PASAP: Power Aware Structured ASIC Placement

Ashutosh Chakraborty
ECE Department
University of Texas at Austin
Austin, Texas, USA
ashutosh@cerc.utexas.edu

David Z. Pan
ECE Department
University of Texas at Austin
Austin, Texas, USA
dpan@ece.utexas.edu

ABSTRACT

Structured ASICs provide an exciting middle ground between FPGA and ASIC design methodologies. Compared to ASIC, structured ASIC based designs require lower non recurring engineering (NRE) costs and turn-around-time but suffer from higher power consumption and lower performance. Power reduction for structured ASICs uses extensive clock and supply power-down of unused circuitry and use of low power devices. However, due to the limited granularity of power-down, physical design (specially placement) should be performed to maximize the components that can be powered down. In this paper, we present the first placement algorithm to specifically target this problem. Our tool, `PASAP` (Power Aware Structured ASIC Placement), minimizes the clock and leakage power by maximizing the fraction of the structured ASIC that can be powered down or disconnected from clock tree. On a set of large benchmark designs, `PASAP` reduces clock and leakage power by 32% and 17% respectively compared to prior structured ASIC placement tool `RegPlace` [1] incurring 17% penalty in wirelength and 30% longer runtime.

Categories and Subject Descriptors

B.7.2 [**Hardware, Integrated Circuit**]: Design Aids

General Terms

Algorithms, Design

Keywords

low power, placement, regular fabrics, structured ASICs

1. INTRODUCTION

Skyrocketing costs and increasing variability associated with an ASIC design flow and unacceptable power and delay penalty associated with FPGA design flow have forced semiconductor companies to look for alternatives. One viable alternative that has emerged over time is the use of *structured* ASICs. Structured ASICs provide an exciting middle-ground between high performance of ASIC designs and short time-to-market FPGA designs. They exploit the fact that not all mask-layers provide equal value for the customers and these layers can be pre-fabricated amortizing their cost over multiple designs [2]. Structured ASIC flow is much simpler than that for traditional ASICs because majority of deep submicron issues such as signal integrity, power grid optimization, low skew clock tree distribution are already taken care of by the structured ASIC vendors.

Higher power consumption compared to the ASIC approach can prove to be the Achilles heel of structured ASIC methodology. To reduce the power penalty, there have been many innovations in their architecture including use of via programming (instead of SRAMs which drain leakage power), low leakage triple oxide devices, and power-down of unused components. From the EDA perspective, the power savings can be increased if the design tools can exploit these power optimization methods. One such step is to enhance placement algorithms to maximize the unused components so that they can be powered down. Distribution of cells as decided by placement determines which areas of the structured ASICs must remain powered on. Due to limited granularity of power down available, many non functioning devices may be forced to remain powered up due to neighboring functioning devices, leading to wasted leakage power. In addition, placement also determines the extent of the clock network that must switch at each clock cycle. Typically, cells belonging to only one clock group can be placed close to each other, therefore wrong distribution of cells can significantly burn much more power by requiring larger clock network to remain active. In this paper, we tackle power aware placement on structured ASIC platforms for the first time and propose methods to reduce the leakage and clock power.

The rest of this paper is organized as follows: We provide background of structured ASIC architecture and placement in Section 2. Section 3 details the leakage and clock power model used to evaluate the placement solution. Power aware structured ASIC placer (`PASAP`) is presented in Section 4. The experimental setup is described in Section 5. Power reduction results are presented in Section 6. Section 7 concludes our paper.

2. BACKGROUND

2.1 Structured ASIC Architecture

Like FPGAs, structured ASICs consist of a regular 2-D repetition of a basic building block called *tile*. Depending on the size of the design to be implemented on structured ASIC, the number of repetition of the tiles in the horizontal and vertical direction can be modified. The physical structure achieved by repetition of the tile a certain number of times, will be referred to as *platform*. Since tile is the basic building block of structured ASIC, we describe its architecture next.

ISLPED'10, August 18–20, 2010, Austin, Texas, USA.

Each tile is a rectangular region in which the devices are pre-fabricated. There can be many different units fabricated in a tile such as logic cells, flip flops, multiplexers, adders etc. In this work, we will primarily target the architecture of a tile as released by leading structured ASIC company `eASIC` [3] as part of their worldwide placement contest. There are four different types of cells fabricated in each tile: logic cells, flip flops units, register files and RAMs. We refer to these cells as `LOGC`, `DFF`, `REG`, and `RAM` respectively in the rest of the paper. The relative size of these cells are 1x1, 1x1, 8x32, and 16x64 thus their area is in the ratio 1:1:256:1024 respectively. Except `LOGC`, all other cell types are sequential in nature. The locations where these cells are pre-fabricated in the tile are as shown in Figure 1. `LOGC` and `DFF` cells are fabricated in columns of height 64 each. There are 36 and 24 columns (not all shown in Figure 1) of `LOGC` and `DFF` cells respectively in each tile. At the center of the tile, one `RAM` is fabricated and half of the columns of `LOGC` and `DFF` each appear on both side of the RAM cell. On the right end of the tile, 4 `REG` are fabricated in 2x2 array formation. In summary, each tile has 2304 `LOGC`, 1536 `DFF`, 4 `REG` and 1 `RAM` cells. Clock routing constraints dictate that not more than a few clocks can be distributed in a tile (though the platform can have many more unique clocks). Therefore, the sequential cells inside each tile should not require more than a given number of unique clocks.

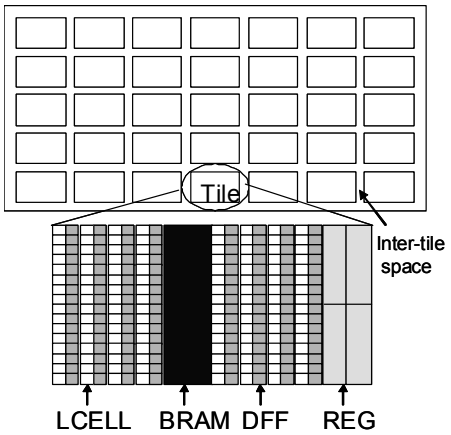

Figure 1: Structured ASIC platform.

2.2 Structured ASIC Placement

Placement on structured ASIC takes as input a) an RTL expressed in terms of `LOGC`, `DFF`, `REG`, and `RAM` cells, b) the architecture of the structured ASIC platform, c) Geometry information about the tile, and d) Clock constraints at the platform and at the tile level; and maps each cell to a pre-fabricated location on the platform such that site compatibility as well as clock constraints are satisfied. The objective of the placement can be to minimize wirelength or delay of the circuit while meeting these constraints.

Previous Work: Recent placement contest sponsored by eASIC [3] fostered research in the field of placement for structured ASICs. Here we describe the grand prize winning entry `RegPlace` [1]. Due to its availability (open source) and high quality, we use `RegPlace` as our placement framework and baseline to compare against. `RegPlace` first transforms the structured ASIC placement problem into simpler row-based placement problem by resizing the larger cells (`REG` and `RAM` to be unit sized and removing the physical space reserved for these cells from the platform. Also, the clock constraints are ignored at this stage. Existing high quality row-based placer (such as CAPO [4], mPL6 [5] etc) is used to generate an initial placement with excellent wirelength. The placement results are then expanded to re-insert physical space removed for `REG` and `RAM` cells while concurrently performing cut minimization due to re-inserted space for these cells. An integer linear programming (ILP) based clock assignment and network flow based clock legalization is performed to satisfy the clock constraints. Finally wirelength reduction is performed to undo any damage due to legalization.

3. LEAKAGE/CLOCK POWER MODEL

Leakage Power Model: The leakage power of a structured ASIC platform is roughly proportional to the number of devices in it. Even if no cell (in the RTL) is mapped on to a device on the platform, the device will still consume leakage power unless disconnected from the power supply. Existing structured ASICs already allow power down (i.e. disconnect from power grid) of unused devices on the platform using single-via power programming. We assume that power-down can be performed at two levels of granularity: a) An unused tile can be completely powered down reducing its leakage (and dynamic) power to zero, and b) Within a tile, an unused column can be powered down if no functioning cells are placed in it, reducing its power to zero. The leakage power savings achieved by power down of a complete tile is equivalent to that achieved by individual power down of each column in the tile individually. If the leakage power of one column is P_{col}, then the leakage power model of the structured ASIC platform is simply the sum of leakage power of all the columns that are not power down.

$$P_{\text{leak}} = P_{\text{col}} \times \sum_{t \in \text{tiles}} \left(\sum_{c \in \text{cols(t)}} E_c \right) \quad (1)$$

where cols(t) denotes the columns in tile t and E_c is a binary variable whose value is 1 if the column c is powered on and 0 otherwise. From Eqn 1, it is clear that we should try to minimize the number of columns powered up to reduce leakage power.

Clock Power Model: Unlike leakage power, clock power is largely proportional to the clock switched capacitance. Due to stringent slew and skew constraints on the clock signal, clock is distributed across the platform using hierarchy of high drive power hungry buffers. To reduce clock power, the aim should be to reduce clock network's switched capacitance. To understand the clock power model, we must first understand clock distribution architecture. We assumed a simplistic clock distribution network as depicted in Figure 2 described using 2x2 tiles to reduce clutter. Figure 2-a depicts the platform level clock distribution whereas Figure 2-b depicts the tile level clock distribution. The architecture we assume is based on the recent works on clock distribution on FPGAs [6] [7].

Here, we describe the clock distribution architecture of a single clock which is duplicated to distribute multiple clocks. In Figure 2-a, level 1 buffer **B1** inserts the clock signal at the center of the chip that drives a horizontal spine. Each level 2 buffer **B2** distributes the clock signal to one vertical spine. Similarly, a level 3 buffers **B3** drives the clock signal into each tile. Inside a tile, as shown in Figure 2-b, Level 4 buffers **B4** drive the `DFF` columns (only 1 shown) as well as `REG` and `RAM` cells. Owing to the larger number of sequential elements in `RAM` and `REG` cells, multiple **B4** buffers are required to drive them (only

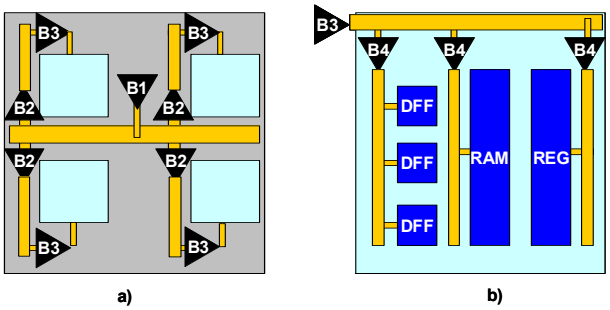

Figure 2: Clock distribution: a) Platform b) Tile level

1 shown). We assume that each **B4** buffer can drive only as many sequential elements as are in a DFF column. To distribute multiple clocks, the **B1** and **B2** clock buffers can be replicated for each clock signal. Due to routing constraints, typically only a subset of platform level clocks can be routed into each tile. Each **B3** and **B4** clock buffers is replicated as many times as the size of subset of clocks that can be routed in each tile. We assume that power-down of clock signal can be performed at three levels of granularity: a) An unused vertical spine's clock can be disconnected if all of the tiles driven by this spine are unused. b) An unused tile can be disconnected if all cells in it are unused, and c) within the tile, the clock driving each DFF column, REG, RAM cell can be disconnected.

For platform with WxH tiles (instead of 2x2 in Figure 2), let C_G clocks at platform level and C_T clocks at tile level be supported. Further, let the number of equivalent (after scaling for larger sizes of REG and RAM) number of DFF columns be S each containing N sequential elements. Under these assumptions, Table 1, shows the constituents of the clock network. Columns 2 contains the number of buffer needed whereas column 3 shows the capacitive load on each of these buffers. The last column shows the total switched capacitance seen by the corresponding level of clock buffers when the input capacitance of an entity X is denoted by L_X.

Table 1: Clock Tree Distribution Characteristics

Buffer Type	Number Instance	Each Load	Total Load
B1	C_G	2 W L_{B2}	2 W C_G L_{B2}
B2	C_G 2 W	0.5 H L_{B3}	2 W H C_G L_{B3}
B3	C_T W H	S L_{B4}	W H S C_T L_{B4}
B4	C_T W H S	N L_{DFF}	W H S N C_T L_{DFF}

Using the clock network architecture and Table 1, the total clock power can be written as

$$P_{\text{clk}} = \sum_{i=0}^{C_G}\left(L_{B1} + \sum_{j=0}^{2W}\left(E_{ij}L_{B2} + \sum_{k=0}^{0.5H}(E_{ij}E_{jk}L_{B3})\right)\right) + \sum_{t\in\text{tiles}}\left(\sum_{m=0}^{C_T}\left(\sum_{n=0}^{S}(E_{mn}L_{B4} + E_{mn}L_{DFF})\right)\right) \quad (2)$$

where all E_{xy} are boolean variables. E_{ij} is 1 if clock i drives spine j, E_{jk} is 1 if spine j drives tile k on that spine, E_{mn} is 1 if clock m in the tile drives the column n. The first term in Equation 2 is the total switched capacitance of C_G global clocks and the second term is the total switched capacitance within a tile, summed over all the tiles. Selective power down of unused components (spine, tile, column) modulate the values of these binary variable thus changing the clock power.

4. LOW POWER PLACEMENT

We now introduce the problem being tackled in this work.

Given a structured ASIC platform and a design, perform cell placement to minimize the clock and leakage power consumption by maximizing the components that can be powered down. Power reduction should not degrade the wirelength by more than a given budget.

4.1 Overall Flow

Figure 3, shows our complete flow. The *rectangular* blocks represent our new contributions whereas the oval blocks represent the steps in RegPlace [1], an existing open source placer for structured ASICs. After describing the overall flow in this section, each rectangular block will be explained in more details in next few subsections.

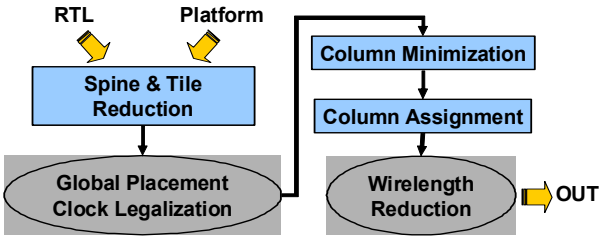

Figure 3: Our placement flow. Blue rectangles are our new contributions. Grey ovals are from [1].

Given the input netlist and the platform, we minimize the vertical spines and tiles that must remain powered up by generating new constraints for global placer. This step works under a given wirelength increase budget. Global placement is then executed on the design followed by clock legalization. In the next step, we minimize the number of columns that must stay powered on by avoiding scenarios where the number of cells in a tile is slightly more than integral multiple of number of cells that can be accommodated in a column, thereby requiring another column. Finally, we cluster and assign the cells into as few columns as possible while minimizing wirelength change.

4.2 Spine and Tile Reduction

To mark a vertical spine as unused, we must ensure that *all* the tiles driven by the spine are marked as unused. To mark a tile as unused, we must make sure that no cell is placed in it. If the design to be placed is much smaller than the physical platform, naturally there would be many such tiles. However, our aim is to maximize such unused tiles by identifying those which would have very low number of cells in them and evicting these cells without much wirelength penalty.

Algorithm 1 depicts our overall spines and tile reduction strategy. We performed rough placement to generate a coarse density map ensuring that the cut-line during partitioning coincide with the tile boundaries (Line 1). The wirelength increase budget is set to k% of the original wirelength (Line 2). For each horizontal trunk in the platform, we go over all the vertical spines connected to it. These spines are sorted in non-decreasing order of the number of cells in all the tiles driven by the corresponding vertical spine (Line 4-5). Then, the sorted set of spines is iterated over while trying to evict the cells in it using procedure EvictSpine. The scheme for reducing the number of tiles is also similar (Line 8-10) wherein the procedure EvictTile is executed for each tile in sorted order. We now describe Algorithm 2 and Algorithm 3 which implement procedure EvictSpine and EvictTile respectively.

Algorithm 1 Vertical Spine and Tile Minimization

```
 1: Perform Rough Placement
 2: BUDGET = k % of wirelength of design {Minimize spine
    utilization in next block}
 3: for all Horizontal Spine hs do
 4:    CONN = All vertical spines connected to hs
 5:    Sort CONN by number of cells in it
 6:    for all Vertical Spine vs connected to CONN do
 7:       EvictSpine(vs, BUDGET)
    {Minimize tile utilization in next block}
 8: T = All tiles in platform
 9: Sort TILES by number of cells in it
10: for all Tile t in TILES do
11:    EvictTile(vs, BUDGET)
```

Algorithm 2 EvictSpine(Spine S, BUDGET)

```
1: SL(SR) = Spine on left(right) of S
2: REST = Tiles in SL ∪ SR
3: for all Tile t in S do
4:    NGHBR = All tiles in REST <= 1 tile away from t
5:    Build flow between t and NGHBR
6: Add Constraint for WL Increase < BUDGET
7: if Flow Possible then
8:    Move cells from tiles in S to SL or SR
```

Algorithm 2 which tries to evict a given spine works as follows. For a target current spine being evicted, the list of all tiles in the spines to the left and right of it is obtained (Lines 1-2). Next we write a network flow formulation where all the cells from each tile belonging to the current spine are moved out of the spine. For each tile in the spine, the possible location of interest are all the tile adjacent to it except itself (Line 3-5). We limit the cells to move to only adjacent tiles to reduce wirelength penalty. The objective of the network flow is to minimize the estimated wirelength increase due to the movement. An additional constraint is added to bound the maximum acceptable wirelength increase (Line 6). If the network flow is possible, the spine S can be evicted and its cells are moved out.

Algorithm 3 EvictTile (tile T, BUDGET)

```
 1: Get xAvg = average X coordinate of cells in T
 2: Get yAvg = average Y coordinate of cells in T
 3: Compute Up/Dn/Lf/RtConn for T
 4: NGHBR = All neighbors of tiles T
 5: for all tiles n in NGHBR do
 6:    if n is an empty tile, continue;
 7:    moveX = n's center X coordinate - xAvg
 8:    moveY = n's center Y coordinate - yAvg
 9:    WLINCR = moveX × (LfConn - RtConn)
10:    WLINCR += moveY × (DnConn - UpConn)
11:    Add flow from T to n with cost WLINCR
12: Add Constraint for WL Increase < BUDGET
13: if Flow Possible then
14:    Evict T and move cells
```

Algorithm 2 can only evict one whole spine at a time. However, there can be many scenarios where a few (but not all) tiles of a spine have small number of cells in it and can be evicted. For this, Algorithm 3 works on each tile in the design. For each tile, we find the average x and y coordinate (XAvg and YAvg) of the cells in it (Line 1-2). We also compute four numbers (RtConn, UpConn, LfConn, DnConn) which represent the number of nets incident on any cell in the tile whose bounding box is strictly to the right, above, on left, or below this tile (Line 3). The intuition is that if all the cells of this tile are moved to another tile, the increase in wirelength by this move can be approximated by combination of these numbers. Next, we iterate over all eight physically contiguous tiles around the target tile (Line 5) which are the possible destinations while computing the estimated wirelength increase (Line 7-10). A network flow problem is built and if there is a legal solution to it, the tile T can be evicted (Line 13-14).

After the termination of Algorithm 1, we obtain the largest set of complete spines and tiles which can be evicted under given wirelength increase budget. Based on this information, we transform the placement problem by adding fixed blockages covering the unused tiles. This transformed placement problem becomes the input to RegPlace global placer.

4.3 Column Minimization

After the global placement and clock legalization has been performed, combinational (i.e. LOGC) and sequential cells belonging to multiple clocks could be distributed in each tile. To exploit this flexibility, we apply Algorithm 4 to a rectangular peephole M tiles wide and N tiles high. Once the number of powered up columns in the peephole is reduced, the peephole is moved to the next adjacent position and Algorithm 4 is rerun. If the whole platform consists tiles that is W wide and H high, Algorithm 4 is executed W-M×H-N times. Now we describe the

Algorithm 4 Column Count Reduction

```
 1: T = All tiles in current peephole
 2: N = Maximum cells in a column
 3: C = All clocks in T ∪ "NIL"
    {"NIL" Clock connects to combinational cell i.e. LOGCs}
    {Let n_ci = cells of clock c in tile i}
 4: for all Clock c ∈ C do
 5:    MOVES = φ
 6:    USED = Σ_{i∈T} ceil(n_ci/N)
 7:    REQD = ceil(Σ_{i∈T} n_ci)/N
 8:    if USED == REQD then
 9:       continue; {Column count already optimized}
10:    for all Tile t ∈ T do
11:       t.Extra = n_ct - N × floor(n_ct/N)
12:    Sort T in non increasing order of Extra(t)
13:    for all Tile t ∈ sorted T do
14:       OTHER(t) = T - {t}. Sort w.r.t. distance from t.
15:       for all Tile o ∈ sorted OTHER do
16:          trans = min(t.Extra, N-o.Extra)
17:          MOVES = MOVES ∪ {trans from t to o}
18:          if t.Extra == 0 then
19:             break
20:    Implement all MOVES stored
```

steps involved in one invocation: We add a dummy clock called "NIL" which connects to all combinational cells (i.e. LOGC cells) (Line 3). The optimization is performed for one clock type at a time (Line 4). The maximum number of cells that can be accommodated in a column is given as N (Line 2). We prune the cases for which the number of columns occupied is already optimal (Line 6-9). For each tile in the peephole, we compute the minimum count of cells (Extra) that, when removed, leave an integral multiple of N cells (Line 10-11). All the tiles in the peephole are sorted according to non-decreasing Extra count

(Line 12). For each tile, every tile except itself is a prospective tile in which the `Extra` cells can be moved (Line 14). To reduce the impact on wirelength increase, we sort the prospective tiles according to distance from the current tile (Line 15) and try pushing the `Extra` cells of current tile into them as long as this move does not create new columns in the prospective tile (Line 17-18). Once all the moves to reduce the column count for a clock are recorded, we actually implement them by moving the cells among the tiles. When moving a cell from tile A to tile B, its new location in tile B is chosen as the point closest to the original location in tile A.

4.4 Column Assignment

After the termination of Algorithm 4, the decision regarding clustering of these cells into columns and how the columns for the different clocks are relatively placed, still needs to be made. For this, we use Algorithm 5 which runs on one tile at a time and packs the cells into individual columns. Algorithm 5 first

Algorithm 5 WL Aware Column Assignment

```
1: COLS = Columns in the tile
2: N = Capacity of each column
3: BINS = φ
4: C = All clocks in tile ∪ "NIL"
   {"NIL" Clock connects to combinational cell i.e. LOGCs}
5: for all Clock c ∈ C do
6:    CLSTR = φ
7:    L = cells of clock c from left to right
8:    for all Cell i ∈ L do
9:       CLSTR = CLSTR ∪ i
10:      if size of CLSTR == N then
11:         BINS = BINS ∪ CLSTR
12:         CLSTR = φ
13: Assign BINS to COLS [assignment problem].
    Cost of assignment = incident wirelength.
14: Swap cells between columns of same clock to reduce WL
```

performs greedy clustering of cells of same clock type into bins of size N (=capacity of each column) starting the sweep from the left edge of the tile (Line 9-12). All the cells belonging to one bin form one cluster and placed in one column. If the number of bins used for packing is b and the number of columns available in the tile is n, mapping the bins to n columns can be cast as an *Assignment Problem* and solved efficiently using Munkres algorithm (Line 13). Finally, to repair wirelength degradation due to sub-optimal clustering of cells into columns, cells are swapped between columns of same clock type if it reduces wirelength.

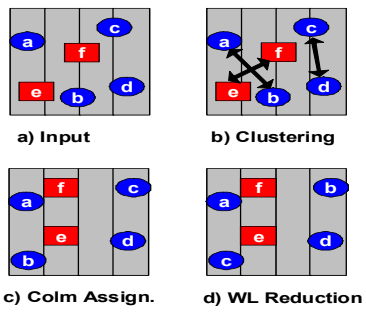

Figure 4: Column assignment: a) Input. b) Cells clustered based on original location. c) Columns assigned to each cluster. d) After WL reduction by swapping

Figure 4 shows an example of column packing in action. For this example, each column can have only 2 cells. Cells driven by the same clock type have the same shape. The cells are divided in three bins as shown in Figure 4-b. These bins are assigned to individual columns as shown in Figure 4-c. Finally, cell swapping may lead to swapping of cells b and c to reduce wirelength.

5. EXPERIMENTAL SETUP

We implemented our low power placement flow `PASAP` in the C++ language inside the open source structured ASIC placement tool `RegPlace` [1]. All experiments were run on a 16 core, 1.6 GHz, Linux workstation. We used GLPK [8] as the integer linear programming (ILP) and network flow solver and a custom implementation of Munkres [9] algorithm for assignment problem. The wirelength increase budget in Algorithm 1 was set at 15%.

Platform: Two different platforms were generated by replicating the *tile* described in Section 2.2. The details of these platforms are in Table 2. Columns 2 and 3 show the number of times the tile is replicated in horizontal and vertical direction respectively. Rest of the columns indicate the maximum number of `LOGC`, `DFF`, `REG`, `RAM` that can be accommodated in the platform.

Table 2: Platform Characteristics

Plat Name	Rep X	Rep Y	Max LOGC	Max DFF	Max REG	Max RAM	Max CLK
A	20	22	1M	675k	1760	440	32
B	22	24	1.2M	811k	2112	528	32

Designs: The benchmarks used were original released by `eASIC` for the placement contest. Table 3 shows the breakup of the design in terms of constituent cells. The total number of cells ranges between 125K to over 1 million. Column 6 shows the number of global clock signals used.

Table 3: Benchmark Characteristics

Name	LOGC	DFF	REG	RAM	CLK
easic1	832,824	87,052	110	172	18
easic2	812,200	45,478	175	686	7
easic3	961,063	52,780	192	0	3
easic4	102,038	23,330	0	44	7
easic5	913,853	84,505	145	262	26

6. RESULTS

Table 4 describes our experimental results comparing the placement generated by `RegPlace` to `PASAP` placement. The benchmark name in the first column also shows the platform on which the benchmark was placed. For example `easic4B` identifies that it is the benchmark easic4 placed on platform B. The metrics being compared for the two placement are : a) unused spines, b) unused tiles, c) unused columns, d) wirelength and e) total CPU runtime. For each of these metrics, three sub-columns are shown in Table 4. These sub-columns depict the value obtained using `RegPlace` (`RP` in table), value obtained using PASAP (`PA` in table) and the difference between these two solutions respectively.

The efficacy of `PASAP` to increase the number of unused spines, tiles and columns is evident from Table 4. Due to the different benchmark sizes and utilization ratio, the numbers vary widely among the different circuits. Therefore, in our discussion, we will primarily focus on the improvement/penalty summed over all the benchmarks. First of all, we note that over all the benchmarks, nearly 58% extra clock spines can be turned off using

Table 4: Increase in number of spine, tile and column that can be powered down for various benchmarks.

Bench	# Unused Spines			# Unused Tiles			# Unused Columns			Wirelength (x10^6)			CPU Runtime (s)		
Name	RP	PA	Δ%	RP	PA	Δ%	RP	PA	Δ%	RP	PA	Δ%	RP	PA	Δ%
easic1A	0	0	0	0	10	inf	12	1940	16066	13.2	15.5	17.3	8747	9968	13.9
easic1B	5	8	60.0	48	89	85.4	3112	7829	151	13.7	17.0	23.9	9984	11076	10.9
easic2A	0	3	inf	8	33	312.5	1086	3748	245	24.8	28.8	15.7	5974	8495	42.2
easic2B	8	12	50.0	88	119	35.2	5728	9163	60	26.1	29.6	13.3	8503	10348	21.7
easic3B	0	0	0	11	52	372.7	814	5285	549	19.7	22.4	13.9	5115	7098	38.7
easic4A	35	53	51.4	325	369	13.5	23185	24056	8.4	2.1	2.8	34.3	543	825	51.7
easic4B	38	56	47.3	417	455	9.1	25473	27014	2.8	2.2	3.0	38.5	562	741	31.9
easic5B	0	4	inf	19	54	184.2	2265	358	36.3	15.9	18.3	15.4	6409	10701	66.9
Total:	86	136	58.1	916	1179	28.7	59675	84837	42.2	118.0	137.8	16.7	45838	59253	29.3

our method. Turning off spine directly corresponds to lower clock capacitance. Benchmarks easic1A and easic3B has such high utilization ratio that both `RegPlace` and `PASAP` cannot find even one unused spine. Similarly, the number of unused tiles can be increased by around 29% thus giving significant enhancement in the power saving ability. As for the number of unused columns, their number improve by approximately 42% using `PASAP` as compared to `RegPlace`. The very low improvement in the number of column used for benchmark easic4A and easic4B is due to its very small size which implies even by using `RegPlace`, most of the columns are unused. The above improvement in increasing the count of unused spine, tiles and column comes at a cost as seen in the wirelength and runtime penalty. The wirelength of the designs increase by approximately 17%. Due to the runtime associated with a rough placement generation and aggressive spine/tile/column reduction, `PASAP` takes approximately 30% more runtime that `RegPlace`. Though runtime degradation is not insignificant, we believe it is a fair trade-off for the kind of power savings possible with `PASAP`.

To compare the power reduction, the change in placement metrics from Table 4 must be converted to clock and leakage power. Using the relation derived in Eqn 1, the leakage power saving achieved by performing placement using `PASAP` as compared to `RegPlace` is plotted in Figure 5. Each column in Figure 5 corresponds to one benchmark with the exception of last column which shows the average savings. `PASAP` reduces the leakage power in the range of 7% to 40% with an average value of 17%.

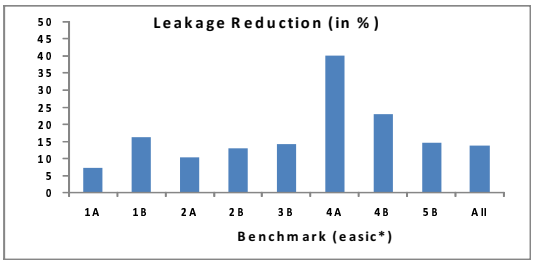

Figure 5: Leakage power savings for placement generated by `PASAP` compared to `RegPlace`.

Next we compare the clock power reduction due to `PASAP`. By aggressive reduction in the number of spines, tiles and columns in which the clock signal needs to be distributed, `PASAP` can significantly minimize the clock switched capacitance. Based on the clock power expression derived in Eqn 2, we plotted the clock power reduction achieved by `PASAP` for different benchmarks in Figure 6. `PASAP` reduces the clock power of different benchmark circuits in the range of 20% to 60% with an average value of 32%. The clock power was computed assuming the frequencies of all the clock signal in the structured ASIC is the same.

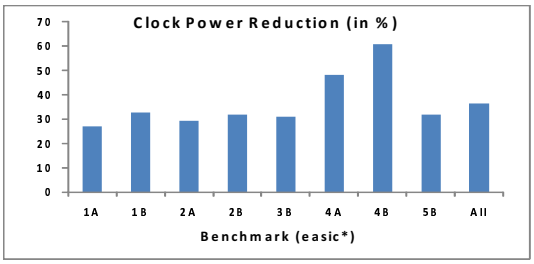

Figure 6: Clock power savings for placement generated by `PASAP` compared to `RegPlace`.

7. CONCLUSIONS

In this paper, we proposed power aware structured ASIC placement (`PASAP`) flow. For a given structured ASIC architecture, `PASAP` minimizes the leakage power and clock power dissipation by maximizing the devices that can be powered down and reducing the clock network's switched capacitance respectively. We proposed algorithms to minimize the number of clock spines, tiles (the building block of structured ASIC platform) and columns that need to be powered on, without significantly impacting the wirelength of the design. Experimental results on large testcases show that `PASAP` can reduce the leakage and clock power consumption by as much as 40% and 60% respectively with 17% penalty in wirelength and 30% longer placer runtime.

8. REFERENCES

[1] A. Chakraborty *et al.*, "Regplace: A high quality open-source placement framework for structured asics," in *Design Automation Conference*, pp. 442–447, ACM, 2009.

[2] H. Schmit *et al.*, "Placement Challenges for Structured ASICs," in *International Symposium on Physical design*, pp. 84–86, ACM, 2008.

[3] "eASIC Corporate Website." `http://www.easic.com/`.

[4] J. A. Roy *et al.*, "Capo: Robust and Scalable Open-source min-cut Floorplacer," in *International Symposium on Physical design*, pp. 224–226, ACM, 2005.

[5] T. F. Chan *et al.*, "mpl6: Enhanced Multilevel Mixed-size Placement," in *International Symposium on Physical design*, pp. 212–214, ACM, 2006.

[6] Q. Wang *et al.*, "Clock power reduction for virtex-5 fpgas," in *Proceeding of the International Symposium on Field Programmable Gate Arrays*, (New York, NY, USA), pp. 13–22, ACM, 2009.

[7] L. Wang *et al.*, "Fpga dynamic power minimization through placement and routing constraints," *EURASIP J. Embedded Syst.*, vol. 2006, no. 1, pp. 7–7, 2006.

[8] "GNU Linear Programming Kit." `http://www.gnu.org/software/glpk/`.

[9] H. W. Kuhn, "The Hungarian Method for the Assignment Problem," *Naval Research Logistics Quarterly*, vol. 2, pp. 83–97, 1955.