Packing a Modern Xilinx FPGA Using RapidSmith

Travis Haroldsen, Brent Nelson, and Brad Hutchings
Department of Electrical and Computer Engineering
NSF Center for High-Performance Reconfigurable Computing (CHREC)
Brigham Young University, Provo, UT 84602
Email: travisdh@byu.edu, brent_nelson@byu.edu, brad_hutchings@byu.edu

Abstract—**Academic packing algorithms have typically been limited to theoretical architectures. In this paper, we describe RSVPack, a packing algorithm built on top of RapidSmith to target the Xilinx Virtex 6 architecture. We integrate our packer into the Xilinx ISE CAD flow and demonstrate our packer tool by packing a set of benchmark circuits and performing routing and timing analysis inside ISE.**

I. Introduction

Academic research into packing algorithms for modern Xilinx architectures has been hindered by the lack of CAD infrastructure for these devices. RapidSmith [1], through the use of the Xilinx Design Language (XDL), has allowed researchers to interweave novel CAD approaches into the standard Xilinx ISE tool flow. With the improvements in RapidSmith 2 [2], RapidSmith now provides the infrastructure requisite to exploring new packing algorithms and integrating them into the ISE tool flow.

In this paper, we present the RSVPack algorithm for packing the Xilinx Virtex 6 FPGA. This algorithm satsifies the many requirements imposed by the Virtex 6 architecture. It is built on top of RapidSmith 2 and, combined with a simulated annealing placer, is interwoven into the ISE tool flow. This allows it to accept a high-quality synthesized netlist of a design from ISE, pack and place the netlist and return the placed design to be routed and analyzed inside the ISE tools. We demonstrate our packer and flow by packing a set of benchmark circuits and analyzing their critical paths inside ISE. Lastly, we describe the challenges associated with packing a modern Xilinx FPGA and the approaches we used to overcome these challenges.

The rest of this paper is outlined as follows. Section II provides background on the structure of the Xilinx Virtex 6 FPGA architecture and reviews the popular VPR tool suite and previous efforts to utilize it to represent a Xilinx architecture. Section III describes the RSV flow and RSVPack algorithm developed in this work. Results of the algorithm and flow are presented in Section IV. Section V concludes this paper.

II. Background

A. Xilinx Architecture

Xilinx structures their FPGAs into three levels of hierarchy. The elements at each level, from top to bottom, are called *tiles*, *primitive sites* (or sites) and *Basic Elements of Logic* (BELs).

This work was supported in part by the I/UCRC Program of the National Science Foundation within the NSF Center for High-Performance Reconfigurable Computing (CHREC), Grant No. 1265957.

1) Tiles: Xilinx architectures are organized as a 2 dimensional grid of tiles with columns alternating between logic performing tiles (CLBs, DSPs, Block Ram) and one or more associated interconnect tiles, also called switch boxes (see Figure 1). In general, each column consists of a single type of tile replicated from the bottom to the top of the chip. With the exception of a few dedicated carry chain wires, all routing between the logic performing tiles in the device is accomplished through the switch boxes.

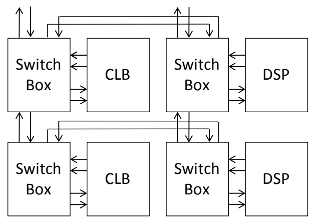


Fig. 1. Organization of tiles in a Virtex 6 architecture

The switch boxes are made up of wires connecting to other switch boxes (general routing), wires connecting to the logic performing tiles and *programmable interconnect points* (PIPs) that optionally connect different wires inside the switch box. The PIPs provide the routing flexibility of the FPGA.

Unlike other architectures, Xilinx FPGAs do not contain an explicit routing crossbar inside the logic performing tiles. Instead, outputs from the logic performing tiles must all exit to the switch box, even if their nets immediately return to the same tile they came from. A small set of PIPs inside the switch boxes, called *bounce PIPs*, provide local routing flexibility by connecting the tile outputs to inputs on the same tile without leaving the switch box. Bounce PIP connections are not fully populated and some outputs may require bouncing through several of these PIPs before returning to the tile. If a net requires multiple bounce PIPs to route, the rest of the design is left with fewer routing resources making the circuit harder to route and leading to slower clock speeds.

2) Sites: Logic performing tiles each contain one or more sites. The sites in turn are composed of a number of BELs connected by muxes. In XDL, many of the sites, such as the DSPs and BRAMs, consist of a single BEL with a set of configuration properties. The configuration properties of these BELs are determined by the synthesis and technology mapping tools and provide little to no packing flexibility. The most common sites, the *slices*, however, have their contents exposed and can be manipulated during the packing phase.

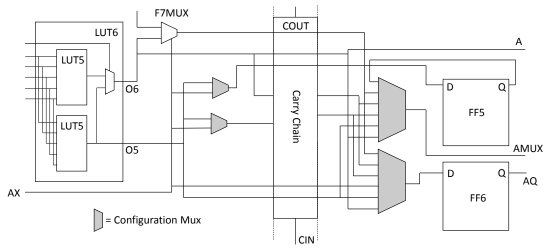


Fig. 2. Simplified view of Virtex 6 Logic Element

The slice contains the LUT/flip-flop pairs in the FPGA. In modern Xilinx FPGAs, the slices also contain other complex structures in addition to the LUTs and flip-flops. In Virtex 6 devices, two slices are combined to make a single CLB tile. Each slice in turn contains four similar, but not identical, units of closely coupled resources we refer to as *logic elements* (LEs) (see Figure 2). The features in each LE include:

- A fracturable 6 input LUT which can be treated as two 5 input LUTs sharing the lower five pins. Some slices, called SLICEMs, support configuring the LUTs as LUTRAMs which can be used as either small RAMs or 16 bit shift registers (SRLs). The SLICEMs also contain additional routing to combine the LUTRAMs into larger or more complex RAM or shift register types.
- An F7 or F8 mux, depending on the LE in the slice, to combine either 2 or 4 LUTs together into a 7 or 8 input logic equation. Each slice contains two F7 muxes, which feed into one F8 mux. These muxes prevent the slice from being represented as four identical LEs.
- A carry logic component which is shared between all LEs in the slice. This component provides additional circuitry which can be used in conjunction with the LUTs to implement chains of adder circuitry. The carry component supports a carry in and carry out signal to chain multiple slices into larger adders.
- Two flip-flops, one with a D input configured to come from the O5 LUT output or the AX input pin, and the other shared between the different components in the LE.
- A dedicated output (A) for the O6 LUT output and an AMUX output shared between the different components in the slice. This second output is frequently a point of contention when trying to fill the slice during packing.

B. RapidSmith 2

RapidSmith [1] is a CAD framework supporting interoperability with Xilinx devices. RapidSmith supports device representation of Xilinx devices and a netlist of packed Xilinx components. Using the XDL language, RapidSmith allows users to import a placed design, make modifications to the design and return the modified design to the Xilinx ISE tools. This has enabled research into novel CAD approaches targeting commercial devices.

In RapidSmith 2 [2], the device representation is extended to expose the internal circuitry of the sites. Additionally, RapidSmith 2 converts the site-level netlist used in RapidSmith into a netlist of BEL-level components, referred to as cells. This netlist can come from an ISE synthesized design and the placed netlist returned to ISE for routing. These two additions enable packing the design.

C. Verilog To Routing

1) Review: Verilog-to-Routing (VTR) [3], formerly Versatile Place and Route (VPR), is an academic tool for FPGA architecture and CAD research. VTR provides users the ability to describe a variety of FPGA architectures and a suite of CAD tools to implement designs onto these architectures. With recent releases, VTR now supports architectures with complex structures such as carry chains and includes front-end synthesis and the AAPack packing algorithm to pack these clusters.

VTR allows users to describe custom architectures. The logic blocks (CLBs for example) are described in a hierarchical manner, with each level of hierarchy describing the different configuration modes and the components and interconnects in the block for each mode. This permits a wide variety of architectures to be described.

The AAPack packing algorithm [4] is a greedy seed-based algorithm for packing designs onto VTR's architectures. AAPack focuses first on being compatible with the large number of complex cluster structures describable by VTR's architectural descriptions. The algorithm works by seeding a cluster with a single *cell*, choosing the type for the cluster and then greedily filling the cluster with related cells. Cells are absorbed into the cluster, usually one at a time, at which point the algorithm must confirm that the cluster has a feasible route. As verifying a route exists for the cluster can be a slow process, AAPack uses a set of techniques such as speculative packing and pin counting to avoid performing a full routing feasibility check after packing each cell when possible. These optimizations are described in [5].

2) Virtex 6 in VTR: Hung et al., in [6] and [7] describe work towards representing a Virtex 6 device in VTR and using the VTR CAD tools to implement circuits on the device. This required creating an architectural description of the Virtex 6 CLB inside VTR. While the authors were able to represent and pack the slice structures in VTR, their modeled slice lacked important features including:

- the F7/F8 muxes (see Figure 2)
- the LUTRAM and SRL capabilities of the SLICEMs
- the slice clock enable and set/reset signals

VTR also uses a different cluster routing structure than Xilinx FPGAs. VTR expects a crossbar in front of the clusters. As previously noted, Xilinx instead uses bounce PIPs to provide the routing flexibility within its tiles. Hung's work does not represent the bounce PIPs, instead representing them as a mostly populated crossbar. As such, Hung's work is unable to distinguish between reentrant paths that use a single or no PIPs and less optimal paths that require several PIPs.

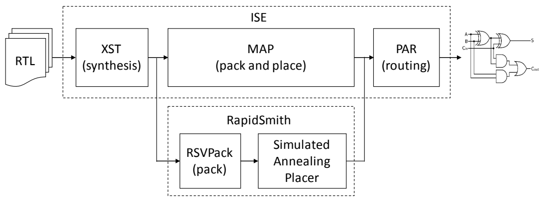


Fig. 3. ISE and RSV Flows

While these device features could possibly be described in VTR's architectural model, they are not supported by VTR's front-end synthesis and therefore are unavailable to the rest of the VTR flow. The lack of these features masks many additional challenges inherent in packing commercial CLB structures such as verifying that incompatible features are not used within the same slice. Further, Hung found that the lower quality synthesis supplied to VTR itself accounted for a significant deterioration in the average critical path delay [7].

III. Approach

A. RSVPack Overview

In traditional FPGA CAD flows, a packing step is employed to group individual components of a technology-mapped design into relatively-placed structures called clusters. This step acts as a localized placement phase, bundling closely related logic together to shorten routing paths between elements and to reduce the total amount of global routing required for the circuit. This, along with abstracting away the constrained routing environment and other complexities of the CLB internals, helps simplify the subsequent global placement.

The primary objective of RSVPack is to be intregratable into the Xilinx ISE CAD flow. This requires being able to accept an ISE synthesized and technology-mapped netlist, accurately representing the tile structures in the device and creating a set of clusters that can be returned to the ISE tool flow and realized in a bitstream.

In our RSV CAD flow (the bottom path in Figure 3), the packer accepts an XST synthesized netlist and produces a set of clusters where each cluster consists of a group of relatively-placed cells. Ideally, we would fully isolate the packer from the ISE flow and allow ISE to place our packed design. However, since the Virtex 5 family, Xilinx ISE tools no longer accept packed but unplaced designs but merge packing and placement into a single *MAP*. The placed circuit is then routed by *PAR*. To integrate our packed circuit back into the ISE tool flow, we are required to place our packed circuit ourselves. We use a simulated annealing placer using a Half-Perimeter Wire Length cost model to place the packed netlist. The placed design can then be returned to ISE using the XDL language and handed to PAR to route the design.

The RSVPack algorithm is specifically targeted toward Virtex 6 FPGAs. By focusing on a particular architecture, we are able to identify and address the specific challenges and requirements associated with the targeted architecture. It also allows the algorithm to focus on supporting the device's unique feature set and to optimize toward generating high-quality circuits for the architecture. Due to similarities between successive generations, the algorithm should be easily portable to more recent Xilinx families.

The packing algorithm takes as input an unplaced, technology-mapped netlist. In this particular work, the input is expected to come from a Xilinx generated netlist. This netlist will thus make use of a wide variety of features supported by the Virtex 6 architecture including LUTRAMs, shift-registers and the F7 and F8 muxes. The netlist that comes from Xilinx already identifies which resources should be mapped to special purpose hardware, for example, whether a RAM should be implemented using hard block RAMs or as LUTRAMs. The role of this packing algorithm is therefore solely to group these components into clusters.

The RSVPack algorithm uses a seed-based, greedy heuristic similar to [5][8][9]. This heuristic leads to a straightforward implementation and allows for simple testing to ensure the generated clusters are legal. However, as is often the case with greedy heuristics, decisions are only locally optimal possibly leading to non-optimal global packing or even to impossible configurations. Care, therefore, must be taken in creating selection criteria that acknowledge the global nature of the problem and avoid using resources that will be required by future clusters. Due to a lack of complete timing information for Xilinx devices, the packer currently does not support any timing-driven features.

B. Packing Units

The packer produces clusters made from a single logical tile combined with its associated switch boxes, hereafter referred to as *packing units*. The tile hierarchy is chosen over the site hierarchy, which itself contains all of the BELs and inflexible interconnect necessary for creating a valid packing, because some tiles contain multiple sites sharing the same switch box (such as the CLBs which contain two slices). Including the switch boxes in the packing units is important as the switch box PIPs determine whether a site output can reach another input in the same tile without leaving the local routing structures. Choosing locations for cells that allow the nets to use only local routing resources and thereby minimize the number of PIPs required to route them is likely to lead to a circuit with less routing contention and shorter routes.

The RSVPack algorithm works on a flattened packing unit, as opposed to AAPack's hierarchical model. This flat model more closely aligns with the device representation presented by XDL from which we create our models. This allows the models of the clusters to be generated based on the XDL. It also acknowledges that the individual LEs are not self-contained structures but overlap and share fast connections between one another that do not fit cleanly within a hierarchical structure.

C. RSVPack Algorithm

1) Filling the Clusters: RSVPack begins each cluster by choosing a seed from the remaining unpacked cells. Currently RSVPack chooses the remaining cell with the most external pins. This seed could potentially seed valid clusters based on different packing units – for example, a LUT cell could be the seed for clusters made from a CLBLL (a CLB with two SLICELs) or a CLBLM (a CLB with a SLICEL and a SLICEM). Rather than predicting which packing unit will yield the best cluster given the seed, RSVPack creates clusters from all possible packing units for the seed and chooses the highest quality cluster from the resulting valid clusters. While this approach can lead to longer pack times, the increase is not prohibitive as there are generally very few packing units which contain any BELs the seed can be placed at.

For each packing unit to be explored, the algorithm fills the packing unit by selecting an unpacked cell, choosing a BEL in the cluster, packing the cell at the chosen BEL and repeating until the cluster is considered complete. Currently a cluster is considered complete when either it is full or no more connected cells are available to pack into the cluster.

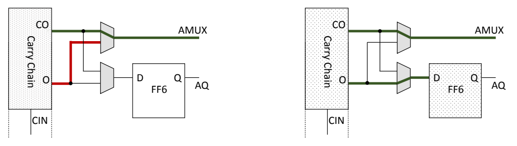
RSVPack uses the same attraction function as non-timing driven AAPack – namely, it prioritizes cells that share many nets with the current cluster. In contrast to AAPack, currently the selection criteria limits its search to cells that are a single hop from the cluster. With a cell selected, the packer identifies a BEL in the packing unit on which to place the cell. The BEL selection criteria seeks to minimize the number of routing resources required to route the cluster. This includes prioritizing BELs which fully absorb nets inside a site and reducing the number of bounce PIPs in the switch box required to route the nets. This approach leads to shorter routes and less contention for valuable routing resources.

2) Cluster Validation: After each cell is packed into the cluster, the algorithm ensures that the cluster is in or can be made into a valid state. Validation is checked through a series of tests. These include the following tests:

- **Routing Feasibility** Ensures that the cluster is packed in a manner that can be routed. This includes checking that connections within the cluster are routable and that connections coming from or to the cluster can reach their intended destinations outside the cluster. To perform this check in a timely manner we use a table lookup-based validation algorithm described in section III-D.
- **Carry Chain Preservation** Ensures carry chains between clusters are maintained. Since the carry chains span multiple clusters and have limited routing flexibility, the packer must check that all elements in the chain are packed such that the carry chain is preserved. Section III-E describes the requirements in more detail.
- **Control Sets Consistency** Some properties of the clusters, such as the flip-flop synchronicities in the slices, are configurable at the site level instead of being configurable for each BEL. These *control sets* require that all cells in the site share the same configurations. This check ensures that the packer does not allow cells requiring conflicting properties to be placed in the same site.
- **LUTRAM Validation** When the input netlist is flattened, the hierarchical LUTRAM cells are broken into individual pieces. The Virtex 6 architecture has requirements about where each piece of the LUTRAM can be placed in the slice. This check ensures that the LUTRAM cells are packed into the same cluster and at compatible locations.

If the cell/BEL combination being evaluated is determined to lead to an illegal cluster by any of the checks, the cell is unpacked from its location and another BEL is chosen for the cell. This process continues until either a valid BEL is identified for the cell or all possible locations are exhausted.

3) Conditional Lookahead: While filling clusters, it is possible for the cluster to enter a state where it is no longer valid in its given configuration, but may return to being valid by absorbing other cells. Rather than invalidating this configuration, the algorithm enters into a conditional mode.



(a) (b)

Fig. 4. (a) Adding the carry chain leads to an unroutable cluster causing RSVPack to enter conditional mode. (b) Upon absorbing the flip-flop, the cluster is again routable.

For example, upon adding a carry chain element using both the O and CO outputs to a cluster (Figure 4a), there are insufficient outputs to route the cluster. This causes RSVPack to enter conditional mode. Upon adding the flip-flop being driven by the O output to the cluster (Figure 4b), the cluster is again routable and RSVPack returns to its normal mode.

While in conditional mode, the algorithm continues to add cells to the cluster under the assumption that the cluster will return to a valid configuration. If RSVPack is unable to return the cluster to a valid state by absorbing more cells, it will roll back the cluster to its previous checkpoint, invalidate the cell/BEL pair that caused it to enter conditional mode and resume normal operation. As returning to a valid state may require adding more than one cell to the cluster, the checkpoint and roll back mechanism operates in a recursive manner. AAPack contains a similar speculative mode.

The recursive nature of the conditional mode mechanism can potentially lead to exponential run times. To limit the time spent in conditional mode, RSVPack guides the cell selection process to cells that will quickly bring the cluster to a valid or invalid state. These cells are identified by the validation tests.

D. Routing Feasibility

The Virtex 6 slice contains a highly-specialized routing structure. The clusters generated by the packer must be capable of being routed using that structure. A simple method to verify that a cluster is routable is to simply attempt to route the cluster. If a valid route is found, the cluster is routable; otherwise, the cluster in its current state is not routable.

The initial approach we used to validate routing used the popular PathFinder routing algorithm [10]. Although PathFinder can identify routable clusters, we found it to be too slow to be used inside the critical loop of an exhaustive-search packing algorithm. It is particularly inefficient at identifying unroutable circuits, being only able to infer the circuit is

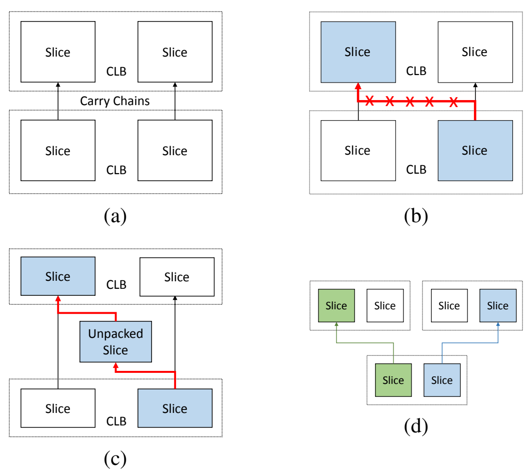


Fig. 5. (a) Each slice in a CLB has a single carry chain. (b) An invalid carry chain. (c) An unpacked slice hiding a broken carry chain. (d) Two merged carry chains driving different clusters.

unroutable after failing several iterations of attempting to negotiate the congested circuit. With the search used by our algorithm, we found that an unroutable circuit was the most common case when adding a cell to the cluster.

Instead of using PathFinder, RSVPack uses a comparison-based method to check routability. This approach quickly compares the current cluster's configuration against each possible routing configuration for the packing unit (as dicated by the various routing mux configurations in the packing unit). The possible routing configurations are each enumerated as entries in a table. As cells are added to a cluster, the resulting cluster is compared against all of the table's entries to see if the cluster's connectivity is compatible with any of those entries. If one or more compatible entries exist, the cluster is routable. If none exist, the cluster is not routable.

Using this algorithm, we saw an order of magnitude speed up over using a re-entrant version of PathFinder. This approach also runs in about the same amount of time for both routable and unroutable clusters. This has allowed us to perform full feasibility checks of the cluster routing after adding each cell.

E. Handling Carry chains

Carry chains provide a fast, dedicated connection between arithmetic units. As the arithmetic chains can be long, a 32-bit adder requires a chain of 8 CLBs, and can often be the critical path in a design, utilizing these paths is critical for creating high-quality circuits.

Xilinx uses carry chains in their DSPs, BRAMs and CLB tiles. In the case of the CLB, both slices contains their own distinct carry chain connected to their corresponding slice in an adjacent CLB (see Figure 5a). To use the carry chain feature, the cells involved in the operation must be packed so that the chain is preserved when going between different clusters. If not packed correctly (see Figure 5b), then the circuit could suffer significantly reduced quality or not even be realizable.

The routing feasibility checker previously described checks that adjacent carry chains are packed, or can be packed, in locations that will preserve the carry chain routing. However, the routing feasibility checker only performs a local validation and situations can arise that lead to invalid packings that the routing feasibility checker is insufficient to prevent.

Two such cases are shown in Figures 5c and 5d. In the first situation (5c), two non-adjacent cells in the same carry chain are packed before all of the in between cells are packed. The localized nature of the routing feasibility check is insufficient to detect that this situation will ultimately lead to an unroutable circuit. In the second situation (5d), two cells from different carry chains are packed into the same CLB. When packing the adjacent carry chain cells, the routing feasibility check only validates the current cluster and not the previously created cluster. It is therefore possible that the two adjacent carry chain cells end up in different clusters resulting in an invalid circuit.

RSVPack uses a couple different strategies to ensure carry chains are properly handled. First, RSVPack will not pack any cell involved in a partially-packed carry chain unless it is adjacent to a previously packed cell. This prevents packing carry chain cells until their required positioning can be determined from their previously-packed neighbors. Second, when two carry chains' cells are packed into the same CLB as in the lower half of Figure 5d, RSVPack merges the adjacent carry chain cells to ensure that they are later packed together.

IV. Results

The goal of this work is to present a packing algorithm that can be integrated into the Xilinx ISE tool flow. As part of this, we have applied our packing to creating clusters for five designs. The five designs are:

- a viterbi decoder
- a 128 bit AES cipher
- a Xilinx Microblaze processor
- a 1024 tap FIR filter
- a double precision floating point divider

Resource utilizations for each design are presented in Table II. Each design was successfully implemented on an xc6vlx75tff484 part using the RSV flow.

We use the standard ISE flow as a reference point. The presented results are obtained from averaging 100 runs using different placer seeds. Figure 6 shows the average run times for both flows. As mentioned, ISE combines packing and placement into a single step in their MAP executable. Therefore, times for the placer are included for a more representative comparison. PAR times are also included as the quality of the packing and placement will influence the time required to route the design. With the exception of the Microblaze processor, all circuits are implemented with a clock constraint set to the circuit's maximum achievable clock rate. The non-timing driven RSVPack ignores this timing constraint, but forwards the constraint on to PAR.

For the tested designs, RSVPack on average packed at a rate of 74.0 cells per second. In general, the run times of RSVPack and the placer were comparable to the Xilinx MAP runtime. In each design, PAR took slightly longer to route the design coming from RSVPack than the design coming from map.

TABLE I
PACKER RESULTS

	Num Slices			Exposed Nets			Minimum Period			Total Wire Length		
	ISE	RSV	$\frac{RSV}{ISE}$	ISE	RSV	$\frac{RSV}{ISE}$	ISE	RSV	$\frac{RSV}{ISE}$	ISE	RSV	$\frac{RSV}{ISE}$
Viterbi Decoder	2621	2482	0.9	12534	13294	1.1	6.14	10.8	1.8	183054	203486	1.1
AES Cipher	3818	5755	1.5	12916	16599	1.3	3.47	6.53	1.9	236920	495418	2.1
Microblaze	4762	5422	1.1	19620	22302	1.1	9.75	15.1	1.5	276630	491350	1.8
FIR filter	2245	2284	1.0	18502	19262	1.0	2.56	4.52	1.8	202561	229973	1.1
FP Divider	1074	1336	1.2	7497	7760	1.0	3.10	4.26	1.4	88359	116597	1.3

TABLE II
BENCHMARK DESIGN RESOURCE UTILIZATION

	5-LUTs*	LUTRAMs	FFs	DSPs	BRAMs
Viterbi Decoder	11,653	0	2,535	0	4
AES Cipher	18,554	0	11,447	0	12
Microblaze	15,612	827	10,853	6	19
FIR Filter	449	5,471	10,108	130	0
FP Divider	5,343	201	5,934	0	0
xc6vlx75tff484	93,120	—	93,120	288	156

* 6-LUTs are counted as two 5-LUTs

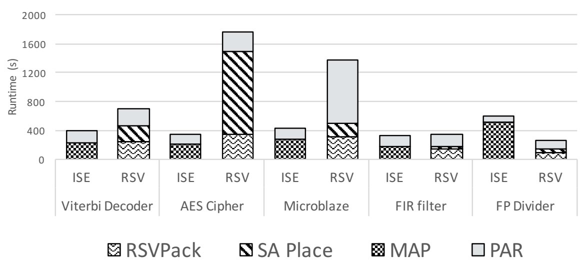


Fig. 6. Flow Run Times

Table I presents statistics from the packed circuits. The number of used slices and number of nets exposed to the general fabric (Exposed Nets) are direct functions of the packer. The total wire length, as measured as the total manhattan distance of all wires used in the design, and minimum achievable clock period on the other hand are products of the entire flow.

RSVPack generally leads to a slightly higher number of nets exposed from the slices and total number of used slices. These differences are not large and suggest RSVPack does a reasonable job of filling the clusters with shared logic. However, there is a significant increase in total wire length and minimum clock period. It is not clear how much of this increase is due to RSVPack and how much is due to the placer.

V. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented the RSVPack packing algorithm for Virtex 6 FPGAs. This packer differs from other approaches targeting Xilinx devices in the following ways:

- Accepts a Xilinx ISE generated netlist as input
- Supports unique Virtex 6 features such as the LUTRAM and SRL capabilities of the LUTs
- Performs an exhaustive exploration of all packing units for a given seed cell.
- Employs a BEL selection criteria that takes into account the local FPGA interconnect unique to the Virtex family.
- Uses a fast comparison-based routing feasibility validation scheme in place of a router.
- Performs additional checks to support the unique requirements of the slices.

This work lays the foundation for exploring novel approaches for packing algorithms targeting commercial Xilinx FPGAs. We plan to eventually include a router to create a complete backend academic CAD flow for Xilinx architectures. The entire flow will then be updated to support the latest Ultrascale architectures and be integrated with the Vivado CAD tools using a forthcoming RapidSmith 2 update.

REFERENCES

[1] C. Lavin, M. Padilla, J. Lamprecht, P. Lundrigan, B. Nelson, and B. Hutchings, "RapidSmith: Do-It-Yourself CAD Tools for Xilinx FPGAs," in *Field Programmable Logic and Applications (FPL), 2011 International Conference on*, Sept 2011, pp. 349–355.

[2] T. Haroldsen, B. Nelson, and B. Hutchings, "RapidSmith 2: A Framework for BEL-level CAD Exploration on Xilinx FPGAs," in *Proceedings of the 23rd ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '15. ACM, to be published.

[3] J. Luu, J. Goeders, M. Wainberg, A. Somerville, T. Yu, K. Nasartschuk, M. Nasr, S. Wang, T. Liu, N. Ahmed, K. B. Kent, J. Anderson, J. Rose, and V. Betz, "VTR 7.0: Next Generation Architecture and CAD System for FPGAs," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 7, no. 2, pp. 6:1–6:30, Jul. 2014. [Online]. Available: http://doi.acm.org/10.1145/2617593

[4] J. Luu, J. H. Anderson, and J. S. Rose, "Architecture Description and Packing for Logic Blocks with Hierarchy, Modes and Complex Interconnect," in *Proceedings of the 19th ACM/SIGDA International Symposium on Field Programmable Gate Arrays*, ser. FPGA '11. New York, NY, USA: ACM, 2011, pp. 227–236. [Online]. Available: http://doi.acm.org/10.1145/1950413.1950457

[5] J. Luu, J. Rose, and J. Anderson, "Towards Interconnect-adaptive Packing for FPGAs," in *Proceedings of the 2014 ACM/SIGDA International Symposium on Field-programmable Gate Arrays*, ser. FPGA '14. New York, NY, USA: ACM, 2014, pp. 21–30. [Online]. Available: http://doi.acm.org/10.1145/2554688.2554783

[6] E. Hung, F. Eslami, and S. Wilton, "Escaping the Academic Sandbox: Realizing VPR Circuits on Xilinx Devices," in *Field-Programmable Custom Computing Machines (FCCM), 2013 IEEE 21st Annual International Symposium on*, April 2013, pp. 45–52.

[7] E. Hung, "Mind the (synthesis) gap: Examining where academic FPGA tools lag behind industry," in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, Sept 2015, pp. 1–4.

[8] E. Bozorgzadeh, S. O. Memik, X. Yang, and M. Sarrafzadeh, "Routability-driven Packing: Metrics and Algorithms for Cluster-based FPGAs," *Journal of Circuits Systems and Computers*, vol. 13, pp. 77–100, 2004.

[9] A. Singh and M. Marek-Sadowska, "Efficient Circuit Clustering for Area and Power Reduction in FPGAs," in *Proceedings of the 2002 ACM/SIGDA Tenth International Symposium on Field-programmable Gate Arrays*, ser. FPGA '02. New York, NY, USA: ACM, 2002, pp. 59–66. [Online]. Available: http://doi.acm.org/10.1145/503048.503058

[10] L. McMurchie and C. Ebeling, "PathFinder: A Negotiation-Based Performance-Driven Router for FPGAs," in *Field-Programmable Gate Arrays, 1995. FPGA '95. Proceedings of the Third International ACM Symposium on*, 1995, pp. 111–117.