Un/DoPack: Re-Clustering of Large System-on-Chip Designs with Interconnect Variation for Low-Cost FPGAs

Marvin Tom, David Leong, Guy Lemieux
Dept of Electrical & Computer Engineering, University of British Columbia
{ marvint | davel | lemieux } @ ece.ubc.ca

ABSTRACT

FPGA device area is dominated by interconnect, so low-cost FPGA architectures often have reduced interconnect capacity. This limited routing capacity creates a *hard channel width constraint* that can make it difficult for CAD tools to successfully map a circuit into these devices. Instead of migrating a design to a high-cost, resource-rich architecture that is easier to route, we present a cheaper alternative: *a fully automated CAD flow* (Un/DoPack) that finds local regions of high interconnect demand and reduces it by spreading out the logic in that region. This is done by introducing whitespace in the form of empty logic elements (LEs) within the configurable logic blocks (CLBs) of the congested region. After spreading, the congested region occupies more routing channels and so obtains access to greater aggregate interconnect capacity. Although this has the side effect of using more CLBs, it has the advantage of *lowering peak interconnect demands* and making a previously-unroutable circuit routable. We also design a new set of synthetic benchmark circuits that model *interconnect variation* within a large design. Using these benchmarks, we show that circuits with high interconnect variation require FPGA devices to have large channel widths. However, since congestion of such circuits is localized, Un/DoPack is very good at reducing the peak demands of circuits with high interconnect variation. Our results suggest that even for an *average* Rent exponent of 0.62 (a modest value), a large *variation* of this exponent within a design will also require FPGAs to have large channel widths. Thus, it is crucial to *study interconnect variation* of benchmark circuits when designing low-cost FPGAs. Previous research studying interconnect properties focuses on average Rent exponent values of each design, but we believe new work should study variation as well. For circuits with high interconnect variation, we demonstrate that channel widths can be reduced by up to ~40% with only ~10% increase in area.

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

General Terms: Algorithms, Design, Experimentation

Keywords: Field-Programmable Gate Arrays (FPGA), Clustering, Packing, Channel Width Constraints

This research has been enabled by the use of WestGrid computing resources, which are funded in part by the Canada Foundation for Innovation, Alberta Innovation and Science, BC Advanced Education, and the participating research institutions. WestGrid equipment is provided by IBM, Hewlett Packard and SGI.

1. INTRODUCTION

As FPGAs increase in capacity and capability, it is common to offer separate low-cost and resource-rich families. For a similar number of logic elements (LEs), the low-cost families often have less embedded memory, embedded multipliers, and routing tracks. This is demonstrated by Table 1, where the low-cost Cyclone family offers significant savings. Unfortunately, some designs may fit within the Cyclone LE and memory capacity limits but not within the routing capacity limits. This can be solved by switching to the resource-rich family at ~4x the cost. Instead, it is preferable to stay in the low-cost family and use the same or next-larger device (at ~2x cost). To do this, the FPGA CAD must meet the device routing capacity by targeting a *hard channel width constraint*. Since interconnect use of a design varies spatially with placement, this can be done by spreading out regions of peak demand to use fewer routing tracks but more CLBs [1][2][3][4].

More CLBs in a local region occupy more routing channels, increasing the aggregate routing capacity available to that region. This reduces the interconnect demand in each individual channel and eliminates the peak, allowing the entire FPGA channel width to shrink and save area. This paper presents an algorithmic way of reducing the *minimum routable channel width* (MRCW) of a logic design by inserting whitespace in the form of empty LEs into congested areas. Whitespace is inserted by identifying a congested region of CLBs, fully unpacking the CLBs of that region into its constituent LEs, and then re-packing these LEs into new CLB clusters so they are "less full" than before. This process of inserting whitespace into each CLB is called *depopulating*.

We believe this to be the first automated FPGA CAD flow which iteratively strives to meet fixed channel width and fixed array size constraints using standard CAD algorithms.

Large System-on-Chip (SoC) designs are often created by combining several IP blocks, each of which is tightly connected internally. To represent this type of design, we created synthetic benchmark circuits and varied the local Rent exponent of each IP block. A large Rent exponent generally translates into high interconnect demand for an IP block. Using these benchmarks, we observe that SoC designs containing high variation in the Rent parameters of their constituent IP blocks require large MRCW values, but these same circuits are also the most amenable to our flow for reducing channel widths. Thus, MRCW of an SoC design can be reduced by taking advantage of its interconnect variation.

Table 1: Features and Costs of Two FPGA Families

Altera Device	LEs	Memory	Multi-pliers	Routing	Price (2005)
Cyclone 1C12	12,060	234 kb	0	80	$56
Stratix 1S10	10,570	899 kb	48	232	$190
Cyclone 1C20	20,060	288 kb	0	80	$100
Stratix 1S20	18,460	1,630 kb	80	232	$350

2. BACKGROUND AND RELATED WORK

The logic capacity of an FPGA can be measured by the number of configurable logic blocks (CLBs) which are fixed-sized clusters of LEs, consisting of LUTs and flip-flops. Logic capacity can also determined by the logical dimensions of the CLB array. Routing capacity is determined by the channel width of the device, *i.e.*, the number of wiring tracks in each channel. We define the **minimum routable channel width (MRCW)** of a circuit as the minimum channel width an FPGA architecture must have to route a given circuit. The **maximum MRCW** (**max MRCW**) is defined as the MRCW required without any depopulation.

Normally, FPGA tools fully pack CLBs with the maximum number of LEs they can hold to achieve 100% logic utilization. Previous work [1][2][3][5] has shown that a trade-off exists between interconnect demand and the logic-utilization of a circuit. DeHon [1] showed that the total area required by a circuit could be reduced by balancing logic utilization and routing elements. However, this work assumed a hierarchical interconnect structure (where interconnect capacities are easy to compute) which is not representative of modern commercial mesh-based FPGAs. Tessier [2] presented a uniform depopulation technique, but this quickly leads to area increases because regions with low interconnect demand are also depopulated. Tom [3] presented a manual method to non-uniformly depopulate congested regions, but that technique relies on the design hierarchy, which is not always available, to partition the design into depopulation regions. Although [3] targets a channel width constraint, it frequently produces solutions that exceed the constraint. Singh [5] attempts to balance interconnect demand using a Rent-based constraint during clustering. However, there is no way to control the amount of depopulation that occurs or to target hard channel width constraints.

Independence, an FPGA placement tool by Sharma [4], targets hard channel width and array size constraints. By using the router tool as an *inner loop* during placement, it runs 4 orders of magnitude slower than regular tools [4]. In comparison, our tool flow is much faster and can work with most existing cluster, place and route tools. Also, Independence inserts entire CLBs as whitespace and leaves already-clustered CLBs intact, while our flow actively re-clusters to insert individual LEs as whitespace.

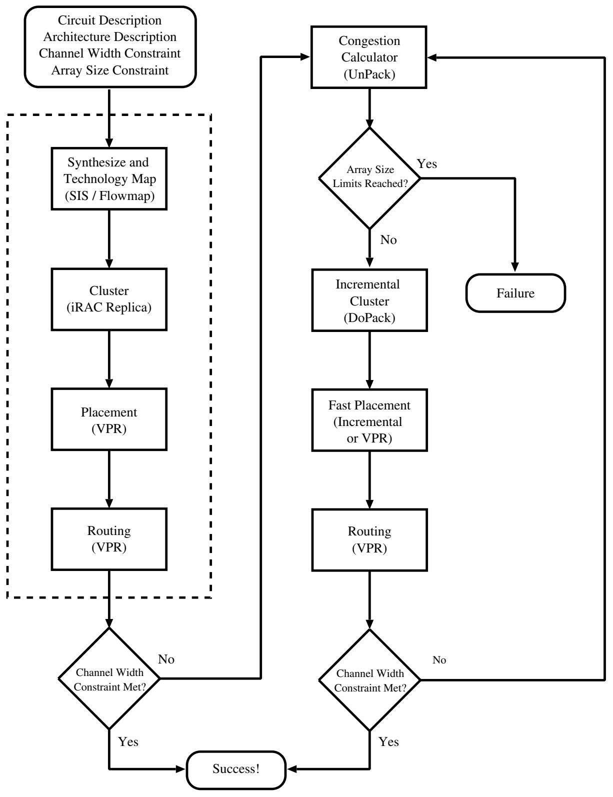
3. UN/DO PACK CAD FLOW

Un/DoPack reduces the MRCW by iteratively performing non-uniform cluster depopulation which effectively inserts whitespace (unused LEs) into CLBs located in high congestion areas. Figure1 describes the overall Un/DoPack flow. Details of each step are discussed below.

There are 4 inputs to Un/DoPack: the circuit description, the architecture description, the hard channel width constraint (interconnect capacity) and the array size constraint (logic capacity). For a given array size M, there are M^2 CLBs – we shall assume this is enough to fit the initial clustered solution.

The region in dashed box of Figure1 is the traditional academic FPGA CAD flow which uses SIS / FlowMap [6] and VPR [7]. If the traditional CAD flow fails to produce a routed solution with the given channel width constraint, the iterative portion of Un/DoPack is invoked to reduce the MRCW.

The iterative portion of Un/DoPack has 4 discrete steps. First, the UnPack step determines which portion of the circuit is congested and fully unpacks the CLBs in this region. Second, the DoPack step clusters all of the unpacked LEs with a smaller cluster size constraint. Third, a new placement is computed. Fourth, a final route is performed again to determine the new interconnect usage. UnPack, DoPack, and the Place and Route steps are further discussed below.



Circuit Description
Architecture Description
Channel Width Constraint
Array Size Constraint
Synthesize and Technology Map (SIS / Flowmap)
Cluster (iRAC Replica)
Placement (VPR)
Routing (VPR)
Channel Width Constraint Met?
No
Yes
Congestion Calculator (UnPack)
Array Size Limits Reached?
Yes
No
Failure
Incremental Cluster (DoPack)
Fast Placement (Incremental or VPR)
Routing (VPR)
Channel Width Constraint Met?
No
Yes
Success!

Figure1: CAD Flow

3.1 UnPack: Congestion Calculator

UnPack is the congestion calculator. It determines the size of the depopulation region and the amount of depopulation for the region. The depopulation amount is controlled by formulating a new cluster size constraint to be used in the clustering tool. The new cluster size **must be smaller** than the current average LEs per CLB of the region to ensure that some depopulation occurs through the production of more CLBs. Details about which CLBs are chosen as part of the depopulation region and how to compute the new cluster size constraint are discussed below.

Following a failed routing attempt, UnPack creates a congestion map based on the final routed solution. The congestion map is created computing a *CLB congestion label* with the maximum local channel width required in each of the 4 routing channels adjacent to the CLB. Note that some wires may have multiple nets assigned to it from the failed (illegal) routing solution. The local channel width required is calculated by counting the total number of *nets* routed through each channel next to this CLB.

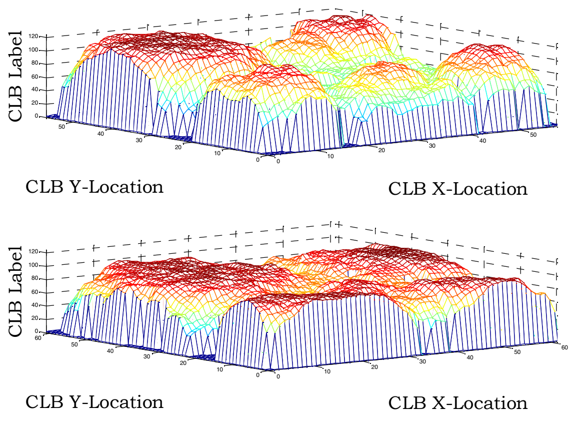


Figure 2: Congestion Map before and after Un/DoPack

Figure 2 shows a sample 3-D congestion map of a circuit before (top) and after (bottom) Un/DoPack meets a channel width constraint of 100. The x-y coordinates indicate the CLB locations in the FPGA array, and the z coordinate indicates the CLB congestion label (just described). This result was produced using the Clone/Stdev008 benchmark circuit (discussed blow). In the figure, we can see that there are some regions of high congestion and some regions of low congestion. The peak / avg / stddev of congestion labels were 120 / 79.4 / 26.9 tracks before Un/DoPack, and 100 / 79.2 / 19.6 afterwards. Notice also that the minimum array size increases from 55 to 60 due to the increase in CLBs.

We attempted two different methods to determine how large the depopulation region is and how much to depopulate the region by per iteration. These two methods are described below.

3.1.1 Single Region (SR) Depopulation

A single depopulation region center is identified by finding the largest label in the congestion map. In the case of a tie, the CLB that is closest to the center of the device is chosen as the depopulation center. All CLBs with a Euclidean distance less than radius R from the depopulation center are considered part of the depopulation region. The radius R is set to 1/4 the array size *M*. For example, in a 19x19 CLB grid, R = $\lfloor 1/4*19 \rfloor$ = 4. The new cluster size is determined such that the increase in the total number of CLBs will fill an entire new row and column in the array (*i.e.*, *M* will have to increase by 1, adding 2*M* + 1 new CLBs). This spreads the depopulation region to span more rows and columns, effectively increasing the aggregate interconnect capacity available to it. Equation 1 below determines the number of LEs to repack into each CLB in the depopulation region:

$$new\_cluster\_size = \left\lceil \frac{num\_LEs\_in\_region}{number\_CLBs\_in\_region + 2M + 1} \right\rceil \quad (1)$$

When the minimum routable channel width approaches the targeted constraint, an end game strategy is applied to ensure convergence of the algorithm. If the peak channel width is equal to the constraint but the solution is still not routable, it will lower the targeted constraint by 5. This is a tunable factor that is circuit dependent. It ensures some depopulation will occur to reach the target.

3.1.2 Multiple Region (MR) Depopulation

The single region depopulation scheme outlined above selects a large amount of CLBs to depopulate per iteration. To further refine the identification process for congested CLBs, a multi-region approach was also attempted. Instead of selecting only one congestion region with a radius of 1/4 the array size, multiple regions with a smaller radius are selected per iteration. Radius sizes of 1/10, 1/15 and 1/20 the array size were attempted. The goal of selecting multiple regions is two-fold. First, using a finer grain selection process aims to select irregularly-shaped regions and fewer CLBs than the single region approach. Second, a multi-region approach may reduce the number of iterations and speed Un/DoPack convergence by targeting all congested areas at once.

The multi-region selection process begins by finding the largest label in the congestion map and selecting all of the CLBs within the small radius from this centre. These CLBs are added to a congestion list for this region. The selection process then loops and finds the next-highest CLB label remaining in the congestion map (*i.e.*, not in any congestion list) until no more labels are higher than the targeted channel width. Overlaps are handled by marking CLBs as "selected" so they are not added twice to different congestion lists. Each of the regions will be reclustered separately – the new cluster size for each region is calculated individually using Equations 2 to 4 below. By targeting the highest CLB labels first, this process tries to apply just enough depopulation per region.

$$\alpha = 45 \cdot radius \quad (2)$$

$$number\_new\_clbs = \alpha \cdot \left(\frac{highest\_clb\_label\_of\_region}{channel\_width\_constraint} - 1 \right) \quad (3)$$

$$new\_cluster\_size = \left\lceil \frac{num\_LEs\_in\_region}{num\_CLBs\_in\_region + num\_new\_clbs} \right\rceil \quad (4)$$

Equation 3 shows that the number of CLBs to add to the region is proportional to how congested the area is multiplied by a scaling factor, α. This scaling factor is shown in Equation 2 as an empirically-determined constant of 45 times the radius of the congestion region (in units of CLBs). We found that adding new CLBs at a rate proportional to perimeter (*i.e.*, radius) of the region converges faster than proportional to area (*i.e.*, radius^2). The new cluster size calculated using Equation 4 is similar in nature to the single region method.

3.2 DoPack: Incremental Re-Cluster

The second step, DoPack, is an *incremental clustering* step. It reads the output from the UnPack stage: a list of LEs in each region and the new cluster size for each region, plus a list of already-packed CLBs. For single region depopulation, there is only a single LE list and single constraint. For multi-region, there are multiple region lists and constraints. DoPack iteratively clusters each designated region using the new cluster size constraint. This step can use *any* existing clustering approach (*e.g.* T-VPack [7], T-RPack [8], iRAC [5]) as the underlying constraint is *only* changing the maximum cluster size. Our implementation is based on the T-VPack code base and does not alter the CLBs that are not congested. However, since it sees the entire circuit, it can do critical-path analysis and be timing-driven when re-clustering. For our results, we use the iRAC algorithm implemented within the T-VPack code base. iRAC was chosen because it is the best congestion-driven clustering tool known, it is fast, and it results in good delay performance (even though it is not timing-driven).

By using a smaller cluster size in the congested regions, this guarantees the production of more CLBs. This is *crucial*: by using more CLBs, the congested region can span more routing channels to obtain more total routing tracks.

In future work, the DoPack step may be improved by treating the new cluster size constraint as an *average target* for each region and not a hard constraint. This may help improve delay performance since LEs on the critical path may remain "fully packed" as long as there are non-critical LEs that can be "less packed" to meet the average target.

3.3 Placement and Routing

The purpose of the place and route steps is to accurately identify regions of routing congestion. Ideally, this could be done with a *fast congestion estimator* that can precisely locate the regions of peak routing demand. Unfortunately, we are not aware of such a tool for FPGAs. Meanwhile, we decided to use actual place and route directly; this is slow, but accurate. Due to iteration in the flow, it is important to speed up both the placement and routing steps as much as possible. These options are discussed below.

3.3.1 Faster Placement

To speed up placement, we modified VPR to perform incremental placement [9]. The incremental placer attempts to provide placement stability by preserving the previous locations of CLBs outside of the depopulation region. This decreases run time and provides consistent and predictable CLB placements as the CAD flow iterates to reduce channel width. The incremental placer works in three stages. The first stage is an "expansion" phase which squeezes the numerous "depopulated" CLBs into the "too small" space left behind. This produces illegal solutions. The second stage is a "compaction" phase used to legalize the solution. The third stage is an optional low-temperature anneal to clean up the solution. Further details of the incremental placer are covered in [9]. The incremental algorithm quickly computes a high quality solution, *e.g.*, incremental placement with 60% changed CLBs takes roughly 1/3 of the time required for a full placement.

It should be noted that VPR placement and our incremental placer is wirelength-driven, not congestion-driven. The use of a congestion-driven placement engine such as [4] or [10] should be explored in the future.

3.3.2 Faster Routing

To speed up routing, we attempted to obtain congestion results from the first iterations of the VPR routing algorithm. At this early stage, there is significant illegal wire sharing. We were unable to successfully use this data. We have not yet attempted to develop an incremental routing algorithm. Hence, for this work we let the VPR router run to completion. This is the *primary reason* why our approach is slow.

4. Methodology

This section presents the experimental framework, methodology, and baseline parameters used when running Un/DoPack. It also presents a new suite of benchmark circuits used to determine the effects of interconnect variation. This new benchmark suite contains a series of circuits of approximately 52,000 LEs, each with an increasing amount of interconnect variation.

4.1 Baseline Parameters and Framework

We use the VPR experimental framework. Our baseline Un/DoPack flow is based on:

- Single Region congestion calculator (described in section 3.1)
- A replica [3] of the iRAC algorithm [5] used as the underlying clustering algorithm for DoPack (described in section 3.2)
- Incremental placer described in [9].
- FPGA architecture with LUT size $k = 6$, cluster size $N = 16$, inputs per cluster $I = 51$, and wires of length $L = 4$.

Because of large run times and limited computing resources, we set a maximum run time of 48 hours. If this limit was exceeded, we conclude that no solution exists. All calculations use a dedicated Pentium 4, 3GHz processor with 1.5GB of RAM. Before Un/DoPack was run on a benchmark circuit, VPR was first used to precompute the MRCW of a circuit *without any* depopulation by invoking the binary search option of the VPR router. This defines the ***max MRCW*** for each circuit. Then, we set various channel width constraints up to 50% below the max MRCW for each circuit and run the Un/DoPack flow. Since these channel width constraints are below the max MRCW, some amount of depopulation must occur to meet the given channel width constraint. Critical path delay numbers are calculated using the highly congested channel width where the circuit is barely routable – this differs from traditional work that usually reports timing after relaxing the routing constraints by increasing channel width by 20% (or more) above minimum.

4.2 Interconnect Variation Benchmarks

Large benchmark circuits are rare for FPGA research, so we have developed a set of synthetic circuits which mimic those used in previous work [3]. One conclusion from [3] was that non-uniform depopulation is important for large System-on-Chip designs that have IP blocks of varying interconnect demand. Such benchmarks are not widely available, so [3] synthesized benchmarks by randomly stitching together existing, smaller benchmarks (MCNC circuits) and called them IP blocks. However, the stitching was somewhat unrealistic as a flip-flop was placed at every IP block output to prevent combinational loops. Instead, we used a synthetic benchmark generator, GNL [11] to create large benchmarks. GNL builds a benchmark hierarchically and permits the Rent exponent, subcircuit size, and other parameters to be specified in each division. This allows us to build benchmarks with a ***controllable amount*** of interconnect variation. We will show that Un/DoPack is effective at reducing MRCW for these GNL circuits regardless of their interconnect variation.

To create our synthetic circuits, our input to GNL describes two levels of hierarchy. The root level defines the overall structure of the circuit. This level includes the total number of logic cells in the circuit, as well as a required input-output count. The number of primary inputs and outputs were defined as 240 and 120 respectively. The root level is also defined to have twenty subcircuits, each of which mimics one of the 20 largest MCNC circuits [12]. Each subcircuit is intended to represent an IP block with a specific interconnect demand. The number of inputs and outputs for each subcircuit was not defined, thus allowing GNL to randomly stitch each region together to form the overall circuit. The number of logic cells and the Rent exponent of each IP block were chosen to match the same values of the 20 MCNC circuits. The Rent values were taken from [5]; the average Rent value used is 0.62 and the standard deviation of these values is 0.08. Using this process and parameters, we have essentially produced a clone of the MetaCircuit from [3] and named it *Stdev008*.

To create a family of circuits, we devised a simple linear interpolation scheme of the Rent exponent to keep the *same average* Rent value, but to vary the *standard deviation*. This produced 4 circuits with smaller variation and 2 with larger variation. Figure 3 shows a graphical representation of our linear interpolation scheme. For clarity, only 10 of the 20 MCNC circuits are shown as "subcircuits" within the larger design.

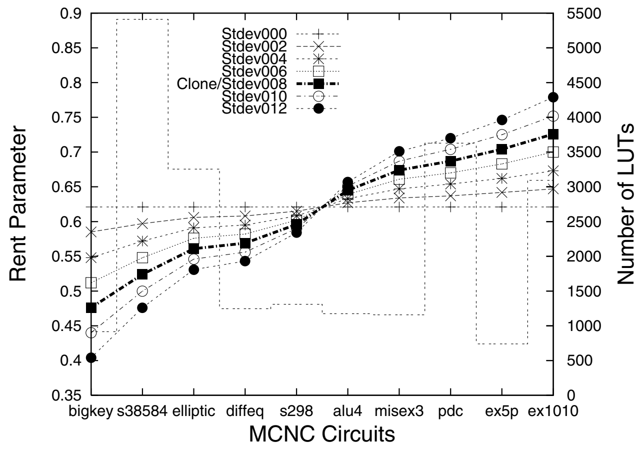


Figure 3: Rent Linear Interpolation Graph

Each line in Figure 3 represents a new benchmark circuit with a specific set of Rent parameter values. Circuit Stdev000 uses 20 identical Rent parameters of 0.62, producing a flat line. Circuit Clone/Stdev008 uses the Rent values from [5]; in the figure, the subcircuits were ordered according to this value. Three other circuits with standard deviations 0.02, 0.04, 0.06 were created by simple interpolation between the flat Stdev000 line and bold Clone circuit line. Circuits Stdev010 and Stdev012 were obtained by extrapolating the Rent parameter 2 steps farther. The "bar graph" line in Figure 3 shows the size of each of the subcircuit in terms of the number of LEs; note that the size does not depend on the Rent parameter.

The resulting circuits have the *same mean* Rent value of 0.62, but the standard deviations differ at 0.00, 0.02, 0.04, 0.06, 0.08, 0.10 and 0.12. All circuits contain 51,900 to 52,200 6-input LEs. The benchmark circuits will show that a large amount of depopulation is necessary to reduce the MRCW of circuits with a low standard deviation in interconnect demand. This is because the Rent exponent is uniform, and routing resources demands should be fairly consistent across the entire circuit. In contrast, with a high standard deviation, interconnect demands should be non-uniform, thus allowing the depopulation scheme to reduce the routing demands of high Rent regions.

It is interesting to compare these synthetic circuits to properties of industrial circuits given in [13]. The *average* Rent exponent of individual industrial circuits is shown to range from 0.4 to 0.8 (0.5 to 0.7 being more "typical"). The average Rent exponent across these different circuits is 0.60 with a standard deviation of 0.063; this is similar in nature to our benchmark properties. Also, [13] demonstrates that Rent exponent is not correlated with logic utilization of an FPGA device, but is strongly correlated with half-perimeter bounding box (wirelength estimate) calculated during placement.

5. Results

The experimental results will be presented in three stages. Single-region depopulation results are presented first, followed by multi-region results and then comparisons to previous work.

5.1 Single Region Experimental Results

Un/DoPack was used to target channel width constraints up to approximately 50% of the max MRCW for each circuit. Figure 4 shows the normalized area increase for each circuit as the channel width constraint is tightened. Overall, there are two competing factors: area decreases due to lower channel widths, but increases due to more CLBs. Note that Figure 4 only shows data points where Un/DoPack was successful in meeting the channel width constraint (within CPU time limit and array size constraints).

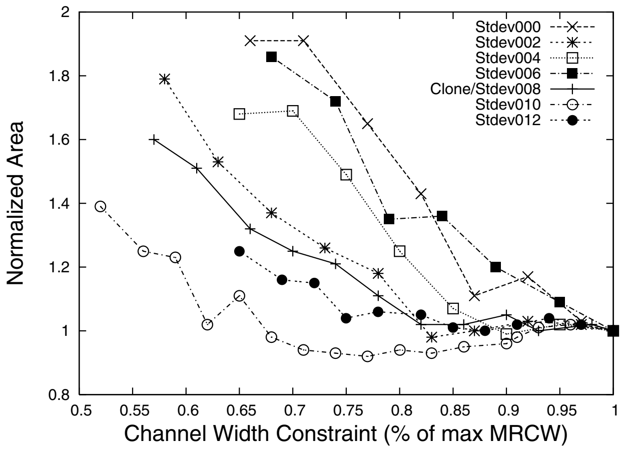


Figure 4: Area vs. Channel Width Constraint

Figure 4 shows that Un/DoPack (single region) was successful in reducing channel width by 30% for all benchmark circuits and up to 45% in some cases. For circuits with high interconnect variation (Stdev010, Stdev012), significant channel width savings is possible with little or no area inflation. Circuits with low interconnect variation (Stdev004, Stdev002, Stdev000) show quick area increases already at modest reductions in channel width. For example, circuit Stdev010 shows a 40% decrease in channel width with only 10% increase in area. This occurs because there is a local high congestion region and only a small amount of depopulation is needed to reduce the congestion in this region. In contrast, circuit Stdev000 shows large area increases for small decreases in channel width. This is because a large amount of depopulation is needed to reduce the channel width of the entire circuit due to little interconnect variation. These observations were verified manually by tracking the size of depopulation regions and the changes in cluster size as the flow iterated over several different circuits.

Figure 5 shows that circuits with high interconnect variation require significantly higher channel widths to route (without constraints). This suggests that it is crucial for FPGA architects to know the amount of interconnect *variation* within their benchmark circuits. If the variation is too high, it is possible that the routing networks will be designed with excess capacity, resulting in undue cost to the consumer. Fortunately, these very circuits are the most amenable to channel width reduction using our Un/DoPack flow. In this figure, the straight line (triangle markers) indicates the area of an FPGA device with 100% LE utilization (no depopulation) at different channel widths. An

FPGA architect must choose a channel width along this line, *e.g.* at 110 tracks per channel where circuits Stdev008, Stdev010 and Stdev012 are unroutable. These 3 circuits can be made routable at 110 tracks using the Un/DoPack flow. Circuit datapoints in the figure give the total FPGA area required (total CLBs at the given channel width, including whitespace overhead) after Un/DoPack.

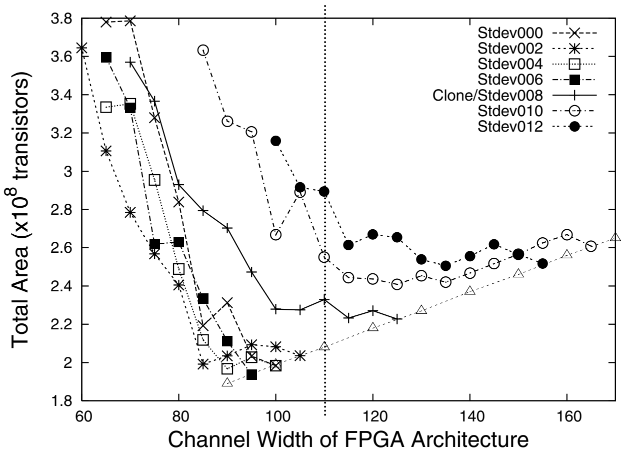


Figure 5: Area vs. Channel Width Constraint

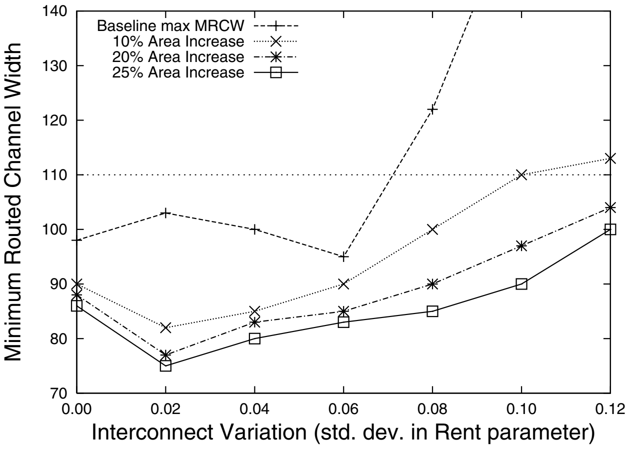


Figure 6: MRCW vs. Interconnect Variation

Figure 6 shows the channel widths that were attainable with no depopulation (the max MRCW at 100% LE utilization) and with channel width constraints that produce net area increases of 10%, 20% and 25%. The *x*-axis represents the standard deviation in the Rent parameter of the circuit. FPGA architects typically choose channel widths for their devices to fit as many circuits as possible. All circuits here all have Rent exponents of 0.62 and seem to be reasonable targets for the benchmark suite. The channel width required to route circuits with high interconnect variation is very large (over 140). However, a more realistic choice for the channel width of the device may be 110 tracks. This would result in a 21% decrease in channel width and translates directly into an area savings. Most circuits (with low variation) map into 110 tracks and achieve a net area (cost) savings compared to 140. The few circuits (with high variation) that could not be mapped can still be depopulated using Un/DoPack to meet the given channel width constraint. This may be done for free if the array size of the device has not yet been met, or it may come at a cost penalty if the next-largest device must be used. However, for most users, a cost savings is produced. This illustrates how understanding the amount of interconnect variation in individual circuits is important for low-cost FPGA design.

Figure 7 shows the normalized critical path delay vs. channel width constraint expressed as a percentage of the max MRCW. There is, on average, a 10% penalty in critical path for a 20% decrease in channel width, and up to 23% increase in critical path for a 45% decrease in channel width. This is a modest penalty that may be improved through the improvements described in Section 3.2. If timing performance is absolutely crucial, depopulation and low-cost FPGAs may not be the right choice.

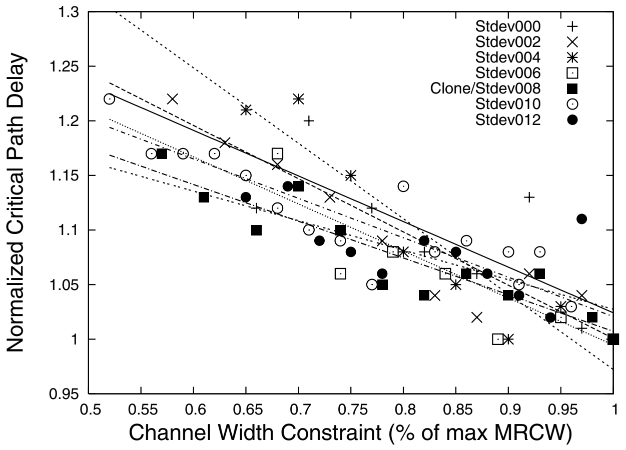


Figure 7: Critical Path Delay vs. Channel Width Constraint

5.2 Multi Region Experimental Results

The single region depopulation experiments presented in the previous section show that significant channel width reductions are possible for circuits with high interconnect variation. In this section, the results for multiple region depopulation will be presented. Different congestion radii of 1/10, 1/15 and 1/20 of the array size were attempted. It was found that with multi-region depopulation, the area and critical path results were similar in all cases to the ones obtained in the single region experiments. Precise tuning of the algorithm parameters takes a long time and can be benchmark-dependent, so only the results for multi-region depopulation using a congestion radius of 1/10 the array size are presented below. Also to improve run-time, fewer channel width constraints were specified and only a subset of the benchmark circuits was used.

Figures 8 and 9 show the normalized area and critical path results. The results are similar to the single region results, but were slightly better in most cases (approx. 3% improvement). Also, the results suggest that multi-region results are less noisy compared to the single region results. This may be a result of fewer data points, or it may be caused by the use of smaller depopulation regions producing a more graceful trade-off. The finer grain areas are able to tailor cluster sizes more appropriately to individual congestion peak sizes and regions. In comparison, single region targets only the highest peak with a single large region that may encompass multiple smaller peaks as well as valley regions.

Figure 10 presents the run time of the Un/DoPack algorithm for single region and multi-region depopulation. Single region depopulation tends to be faster when the channel width constraint is above 0.75 of the max MRCW. Below 0.75, multi-region depopulation is significantly faster. The threshold of 75% is somewhat not sharp. The reason for run-time difference is due to the interconnect variation. During the first few iterations of the flow, the most congested peaks are targeted. For single region depopulation, the algorithm depopulates a large area and immediately flattens the most congested **and** surrounding areas. This can be seen from the **Clone/Stdev008** data line where it requires the same amount of runtime between 0.85 and 1.0 of max MRCW. On the other hand, multi-region is fine grain and only targets the specific congested areas. After the initial congested areas have been depopulated, the shifting locations of interconnect demand still cause the multi-region algorithm to iterate. Once the major congested peaks are all flattened, the entire circuit needs to be depopulated to reduce channel width below 0.75 of max MRCW. At this point, multi-region depopulation is much more effective than single-region because it can target multiple regions across the chip. At the extreme of 50% channel width reductions, multi-region run-times are 2 to 4 times faster than single region. It may be possible to improve run-times further through algorithm tuning.

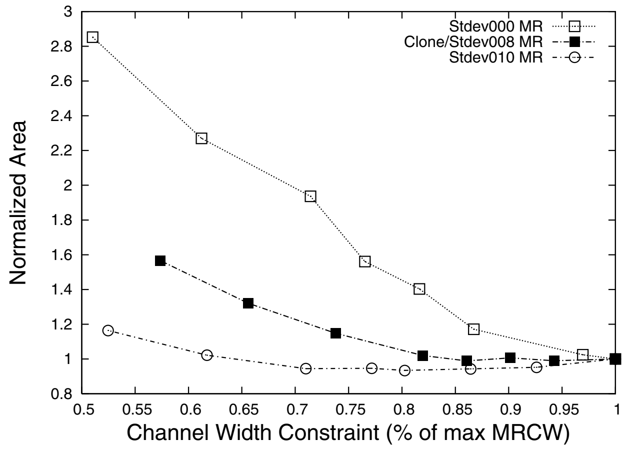


Figure 8: Area Increase vs. Channel Width Constraints

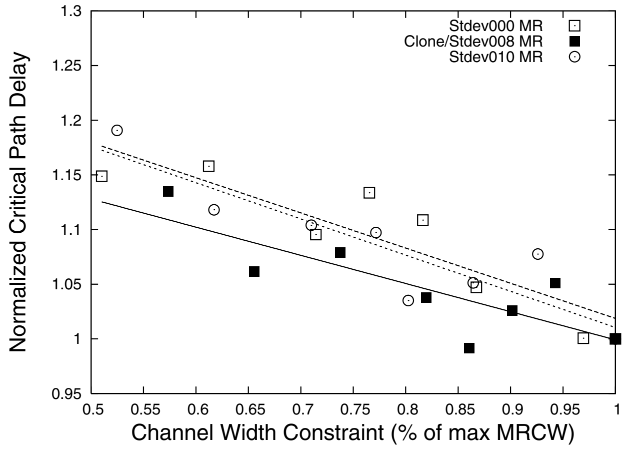


Figure 9: Critical Path Delay vs. Channel Width Constraints

5.3 Comparisons to Previous Work

The technique described in [3] creates a large benchmark circuit by stitching IP blocks together post-clustering. Essentially, it treats each IP block as an independent depopulation region. We ran Un/DoPack for 3 of the benchmark circuits described in [3]: Clique, Pipeline, and Independent for comparison.

Figure 11 shows the final routed channel width vs. the channel width constraint for Un/DoPack and for the technique described in [3]. Although [3] attempts to target a channel width constraint, it does not iterate so many solutions are generated that exceed the constraint. In contrast, Un/DoPack consistently meets the given channel width constraint. Although not shown, on rare occasions Un/DoPack sometimes beats the constraint by 1 track.

Figure 12 shows the minimum array size needed for the channel width reduced solutions generated using both techniques for one circuit. Compared to [3], Un/DoPack is sometimes able to use considerably smaller array sizes. In other results (not shown), Un/DoPack typically obtains similar or smaller array sizes to [3].

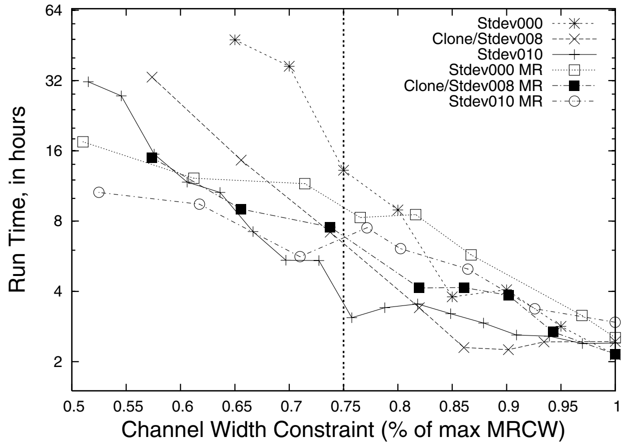


Figure 10: Algorithm Run Time

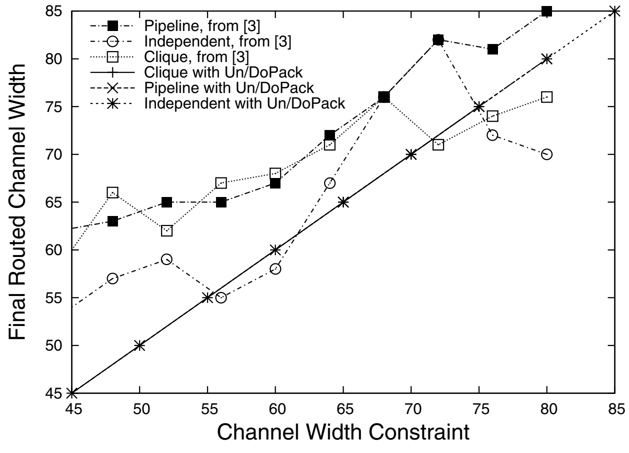


Figure 11: Final Routed Channel Width vs. Constraint

6. CONCLUSIONS

We have presented a new CAD tool Un/DoPack that is able to reduce the minimum routable channel width (MRCW) of any circuit at the expense of logic utilization. Key findings include a 40% decrease in MRCW with a 10% increase in area for circuits with high interconnect variation. For circuits with low interconnect variation, channel width reduction is still possible at the expense of area. For channel width reductions of less than 25%, single region depopulation with a large radius was found to be the fastest in run time. For significant channel width reductions of greater than 25%, a multi-region depopulation approach with a small radius is 2-4 times faster. It was found that the interconnect variation in a circuit and not just average interconnect usage helps dictate the amount of routing resources necessary in FPGAs. This is crucial in the design of low-cost FPGAs.

Future work will involve investigating other techniques to improve run time by using a congestion estimator instead of place and route or through the use of incremental or fast routing approaches. However, this is most important only for the "first compilation" of a large design. In production use, after a design has been compiled with Un/DoPack at least once, the most congested regions will have already been identified and will likely remain the same from one compilation run to the next. Hence, several iterations can be saved from the Un/DoPack flow during the edit-compile-debug process.

Our incremental clustering approach could be improved to be "more aware" of the critical path as suggested in Section 3.2. Also, the approach should be augmented with a congestion-aware placement tool such as [10]. We are presently integrating our framework with Quartus, which is reportedly congestion-aware as well.

To be truly industrial-strength, the approach needs to be extended to include the effects of hard macro blocks (such as memories, multipliers, and fast carry chains). Also, the effects of very long interconnect wires found in FPGAs need to be studied – it is anticipated that this actually makes Un/DoPack easier by removing some "high frequency" variation in interconnect demand and leaving just the easier-to-handle "low frequency" variation. These issues are also the subject of on-going investigation.

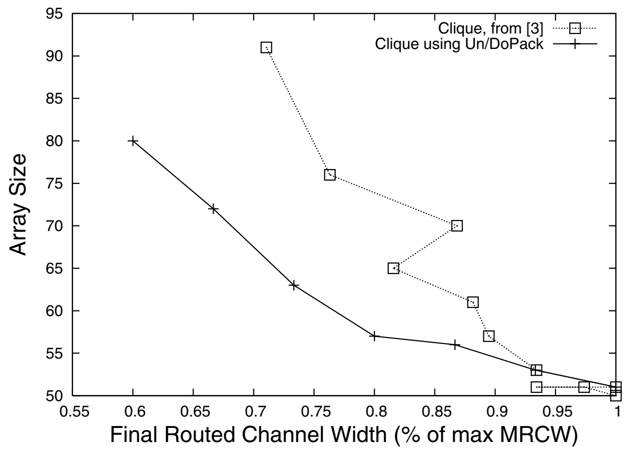


Figure 12: Array Size vs. Channel Width Constraint

7. REFERENCES

[1] A. DeHon, "Balancing Interconnect and Computation in a Reconfigurable Computing Array (or, why you don't really want 100% LUT utilization)," *Field Programmable Gate Arrays*, 1999.

[2] R. Tessier and H. Giza, "Balancing Logic Utilization and Area Efficiency in FPGAs," *Field Programmable Logic (FPL)*, 2000.

[3] M. Tom and G. Lemieux, "Logic Block Clustering of Large Designs for Channel-Width Constrained FPGAs," *DAC*, 2005.

[4] A. Sharma, C. Ebeling and S. Hauck, "Architecture-Adaptive Routability-Driven Placement for FPGAs," *FPL*, pp. 427-432, 2005.

[5] A. Singh and M. Marek-Sadowska, "Efficient Circuit Clustering for Area and Power Reduction in FPGAs," *Field-Programmable Gate Arrays*, Feb, 2002.

[6] J. Cong and Y. Ding, "FlowMap: An Optimal Technology Mapping Algorithm for Delay Optimization in Lookup-Table Based FPGA Design," *IEEE TCAD*, Vol.13, no.1, pp.1-12, Jan 1994.

[7] V. Betz, A. Marquardt, and J. Rose, *Architecture and CAD for Deep-Submicron FPGAs*, Kluwer, Norwell, MA, 1999

[8] E. Bozorgzadeh et al, "Routability-Driven Packing : Metrics and Algorithms for Cluster-based FPGAs," *Journal of Circuits, Systems, and Computers*, 13(1), pp. 77-100, Feb. 2004.

[9] D. Leong and G. Lemieux, "iPlace: An Incremental Placement Algorithm for Field-Programmable Gate Arrays," in preparation, 2006.

[10] U. Brenner and A. Rohe, "An Effective Congestion Driven Placement Framework," *Int'l Symp. on Physical Design*. pp.6-11, 2002.

[11] P. Verplaetse, D. Stroobandt and J. Van Campenhout, "Synthetic Benchmark Circuits for Timing-driven Physical Design Applications," *Int'l Conf. on VLSI,* pp. 31-37, 2002.

[12] LGSynth93 Benchmark Suite, Microelectronics Centre of North Carolina. Tech. Report, 1993.

[13] J. Pistorius and M. Hutton, "Placement Rent Exponent Calculation Methods, Temporal Behaviour and FPGA Architecture Evaluation," *Int'l Workshop on System-Level Interconnect Prediction,* pp.31-38, 2003.