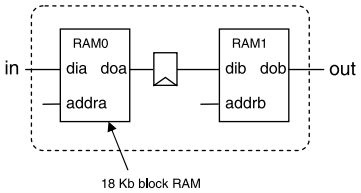

Fig. 22. Block RAM chaining example.

The results can be seen in Table I, 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.

5. CONCLUSIONS

Packing for commercial FPGAs represents a more complex problem than typically addressed in the academic literature, with more opportunities for optimization, yet with more constraints. Packing has a significant impact on design speed, area and power, and has recently been combined with other phases of the tool flow, including placement and technology mapping. Xilinx Virtex-5 FPGAs have logic blocks containing dual-output LUTs, capable of implementing one large logic function, or two smaller logic functions. The IP blocks in today's FPGAs present unique packing problems, not yet considered in the literature. In this paper, we described approaches for logic and IP block packing for Virtex-5 FPGAs. Our dual-output LUT packing lowers the number of used SLICEs by 9.5%, on average, with a 1.6% performance hit. The block

Table I. 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%

RAM packing is targeted towards DSP circuits and improves performance by 11%, on average. Both optimizations are available in the Xilinx ISE™ 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, perhaps using a recently published mapping algorithm that produces LUTs with fewer used inputs [Jang et al. 2008]. 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.

ACKNOWLEDGMENTS

The authors thank Brad Taylor, Rajat Aggarwal, and Kamal Chaudhary for their 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.

REFERENCES

AHMED, T., KUNDAREWICH, P., ANDERSON, J., TAYLOR, B., AND AGGARWAL, R. 2008. Architecture-specific packing for Virtex-5 FPGAs. In *Proceedings of the ACM International Symposium on Field-Programmable Gate Arrays*. 5–13.

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

BETZ, V. AND ROSE, J. 1997. Cluster-based logic blocks for FPGAs: Area-efficiency vs. input sharing and size. In *Proceedings of the IEEE Custom Integrated Circuits Conference*. 551–554.

CHEN, D. AND CONG, J. 2004. Delay optimal low-power circuit clustering for FPGAs with dual supply voltages. In *Proceedings of the ACM/IEEE International Symposium on Low-Power Electronics and Design*. 70–73.

DEHKORDI, M. AND BROWN, S. 2002. The effect of cluster packing and node duplication control in delay driven clustering. In *Proceedings of the IEEE International Conference on Field-Programmable Technology*. 227–233.

EISENMANN, H. AND JOHANNES, F. 1998. Generic global placement and floor planning. In *Proceedings of the ACM/IEEE Design Automation Conference*. 269–274.

GUPTA, S., ANDERSON, J., FARRAGHER, L., AND WANG, Q. 2007. CAD techniques for power optimization in Virtex-5 FPGAs. In *Proceedings of the IEEE Custom Integrated Circuits Conference*. 85–88.

HASSAN, H., ANIS, M., AND ELMASRY, M. 2005. Lap: A logic activity packing methodology for leakage power-tolerant FPGAs. In *Proceedings of the ACM/IEEE International Symposium on Low-Power Electronics and Design*. 257–262.

HUTTON, M., SCHLEICHER, J., LEWIS, D., PEDERSEN, B., YUAN, R., KAPTANOGLU, S., BAECKLER, G., RATCHEV, B., PADALIA, K., AND ET. EL. 2004. Improving FPGA performance and area using an adaptive logic module. In *Proceedings of the International Conference on Field-Programmable Logic and Applications*. 135–144.

JANG, S., CHAN, B., CHUNG, K., AND MISHCHENKO, A. 2008. Wiremap: FPGA technology mapping for improved routability. In *Proceedings of the ACM International Symposium on Field-Programmable Gate Arrays*. 47–55.

KUON, I. AND ROSE, J. 2007. Measuring the gap between FPGAs and ASICs. *IEEE Trans. Comput.-Aid. Des. Integr. Circ. Syst. 26*, 2, 203–215.

LAMOUREUX, J. AND WILTON, S. 2003. On the interaction between power-aware FPGA CAD algorithms. In *Proceedings of the IEEE International Conference on Computer-Aided Design*. 701–708.

LIN, J., CHEN, D., AND CONG, J. 2006. Optimal simultaneous mapping and clustering for FPGA delay optimization. In *Proceedings of the ACM/IEEE Design Automation Conference*. 472–477.

MARQUARDT, A., BETZ, V., AND ROSE, J. 1999. Using cluster based logic blocks and timing-driven packing to improve FPGA speed and density. In *Proceedings of the ACM International Symposium on Field-Programmable Gate Arrays*. 37–46.

SCHABAS, K. AND BROWN, S. 2003. Using logic duplication to improve performance in FPGAs. In *Proceedings of the ACM International Symposium on Field-Programmable Gate Arrays*. 136–142.

SHANG, L., KAVIANI, A., AND BATHALA, K. 2002. Dynamic power consumption of the Virtex-2 FPGA family. In *Proceedings of the ACM International Symposium on Field-Programmable Gate Arrays*. 157–164.

SINGH, A. AND MAREK-SADOWSKA, M. 2002. Efficient circuit clustering for area and power reduction in FPGAs. In *Proceedings of the ACM International Symposium on Field-Programmable Gate Arrays*. 59–66.

VISWANATHAN, N., NAM, G.-J., ALPERT, C., VILLARRUBIA, P., REN, H., AND CHU, C. 2007. RQL: Global placement via relaxed quadratic spreading and linearization. In *Proceedings of the ACM/IEEE Design Automation Conference*. 453–458.

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

Received June 2008; revised September 2008, December 2008; accepted January 2009