Process Migration of Hard IPs

Fang Fang, Jianwen Zhu

Technical Report TR-09-01
September 2001

Electrical and Computer Engineering
10 King's College Road
Electrical and Computer Engineering
University of Toronto, Ontario M5S 3G4, Canada

{ffang, jzhu}@eecg.toronto.edu

Abstract

While essential for high-performance circuit design, the custom nature of datapath components confines their use in only a few microprocessor companies. The reusability of datapath intellectual property (IP) libraries is largely limited by their dependence on process technology. Layout migration tools today, which are based on layout compaction developed decades ago, cannot cope with the challenges involved. In this paper, we present a comprehensive datapath IP development framework that can perform process migration by accommodating advanced circuit considerations, layout architecture and transistor sizing, in addition to design rule satisfaction. We demonstrate the effectiveness of the framework by migrating the Berkeley low power library, originally developed for 1.2um MOSIS process, into TSMC 0.25um and 0.18um technology.

Contents

1 Introduction

The performance gap between a custom design methodology, typically practiced by the microprocessor industry, and standard cell based design methodology, typically practiced by the ASIC industry, can be as much as one order of magnitude on the same process [2]. There are many factors that contribute to the performance gap between ASICs and microprocessors. One of the primary reasons is that the state-of-the-art ASIC design methodology treats every design as sea-of-gates, or random logic, while ignoring the regularity inherent in high-performance applications, which typically process data flow in a highly regular fashion.

The structured VLSI design methodology, advocated by Mead and Conway in the early 1980s [9], suggested the separation of control-path, either random-logic based or PLA/ROM-based, from the datapath, which are organized in identical or similar *bit slices* to perform a word-level computation. Each datapath component, for example, a 32-bit adder, is implemented by an array of *leaf cells*, each of which is contained in the corresponding bit slice. The leaf cells are carefully crafted so that area and routing efficiency is achieved: power/ground and control signals are routed implicitly by abutment, and data signals are routed in the perpendicular direction. This efficiency in area is further translated into efficiency in performance, since wire delays are minimal.

This rather "conventional" wisdom was always followed by microprocessor designs, yet largely ignored by ASIC design methodology based on logic synthesis. The major obstacle that prevents the synthesis of high-performance datapath is the high cost associated with the development of a datapath library as compared to standard cell libraries. First, datapath library has larger *functional variety*: it includes components performing not only logic functions, but also arithmetic functions, data-steering functions and storage functions. Second, datapath library has larger *lay-*

out complexity for each of leaf cell: it may contain many channel connected components placed in two dimension, rather than in one dimension for typical standard cell. Third, there are more *architectural constraints* to be satisfied: not only power/ground nets need to be aligned, many other control ports of different cells need to be aligned. Fourth, *performance characterization* is much harder: datapath components are parameterized over bitwidth and other parameters and it is difficult to drive an analytical performance model for them.

As the manufacturing process updates every 18 month, as dictated by Moore's law, even the cost of standard cell development cannot be contained in a fabless company, hence the birth of many intellectual property (IP) companies offering standard cell hard IPs. The manual development of datapath IP libraries, as argued earlier, are unlikely to survive process upgrade, therefore automatic process migration techniques have to be used.

Unfortunately, layout migration tools available today cannot cope with all the challenges involved. First, all techniques reported in the literature are designed to migrate a specific circuit that uses a library of cells, rather than the library itself. Without considering the *overall library architecture* such as power/ground net width, routing track number and port matching, the cell layouts migrated under this circuit-driven strategy work only for the specific circuit, and there is no guarantee they work under all occasions. Second, all migration tools are based on layout compaction, a technology developed decades ago, when layout area is the primary concern. Layout compaction often compresses space between polygons recklessly as long as design rules are not violated. In modern design using aggressive circuit style in deep submicron processes, space is often among the first class citizens of *advanced layout considerations*, for example, to combat signal integrity. Other specialized *advanced circuit considerations*, such as new transistor sizes, device matching, are rarely considered in an integrated fashion.

In this paper, we present an integer linear programming (ILP) based framework customized for datapath IP migration, where a datapath IP library for a new process is generated given a library of another process and a library architecture specification. Several innovations are introduced to help solve the difficult problems discussed earlier. First, we introduce a new optimization objective, called *geometric closeness*, to reward geometric resemblance of migrated layout to the original layout. Under this metric, space is explicitly represented. Preservation of space and preservation of non-space polygons are given equal priority. This ensures that the circuit and layout level considerations of the original designer are not corrupted. Second, we employ a *dual-pass strategy*, that is, running the migration engine twice, to address the library-driven requirement of our tool. This is in contrast to the top-down constraint propagation strategy employed by traditional hierarchical compactors, which are limited only to area minimization. Third, since the library architecture specified by users are likely to lead to infeasible solutions, we propose a new concept, called *soft constraints*, in order to obtain a best-effort solution. A concrete feedback is provided on where architecture requirements fail, which thereby helps the user interactively define the proper library architecture.

The organization of this paper is as follows. A tutorial on the general architecture of datapath layout is then given in Section 2. The ILP-based migration framework is introduced in Section 3. Section 4 elaborates on how practical issues, such as port matching, routing grid snapping, power line sizing, transistor sizing are handled in our framework. Experimental results are given in Section 5. The related work is reviewed in Section 6.

2 Datapath Layout Architecture

Before we discuss the datapath leaf cell layout structure, let's first look at how a datapath function unit, such as adder, register and multiplexer, is composed. In order to make use of of regularity of datapath layouts, an N-bit function block can be easily constructed by abutting predesigned leafcells in a one-dimensional or two-dimensional array topology.

Figure **??**(a) gives a function unit with N bit leafcell in Y direction. The control cell which is shared by N bit leafcell is abutted to the first bit or the last bit cell.

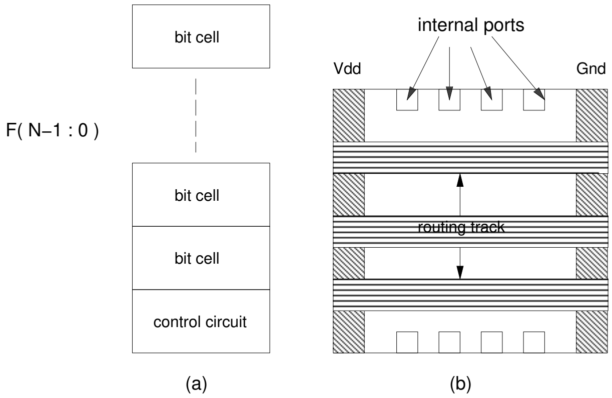


Figure 1. (a) Bit slice; (b) Leaf cell

.

The bit cell and control cell must be carefully designed beforehand so that they can be perfectly matched. FIGURE **??**(b) shows a typical leafcell layout. From this figure, we can find some features that make datapath leaf cell unique from other digital circuit layout.

- The horizontal interfaces have power lines and internal ports that allow abutment as is called "pitch matching".
- There are horizontal routing tracks which are placed above the leafcell with high level metal to increase "porosity" of the circuit. The width and spacing of routing track are decided by user, as we call it "routing track grid requirement".
- The widths of the Vdd line and Gnd line are decided by user as we call it "power line requirement".

3 ILP-based Migration Engine

The migration engine of our tool assumes that the leaf cell layout is Manhattan. The migration engine can therefore generate the migrated layout by determining new positions of horizontal and vertical edges of all geometries. In addition, it employs the traditional 1-D compaction strategy by first migrating along the X direction, or determing positions of vertical edges, and then the Y direction. Without loss of generality, in the discussion that follows, we assume migration in the X direction only.

A *constraint-based migration* of a leaf cell can be formulated as an integer linear programming (ILP) problem:

$$\begin{array}{ll} minimize & o^T x \\ subject\ to & Ax \geq b \\ & x \geq 0 \end{array}$$

Here x is a vector of variables to be determined, and in the simplest case would just be coordinates for all vertical edges. Vectors o represent the coefficients of x in the objective function of optimization. A is the constraints, each row of which represents coefficients of x in an inequality. Typically, an equality will be in the form $x_i - x_j \geq b_k$, where the constant b_k represents the minimum distance between two edges.

The migrated layout is largely influenced by the choice of the objective function. The traditional minimum area objective function is defined by

$$x_r - x_l \tag{1}$$

where x_r is the X coordinate of the right most vertical edge in the layout, and X_l is the X coordinate of left most vertical edge, as illustrated in Figure 2 (a).

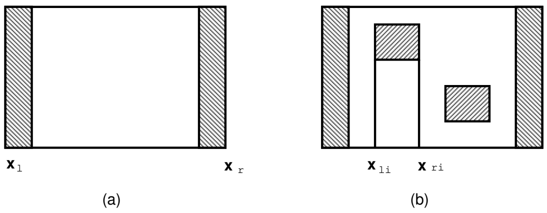


Figure 2. (a)Minimum area layout objective function. (b)Geometric closeness objective function.

To preserve advanced design considerations in the original layout, as argued earlier, we define a new objective function, called the *geometric closeness* as follows:

$$\sum |(x_{ri} - x_{li}) - (x_{ri}^{old} - x_{li}^{old})| \tag{2}$$

Here, x_{ri} and x_{li} are the X coordinates of the right and left edges of each rectangle in the layout. The constants x_{ri}^{old} and x_{li}^{old} are the X coordinates of the right and left edges of the corresponding rectangle in the original layout, as shown in Figure 2 (b). Note that space is explicitly represented as rectangles and are counted in the geometric closeness calculation. This is most conveniently achieved by using the corner-stitched layout representation [7].

To linearize, or to remove the absolute value computation in (2), we use a method similar to [5] by introducing two variables R_i and L_i for each rectangle, such that (3) are introduced as constraints,

$$\begin{aligned} R_i &\geq x_{ri} - x_{li} \\ R_i &\geq x_{ri}^{old} - x_{li}^{old} \\ L_i &\leq x_{ri} - x_{li} \\ L_i &\leq x_{ri}^{old} - x_{li}^{old} \end{aligned} \tag{3}$$

and the objective function is replaced by (4):

$$\sum (R_i - L_i) \tag{4}$$

The constraint generation is another key component of the migration process. It discovers the relative position requirements between layout edges coming from different sources. Let $C = (A, b)$ be the set of all inequality constraints. One source of constraints, denoted as C_{DRC}, is imposed by the design rules. The second source of constraints is imposed by high-level architectural requirements, which have to do with the global structure of the entire library, rather than individual cells. This includes the routing track constraints $C_{RouteTrack}$, the power line constraints $C_{PowerLine}$, the transistor size constraints C_{Size} and the port matching constraints $C_{PortMatch}$.

Our migration framework employs a two-pass strategy, as illustrated in Figure 3 to break the interdependence between different leaf cells. The tool accepts a library of datapath leaf cells with layout information from one process, and a specification of the library architecture. In the first pass, it analyzes the layout of each leaf cell, generates the design rule constraints, and drives an ILP solver under the geometric closeness objective function to arrive at a temporary migration solution of each cell. Note that the ILP problems are solved separately for different cells. In the second pass, the different architectural and circuit requirements are translated into linear constraints. Here, the temporary solution obtained in the first pass is exploited so that the new constraints relate variables originating from the same cell. The ILP solver is then called again to obtain the new, and final solution to accommodate the new constraints. Note again that the ILP problems are solved separately for each cells.

4 Datapath Migration

Design rule constraint generation is standard and readers are referred to [3, 4]. In this section, we focus on how architectural-level and circuit-level requirements are translated into linear constraints and used in our ILP framework.

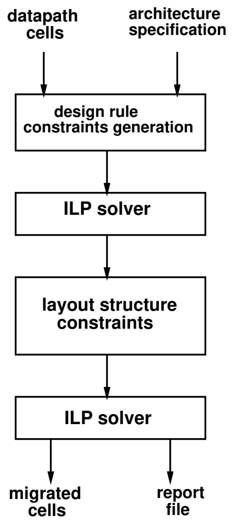


Figure 3. ILP-based migration framework.

4.1 Port Matching

The area efficiency in the datapath is achieved by the *tiling* strategy where important signals, such as controls and carries, are implicitly routed by abutment. This requires the ports of different cells to *match* exactly in position and size, as shown in Figure 4.

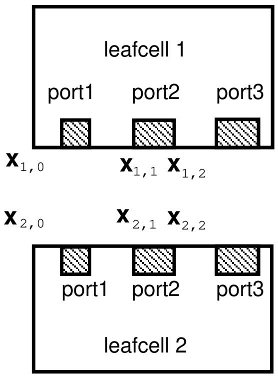


Figure 4. Port matching of leaf cells.

One naive way of translating the port matching constraints is to make the relative port position on the cell boundary and port widths equal for all matching cells. The constraints result from Figure 4 are then:

$$x_{1,1} - x_{1,0} = x_{2,1} - x_{2,0} \quad (5)$$

$$x_{1,2} - x_{1,1} = x_{2,2} - x_{2,1} \quad (6)$$

The problem of this approach is that the constraints bind variables from different cells together. Therefore they have to be migrated simultaneously. This may increase the number of variables in the underlying ILP solver substantially. With large datpath library, this approach quickly becomes infeasible.

We instead introduce a pair of constants, d and w to break the dependency between different cells:

$$x_{1,1} - x_{1,0} = d \quad x_{2,1} - x_{2,0} = d \quad (7)$$

$$x_{1,2} - x_{1,1} = w \quad x_{2,2} - x_{2,1} = w \quad (8)$$

We obtain d and w by taking the maximum value of $x'_{i,1} - x'_{i,0}$, and $x'_{i,2} - x'_{i,1}$ among all the leaf cells that have to be matched, where x' is the temporary solution we obtain in the first pass. This method effectively stretches the port whose distance to boundary can be smaller than the maximum value among all the leafcells, making all ports align together.

4.2 Routing Track Matching

Data signals in the datapath are always routed horizontally (perpendicular to control signals), and over-the-cell. Typically, they have to be aligned to a routing grid, for which the new migrated library may be different than the original. For example, for a leaf cell layout given in Figure 5, and a routing grid characterized by $\langle rs, ro, rw \rangle$, representing the routing pitch, offset and width respectively, the Y coordinate of the lower and upper edge of each routing track, y_{si} and y_{wi}, must satisfy constraints:

$$y_0 + ro + K \cdot rs = y_{si} \quad (9)$$

$$K \geq 0 \quad (10)$$

$$y_{wi} - y_{si} = rw \quad (11)$$

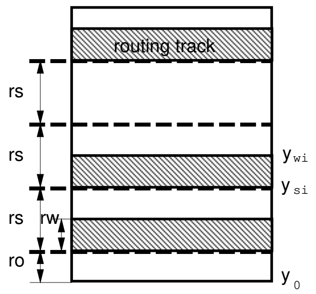


Figure 5. Cell with routing track for data signals.

4.3 Power/Ground Net Sizing and Transistor Sizing

There are two kinds of constraints involved with power/ground nets. First, power/ground ports of different cells have to match. This can be solved by the port matching method described earlier. Second, the width of the power/ground net needs to satisfy the architectural specification. Often, the width is determined by separate power/grid design methodology and layout migration has to faithfully follow the specification. These constraints are expressed as follows:

$$x_{i,2} - x_{i,1} = WP \tag{12}$$

where $x_{i,2}$ and $x_{i,1}$ are the two edges of the power/ground net and WP is the user specified width of power line.

Similarly, a circuit-level transistor sizing tool may determine an optimal transistor size that is different from the original layout, and it will rely on the migration tool to perform the change. After the identification of transistors in the layout, the sizing requirement can be expressed as an equality constraint. Note that we only expect modest change in the transistor size, and therefore mildest change in layout topology. For example, the introduction or elimination of transistor fingers is not needed.

A practical problem frequently encountered is that it is highly possible that the layout architecture specified by the user may lead to infeasible solutions. For example, the routing grid specification may conflict with existing design rules constraints. If the migration tool provides only a binary answer of failure, it is unlikely that the user can trace back to where the problem is and make a decision on where to change architecture or manually modify the layout at some specific location. It is instead highly desirable that the tool can provide some hints.

We introduce a new concept, called *soft constraints* to address this problem. Typically, routing track constraints, power/ground sizing constraints, and transistor sizing constraints will be introduced as soft constraints, in contrast to hard constraints such as design rule constraints. Consider a constraint of the form

$$\sum_j A_{ij} x_j \geq b_i \tag{13}$$

We introduce a positive variable, called the elastic variable e_i, that will be added to the inequality:

$$\sum_j A_{ij} x_j + e_i \geq b_i \tag{14}$$

$$e_i \geq 0 \tag{15}$$

$$\tag{16}$$

Since the values of e_i can be set arbitrarily large, the original constraint can be relaxed or *softened*, in other words, they do not have to be satisfied. This effectively enlarges the feasible solution space of the ILP problem.

On the other hand, we penalize those solutions that were not supposed to be feasible originally by adding them to the objective function with a large weighting factor. Combining (4), the new objective function becomes

$$\sum_j (R_j - L_j) + W \sum_i e_i \tag{17}$$

This way, if conflicting constraints exist, by looking for all non-zero elastic variables, users can easily pinpoint the wrong constraints and revise them accordingly.

5 Experimental Results

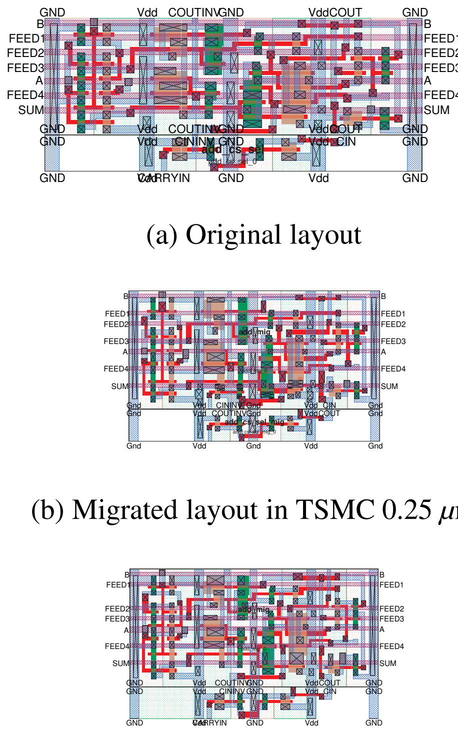
We implemented the migration tool on top of an IP-centric CAD infrastructure, called *ipsuite*. An open-source ILP solver is used as well. The migration tool itself is implemented by 10K lines of C code.

To test the effectiveness of our tool, we apply our tool on the low-power datapath library developed by Burd [1] at University of California, Berkeley. This library was based on SCMOS 1.2μm technology. Our targeted process is TSMC 0.25 μm and TSMC 0.18μm technologies.

Figure 6 (a) shows an one-bit adder constructed by abutting the *add* cell and the *add_cs_sel* cell in the library. Figure 6 (b) and (c) show the migrated layout output by our tool for TSMC 0.25 μm and 0.18μm technologies respectively. Note that all layouts are scaled to the same height for comparison. Different aspect ratios are apparent as the consequence of the non-linearity of design rule scaling.

Figure 6 shows the migration result of an 8-bit adder.

Table 1 gives more comprehensive results carried out on a SUNBlade 100 system running at 500 MHZ. Here, related cells that need port matching are migrated in groups. The third column demonstrates two important figures, *ASR* is the actual area scaling ratio achieved, as measured by $\frac{migrated\ layout\ area}{old\ layout\ area}$, and *TSR* is the linear area scaling ratio measured by $(\frac{migrated\ feature\ size}{old\ feature\ size})^2$, which as we know, direct layout shrinking with this ratiomay result in design rule violations. These two values are shown for both targeted technologies. The number of rectangles (including space) in the layout, and the total numbers of constraints generated are shown in the fourth and fifth column. The last two columns show the runtime spent on



(a) Original layout

(b) Migrated layout in TSMC 0.25 μm

(c) Migrated layout in TSMC 0.18μm

Figure 6. One-bit adder.

constraint generation (measured in seconds), and ILP solving (measured in minutes). As we can observe, the ILP solution dominates the computation time. Since our dual-pass framework allows us to run ILP one cell at a time, the total migration time is limited in minutes and is therefore tolerable.

6 Related Work

Automatic layout migration was among the oldest CAD problems investigated and a large body of research was carried out under the layout compaction problem. A good survey of layout compaction can be found in [3] and [4]. The compaction methodologies include: shear-line approach, virtual grid approach and constraint graph approach. Our migration tool extends the constraint graph approach, which represents constraints only as distance inequalities, into general linear formula, which

(a) (b) (c)

Figure 7. Migration of 8-bit adder (a)Original layout; (b) Migrated layout in TSMC 0.25 μm; (c) Migrated layout in TSMC 0.18μm.

are needed in some occasions.

The port matching problem, or sometimes referred to as pitch matching problem, was solved in the past by hierarchy compaction [6] [8]. This was achieved by first deriving the so-called port abstraction of each cell, which can be considered a simplification of the constraint graph for each cell, where constraints unrelated to the ports are removed and the longest path between ports are computed. The port locations of each cell are then solved by solving the combined port abstraction graph of the circuit that uses the cell to be migrated. The result is then set as constraints to drive the migration of each leaf cell. While this process might seem to be similar to our dual-pass strategy, it is not designed to migrate a library where a top-level circuit does not exists, and it can only perform minimum area compaction, which as argued before, is less important than other considerations.

The *minimum perturbation objective function* proposed in [5] was the first work that departed from the traditional optimization goal and argued to the importance of rewarding geometric similarity between the migrated layout and the original layout. However, the quantitative measure they develop for geometric similarity is asymmetrical and penalizes right edges in the X direction and upper edges in the Y direction and therefore is inferior to ours.

7 Conclusion

In conclusion, we have argued that migration technology is essential for the success of hard IPs in general, and datapath library IPs in particular. The successful migration of datapath IP needs not only a migration engine with new optimization goal of geometric closeness, but also a comprehensive framework to handle architectural requirements and practical considerations. Future work includes a more ambitious transistor-sizing capability where layout topology can be changed to insert transistor fingers, as well as extending the application to other IP libraries.

References

[1] T. Burd. Very low power cell library. Technical report, University of California, Berkeley, 1995.

[2] D. Chinnery and K. Keutzer. *Closing the gap between ASIC & custom*. Kluwer Academic Publishers, 2002.

[3] Y. Cho. A subjective review of compaction. In *22nd Design Automation Conference*, pages 396–404, 1985.

[4] D. G.Boyer. Symbolic layout compaction review. In *25th ACM/IEEE Design Automation Conference*, pages 383–389, 1988.

[5] F.-L. Heng, Z. Chen, and G. E.Tellez. A VLSI artwork legalization technique based on a new criterion of minimum layout perturbation. In *1997 International Symposium on Physical Design*, pages 116–121, 1997.

[6] C. Kingsley. A hiererachical, error-tolerant compactor. In *21st Design Automation Conference*, pages 126–132, 1984.

[7] J. K.Ousterhout. Corner stitching: A datastructuring technique for VLSI layout tools. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 87–100, 1984.

[8] C.-Y. Lo and R. Varadarajan. An o($n^{1.5}$logn) 1-d compaction algorithm. In *27st ACM/IEEE Design Automation Conference*, pages 382–387, 1990.

[9] C. Mead and L. Conway. *Introduction to VLSI Systems*. Addison Wesley, 1979.

Table 1. Experiment results.

Function unit	Cell name	ASR:TSR 0.25μm / 0.18μm	Number of polygons	Number of constraints	constraints generation run time (s) 0.25μm / 0.18μm	ILP solver run time (m) 0.25μm / 0.18μm
adder	add	0.07:0.04 / 0.04:0.02	452	56127 / 36866	134.9 / 149.5	18.5 / 19
	add_cs_sel	0.06:0.04 / 0.03:0.02	71	4288 / 3070		
	add_cs_0	0.04:0.04 / 0.03:0.02	11	225 / 223		
substractor	sub	0.07:0.04 / 0.04:0.02	479	36538 / 44535	136.7 / 114.4	20.6 / 19.8
	sub_cs_sel	0.06:0.04 / 0.03:0.02	72	3071 / 3071		
	sub_cs_1	0.03:0.04 / 0.015:0.02	11	229/229		
mux2	mux2	0.08:0.04 / 0.04:0.02	145	11797 / 7745	10.9 / 10.5	3.2 / 2.9
	mux2_cs1	0.06:0.04 / 0.03:0.02	52	4248 / 2203		
	mux2_cs2	0.06:0.04 / 0.027:0.02	61	4577 / 2279		
	mux2_cs3	0.07:0.04 / 0.03:0.02	104	7512 / 4771		
	mux2_cs4	0.06:0.04 / 0.027:0.02	149	11096 / 8518		
mux3	mux3	0.08:0.04 / 0.04:0.02	211	11768 / 11796	36.4/38.7	15.2 / 12.8
	mux3_cs1	0.06:0.04 / 0.03:0.02	247	14247 / 14243		
	mux3_cs2	0.06:0.04 / 0.03:0.02	248	14573 / 14574		
	mux3_cs3	0.07:0.04 / 0.03:0.02	293	17514 / 17512		
	mux3_cs4	0.06:0.04 / 0.03:0.02	331	21099 / 21098		
tribuf	tribufl	0.07:0.04 / 0.03:0.02	151	7880 / 7880	7.8 / 8.1	1.1 / 1.7
	tribufl_cs1	0.07:0.04 / 0.03:0.02	59	2641 / 2641		
	tribufl_cs2	0.05:0.04 / 0.03:0.02	105	5856 / 4723		
	tribufl_cs3	0.06:0.04 / 0.03:0.02	144	7164 / 7163		
register	rf0	0.06:0.04 / 0.03:0.02	66	2748 / 2748	3.7 / 3.9	0.6 / 0.6
	rf1	0.06:0.04 / 0.03:0.02	66	2756 / 2756		
	rf01_cs1	0.07:0.04 / 0.03:0.02	92	3516 / 3517		
	rf01_cs2	0.07:0.04 / 0.03:0.02	92	3584 / 3584		
	rf01_cs3	0.06:0.04 / 0.03:0.02	92	3464 / 3464		
shifter	shcs1	0.06:0.04 / 0.03:0.02	87	3957 / 3957	8.9 / 8.7	0.8 / 0.8
	shl1bit	0.04:0.04 / 0.02:0.02	88	3921 / 3921		
	shl1end	0.05:0.04 / 0.02:0.02	81	3478 / 3478		
	shl1lsb	0.05:0.04 / 0.02:0.02	130	6539 / 6539		
inverter	invs	0.04:0.04 / 0.02:0.02	57	2029 / 2029	1.2 / 1.3	0.2 / 0.2
	invm	0.05:0.04 / 0.03:0.02	80	3098 / 3098		
random logic	and2blp	0.06:0.04 / 0.03:0.02	112	5940 / 5940	6.4 / 6.6	3.3 / 3.3
	nand2lp	0.05:0.04 / 0.03:0.02	83	4015 / 4015		
	nandand3lp	0.07:0.04 / 0.03:0.02	113	6706 / 6706		
	or2blp	0.06:0.04 / 0.03:0.02	99	5319 / 5319		
	xnor2lp	0.06:0.04 / 0.03:0.02	132	8075 / 8075		
	xor2lp	0.06:0.04 / 0.03:0.02	132	8003 / 8003		
pipeline tspcr register	tspcr	0.07:0.04 / 0.03:0.02	137	7580 / 7582	14.7 / 14.9	5.7 / 6.1
	tspcr_fb	0.08:0.04 / 0.04:0.02	173	10637 / 10637		
	tspcr_cs1	0.05:0.04 / 0.03:0.02	49	1792 / 1792		
	tspcr_cs2	0.05:0.04 / 0.03:0.02	52	2003 / 2003		
	tspcr_cs3	0.06:0.04 / 0.03:0.02	91	3792 / 3792		
	tspcr_cs4	0.06:0.04 / 0.03:0.02	103	4602 / 4602		
	tspcr_csi1	0.06:0.04 / 0.03:0.02	1033	980 / 980		
	tspcr_csi2	0.06:0.04 / 0.03:0.02	34	1110 / 1110		
	tspcr_csi3	0.05:0.04 / 0.03:0.02	96	4595 / 4595		
	tspcr_csi4	0.05:0.04 / 0.03:0.02	126	6008 / 6008		